FI MU

Faculty of Informatics
Masaryk University Brno

Parameter Identification and Model Ranking of Thomas Networks

by

H. Klarner
A. Streck
D. Šafránek
J. Kolčák
H. Siebert

FI MU Report Series **FIMU-RS-2012-03**

 August 2012

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**

Parameter Identification and Model Ranking of Thomas Networks*

H. Klarner, H. Siebert
Freie Universität Berlin
Berlin, Germany
Hannes.Klarner@fu-berlin.de

A. Streck, D. Šafránek, J. Kolčák
Masaryk University
Brno, Czech Republic
safranek@fi.muni.cz

November 1, 2012

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 the most optimal ones; (iii) we propose a way of visualising parametrizations dynamics wrt time-series data. The methodology is validated on a rat neural development case study; (iv) finally we provide description of developed algorithms and evaluation of their performance.

1 Introduction

Discrete modeling frameworks are commonly used in systems biology as a tool that assists in revealing regulatory mechanisms found in biological networks [15, 11, 24]. A widely used formalism for gene regulatory networks is that of R. Thomas et al. [25] (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

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

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, 18, 4, 23].

Although discrete regulatory models are very abstract, parameters determining the behaviour 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 [16] for review).

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 temporal logic. Naive (bottom-up) approaches [4, 13] repeat the procedure of 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 behaviour 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. Even though the algorithms performed well in the average case, their properties could not have been guaranteed.

In [13], 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 using preliminarily 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 [13] introducing a *classification of acceptable parametrizations* based on optimal satisfaction of

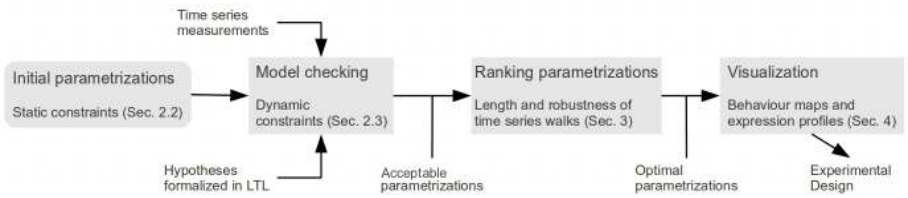


Figure 1: Parameter identification workflow.

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 a visualisation approach allowing quick and intuitive understanding of the behaviour generated by different parametrizations. The workflow is outlined 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 behvaiour mapper. 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 behaviours (in model checking terms: generation of all counterexamples for a given time series formula) and parameter ranking. The behaviour mapper extracts portion of the transition graph relevant to the time series employed and plots it in an intuitive manner.

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 rat central nervous system is considered in Sect. 5. Further

information on implementation and performance are provided in Sect. 6. The formal description of algorithms employed concludes the paper.

2 Background

Our contributions benefit from the properties of the modeling workflow in question. This system consists of three consecutive steps:

1. Definition of interaction graph using a generalized version of Thomas formalism.
2. Specification of the parametrization space using regulatory constrains.
3. Verification of models by the method of colored LTL model checking.

In this section we provide formal definitions associated with each of these steps.

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 [25].

2.1.1 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 $V = \{v_1, \ldots, v_n\}$ 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 behaviour 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 $\theta(uv) = (t_1, \ldots, t_k)$ to each interaction $uv \in E$. Each interaction may have a different number $1 \leq k$ of thresholds. The thresholds must be ordered: $t_1 < \cdots < t_k$ and within the non-zero activities of the regulator: $1 \leq t_1$ and $t_k \leq \rho(u)$.

The interaction thresholds $\theta(uv) = (t_1, \ldots, t_k)$ of an interaction uv divide the activities of u into $k + 1$ intervals $[0, t_1 - 1], [t_1, t_2 - 1], \ldots, [t_k, \rho(u)]$ 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.

2.1.2 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.

2.1.3 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:

$$F : X \to X, \;\; x \mapsto (P^{\omega_1}_{v_1}, \ldots, P^{\omega_n}_{v_n}),$$

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*: $y_v - x_v = \mathrm{sign}(F(x)_v - x_v)$. Here $\mathtt{sign}$ 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

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

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 behaviours 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.

2.2.1 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 [13, Def. 2.9]). Certain edge labels have already been used successfully in case studies of D. Thieffry (see e.g. [21],[10]) and also implemented in analysis tools [20, 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 $\{+, -, \mathrm{mon}+, \mathrm{mon}-\}$, 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.

2.2.2 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 behaviour 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 $(m^1, \ldots, m^k)$ 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 $(m^1, \ldots, m^k)$ if and only if there is a state $x \in X^0$, such that the LTL specification

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

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} [13] 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 behaviour 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 $BA(\mathcal{R}, \varphi)$

2.3.1 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 $BA(\varphi) \equiv (\Sigma, X_A, X_A^0, \bar{\delta}_A, F_A)$: $BA(\mathcal{R}, \varphi) := (\mathcal{P} \times \Sigma, X, X^0, \delta, F)$ 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.

2.3.2 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 $cl^{\hat{\mathcal{P}}}_{\hat{X}}(x) := \{P \in \hat{\mathcal{P}} \mid \exists \hat{x} \in \hat{X} : \hat{x} \xrightarrow{P}{}^{*} x\}$. Using this mapping, the CMC procedure can be described as follows:

For each $x \in F$:

(1) Compute coloring $reach_x \equiv cl^{\mathcal{P}}_{X_0}(x)$ reaching accepting state x.

(2) Compute coloring $cycle_x \equiv cl^{reach_x}_{\{x\}}(x)$ 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 2.2.2, 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 behaviour 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 behaviour. 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 behaviour 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, e.g. if there is only one path between two states in the state transition graph, the behaviour 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.

Recall that a time series is sequence of measurements $(m^1, \ldots, m^k)$ and a state transition graph reproduces the time series if it contains a finite walk $(x^i)_{1 \leq i \leq r}$, such that there

is a mapping $M : [1, k] \to [1, r]$ that is

$$\begin{aligned} &\text{ordered:} && i < j \implies M(i) \leq M(j) \\ &\text{and correct:} && x^{M(i)} \in m^i. \end{aligned}$$

A walk satisfying the given properties is called a *time series walk*.

In the following, we focus only on a subset of possible time series walks. Let P be a parametrization and and $(m^1, \ldots, m^k)$ a time series. A time series walk $\omega = (x^i)_{1 \leq i \leq r}$ is called a *simple time series walk*, if there is an ordered and correct mapping $M_\omega : [1, k] \to [1, r]$ that satisfies $M(1) = x_1$ and $M(k) = x_r$. Additionally, ω is cycle-free between two subsequent measurements, i.e., for all $l \in \{1, \ldots, k-1\}$ we have $x^i \neq x^j$ for all indeces $i \neq j$ with $M(l) \leq i, j < M(l+1)$. Note that the mapping M_ω is generally not unique. We denote the set of all simple time series walks with $\widetilde{W}_P$.

It is easy to see that every time series walk contains a simple time series walk that can be obtained by eliminating spurious path segments. However, the additional conditions ensure that the set $\widetilde{W}_P$ is finite since the state space is finite and the length of a simple time series walk is bounded by a term depending on the cardinality of state space and the number of measurements in the time series.

To define the robustness of a walk we take a local view point and start by defining the robustness in a given state of the walk. Since the objective is to reproduce the time series, we basically test whether deviation from the path potentially still yields a simple time series walk. That is, if we choose in a given state of the walk a successor of that state that does not coincide with the next state of the walk, we see whether we can continue this new walk in a way that results in a simple time series walk.

To make this idea more precise, let $\omega = (x^i)_{1 \leq i \leq r}$ be a simple time series walk and $j \in \{1, \ldots, r\}$. A *valid successor of* x^j is a successor $y \in N^+(x_j)$ of x^j in the state transition graph such that there exists a simple time series walk $\omega' = (y^i)_{1 \leq i \leq r'}$ with $x^i = y^i$ for all $1 \leq i \leq j$ and $y^{j+1} = y$. Note that we do not demand that the mappings M_ω and $M_{\omega'}$ coincide in any way apart from the requirements concerning the start and end vertex of the walk. We denote the set of all valid successors of x^j with $\mathcal{S}_{valid}(x^j)$.

Note that, since we are generally dealing with non-deterministic systems, we had to make a choice on the strictness of the condition characterizing valid successors. Here, we chose to demand the existence of a path that compensates the perturbation, but it would also be possible to require that the perturbation is not capable at all to produce a behaviour not in agreement with the time series. Since the non-deterministic mod-

eling approach employed here often produces spurious paths that are not biologically realistic, we decided to utilize the weaker condition.

Denoting the cardinality of a set Q by $\mathrm{card}(Q)$, we define the robustness in state x^j of the time series walk as

$$0 < \mathrm{Rob}(x^j) := \frac{\mathrm{card}(\mathcal{S}_{\mathrm{valid}}(x^j))}{\mathrm{card}(N^+(x^j))} \leq 1 .$$

The robustness of the simple time series walk ω is then defined as

$$0 < \mathrm{Rob}(\omega) := \prod_{i=1}^{r} \mathrm{Rob}(x^i) \leq 1 .$$

If a walk is deterministic in the sense that all states of the walk have out-degree one, then the robustness of the walk is one, in agreement with our interpretation of robustness with respect to the impact of time delay perturbations. A high robustness, even robustness one, is still possible for non-deterministic systems, if perturbations leading to a deviation from the original walk can still be completed to a simple time series walk. That is, the system is capable of correcting the perturbation. Low robustness indicates that deviation from the walk often result in a situation that does not allow for correction in the sense of measurement recovery.

It is easy to see that a single state of the walk with a low robustness can have a strong impact on the robustness of the entire walk, illustrating the point that naturally there is more information to be had when considering not only the global robustness of the walk but also the local robustness of the states. We could have lessened the impact of the local robustness by defining the walk robustness via, e.g., the mean instead of the product of state robustness values. However, then it would be possible for a walk to exhibit very high robustness although it might be extremely susceptible to perturbations in particular states.

We now extend the notion of robustness from a single walk to the model by averaging robustness values of all simple time series walks:

$$0 < \mathrm{Rob}(P) := \frac{1}{\mathrm{card}(\widetilde{W}_P)} \sum_{\omega \in \widetilde{W}_P} \mathrm{Rob}(\omega) \leq 1 .$$

Again, we see that a deterministic system has robustness one. More general, every system such that every finite random walk starting in a state consistent with the first measurement is a time series walk has robustness one. Low robustness reflects that many of the possible simple time series walks cannot recover the time series after perturbations of the time delays.

Often, robustness is considered not with respect to temporal processes but rather with respect to perturbations in state space. The definitions given above can be adjusted to fit this notion. Rather then considering the successors in the definition of robustness in a state x^j, we would then have to consider, e.g., states in the 1-neighborhood of x^j, i.e., the states that have Hamming distance 1 to x^j. The valid states in the neighborhood would be the states such that a walk consisting of ω up to x^j joined with a walk starting in the considered neighborhood state reproduces the time series. The last condition would also necessitate a definition of when a union of walks reproduces the time series.

The notion of robustness presented above is very descriptive of the property we want to evaluate in the models, however, computational calculation is hampered by its comprehensive nature. In the following we present a simplification of this notion that allows for efficient implementation. The first step is to phrase the robustness of a state of a walk in terms that allow for a purely local verification. We achieve this by dropping the validity evaluation of the successors and simply consider the out-degree, i.e., we use the inverse of the out-degree as measurement of robustness for each state. This results in the notion of robustness coinciding with 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

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

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

Two further simplification steps are utilized for the definition of robustness of a parametrization P. First, we do not consider the entire set of simple time series walks, but only the set of shortest walks SW_P defined in the preceding section. Second, we average the robustness values of all considered walks by the cardinality of the set of states consistent with the first measurement m^1. The resulting definition is

$$\mathrm{Robustness}(P) := \frac{\sum_{w \in SW_P} (\mathrm{Prob}(w))}{card(m^1)}.$$

Note that the averaging using m^1 still ensures that $\mathrm{Robustness}(P)$ has at most value one due to the definition of $\mathrm{Prob}(w)$ based on the out-degree. Walks starting in the same state x^1 and reproducing the time series can be seen as branching off of each other leading automatically to increasing out-degrees and thus decreasing probability values. This ensures that the sum of probabilities over the set of such walks does not exceed the value one.

This simplified notion of robustness is a good starting point for analysis since it still distinguishes parametrizations that reproduce the time series with low ambiguity. In addition, it is easy to formalize and compute. The more involved definition, in contrast, is based not only on the out-degree of a state of a time series walk, but differentiates and weights whether the different successors of the state are themselves states of a time series walk. Obviously, it captures the intuitive understanding of robustness much better. Efficient methods for computing this more involved notion of robustness and the development of robustness notions of intermediate complexity will be a focus of 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 imposed edge constraints.
3. Compute the set of acceptable parametrizations based on an LTL formula.
4. From the set, select parametrizations having the globally minimal cost.
5. Finally, select parametrizations with the globally 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 [13, Sec. 5.1] the focus is on deriving consensus edge labels.

Here we propose a method for construction of *behaviour maps* that visualize the transitions within the state space of *a set of acceptable parametrizations*, highlighting agreement between parametrizations. The information whether certain state transitions are shared by the walks, if present, 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 shortest time series walks easy to assess.

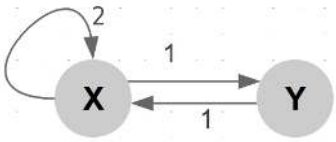
Let *SW* be any finite set of shortest time series walks of $(m^1, \ldots, m^k)$. In each walk we mark the measurements $1, \ldots, k$. We lay all the transitions of these walks horizontally and align for every $1 \leq i \leq k$, the states marked as the i^{th} measurement vertically. This way we can interpret the horizontal axis as a discrete time axis, progressing from earlier (left) to later (right).

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 equal if they appear between the same pair of measurements. Note that acyclicity between measurements is ensured as the path containing cycle between two measurements is surely not the shortest one and therefore is not present in the set *SW*.

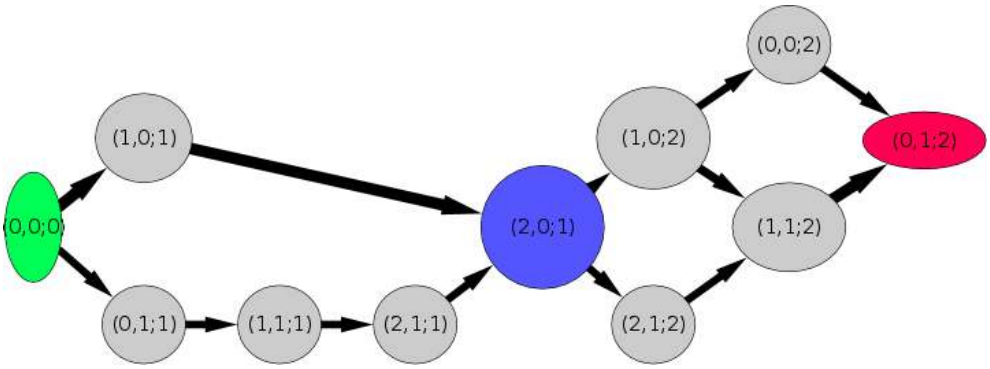
If time series walks of more than one parametrized structure are plotted at once, we scale nodes and edges of the map to highlight those appearing more often. Note that sizes of incoming edges usually do not correspond to size of outgoing ones for a single node. This property is satisfied only in the trivial case i.e. when the state has only one predecessor and one successor.

In Fig. 2 we provide an example of such a plot. For this example we have constructed a simple regulatory network as depicted in Fig. 2a. The behaviour map of this network wrt time series $m^1 = \{(0, 0)\}, m^2 = \{(2, 0)\}, m^3 = \{(0, 1)\}$ is given in Fig. 2.

Colored states represent points of measurements. The stroke of a transition scales linearly with occurence of the transition in the set *SW*. Shape of a state expresses two properties - width of the ellipse grows with the in-degree of the state and its out-degree is depicted in the same manner by its height. States are labeled with comma separated list of integers corresponding to current activation levels of all components. The last, semicolon-separated integer marks current measurement interval.



(a) Sample interaction graph.



(b) Plot of the shortest time series walks.

Figure 2: Visualisation example.

5 Development of the central nervous system in rats

We have applied our methods to the model of development of the central nervous system (CNS) in rats. We base our results on data published by two groups. First, in 1998, Wen et al. ([27]) recorded the gene expression patterns of more than a hundred signaling genes in different development tissues. They observed that the patterns cluster into four "waves" of similar activity. Although interesting in itself, they remarked that this does not explain "the nature of genetic information flow" and that "simple models may be required to conceptualize" it.

Consequently, in 2001, Whade and Hertz ([26]) suggested a number of abstractions that lead to a model consisting of only four differential equations. Each equation describes the activity of one of the gene clusters. The model consists of 24 parameters, 16 of which determine the regulatory effects between the clusters. The parameters were then fitted to the expression patterns by a genetic algorithm that returned average and significance values. They concluded with a gene cluster interaction graph we use in our study.

5.1 Formulating the model

The aim of this case study is to find and rank all boolean models that are

1. compatible with the interaction graph and interaction strengths of Whade and Hertz, and

2. can reproduce the expression patterns of Wen et al.

The results will then be used to describe, in logical terms, the regulatory control between the gene clusters.

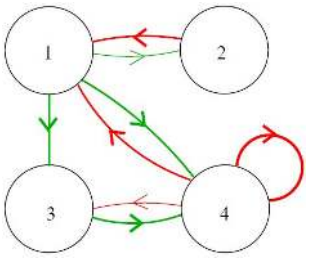
The interaction graph of Whade and Hertz, including interaction strengths, is depicted in Fig. 3a. In the graph positive effects are depicted by green and negative effects by red arrows. Additionally, thick lines indicate strong effects (large absolute parameter values) while thin lines indicate weak effects.

To tackle the first step, we have to discuss ways of incorporating the *strength of an interaction* into the modeling process. We could, for example, restrict the admissible logical functions of targets of strong interactions to *canalizing* functions. The value of canalizing functions is completely determined when a strong interaction is effective, reflecting the dominance (or strength) of those interactions. However, this approach goes beyond the methodology defined in the previous sections.

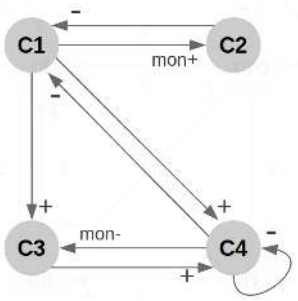
Another approach is to enforce different static constraints for strong and weak interactions. In Fig. 3b we have translated strong interactions into constraints requiring observability, as defined in Sec. 2.2.1, while weak interactions are merely monotonous and may not be observable in the resulting parametrizations. In this interpretation of interaction strengths we therefore allow the effect of weak interactions to be negligible.

The second step requires us to interpret the quantitative data of Wen et al., depicted in Fig. 3c, in qualitative, binary terms. We binarized the data using the approach of [22] as implemented in the BoolNet package [17] for the R software [19].

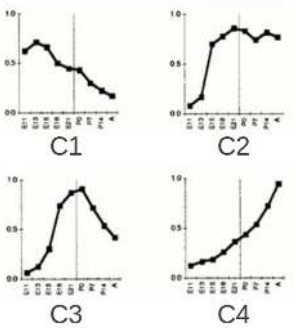
The original time series consists of 9 measurements. However, these are in some cases mapped to the same binary vector. After removing the duplicate subsequent measurements we obtained a time series of five measurements as depicted in Fig. 3d.



(a) Regulatory graph



(b) Static constraints.



(c) Quantitative time series.

	C1	C2	C3	C4
1.	1	0	0	0
2.	1	1	0	0
3.	1	1	1	0
4.	0	1	1	1
5.	0	1	0	1

(d) Qualitative time series.

Figure 3: CNS development data.

Even though the structure provided in [26] is quite strict, its parametrization is still not obvious - there are 162 possibilities of how kinetic parameters may be aligned. From these 108 result in a transition graph containing the required time series walk. From the point of standard analysis using a temporal logic formula, these 108 are completely equal.

5.2 Optimal parametrizations.

The minimum cost among the 108 compatible parametrizations is 6. In fact, every parametrization among the 108 has a time series walk of this cost. The robustness ranges from roughly 1% to 25%, whereas the value of 25% is attained by 2 parametrizations that we consider to be optimal in this case. The parameter values of each of the two are given in Fig. 4.

C1	P	C2	P	C3	P	C4	P
∅	1	∅	1	∅	**0,1**	∅	0
C2	1	C1	1	C1	1	C1	0
C4	1			C4	0	C3	1
C2, C4	0			C1, C4	1	C4	0
						C1, C3	1
						C1, C4	0
						C3, C4	0
						C1, C3, C4	1

Figure 4: The 3 optimal parametrizations with cost 6 and robustness 25%. Differing values are bold.

Since the CNS model consists only of binary components, we can convert every parametrization into boolean functions - one for each component. We have therefore converted both parametrizations into following boolean functions:

$$\begin{aligned} &C1 := \neg(C2 \wedge C4) \\ &C2 := 1 \\ &C3 := C1 \text{ or } C3 := C1 \vee \neg C4 \\ &C4 := C3 \wedge (C1 \vee \neg C4) \end{aligned}$$

The ambivalence between the optimal parametrizations rises from the insufficiency in the amount of data provided - with more precise time series or somehow more detailed specification we might be able to pick only one of the two, but since there is no step in the time series where both the regulators of C3 are not present, there is not way how to determine the optimal behaviour for such a case. Motivated by the *Occam's razor* principle, we would suggest taking the parametrization with $C3 := C1$ as the single most optimal one.

With this qualitative model, we can now attempt to "conceptualize the flow of genetic information" as Wen et al. demanded:

- The genes of cluster C1 are only inhibited when both clusters C2 and C4 are active. This logical conjunction suggests either the formation of complexes of some kind between proteins of C2 and C4, or additive inhibitory effects, where neither

proteins of C2 or C4 alone are sufficient to inhibit the expression of C1 proteins fully.

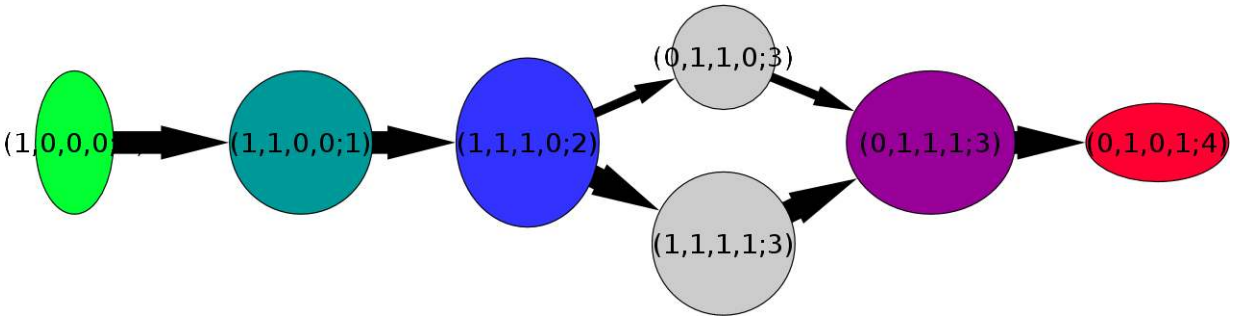
- C2 protein synthesis is autonomous, i.e. independent, of the activity of proteins from the other clusters. This suggests that the second wave may occur earlier (before C1) or later (after C3) without disturbing the CNS development.
- Proteins of C3 require sufficient activities in C1-proteins.
- C4-proteins require sufficient activity levels of C3-proteins for synthesis (when $\neg$C4 is true). After synthesis, in order to sustain high levels of activity, additional presence of C1-proteins is required.

5.3 Visualization.

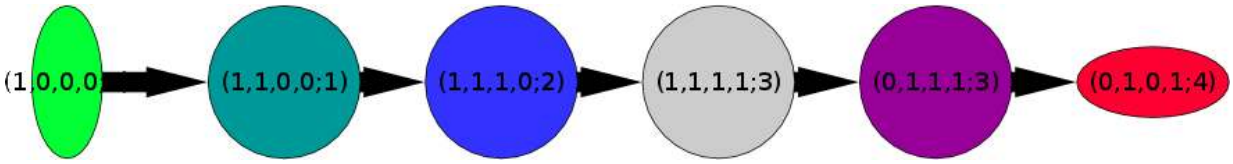
To give a better picture of the behaviour the models exhibit alongside the time series path, we present two behavioural maps.

The one in Fig. 5a depicts transitions recorded from all the parametrizations that were compatible with the time series. As can be seen the time series walk is quite straightforward which, as we will show later in this section, may be a sign of a well-formed model.

The second map is built from transitions belonging to the two parametrizations that were evaluated as optimal. In this case the map is even simpler since the path between the only two consecutive measurements that differ in more than a single value is now deterministic. Since this map depicts behaviour of models with the highest robustness, we conclude that in this case the metric indeed filtered the models which were less deterministic in their behaviour alongside the path given by the time series.



(a) Complete behavioural map



(b) Optimal behavioural map

Figure 5: CNS behvaioural maps.

5.4 Reverse time series experiment

Our metrics are based on the hypotheses that models with better ranking i.e. higher robustness or lower cost are more compliant with their respective time series and therefore impose more reasonable behaviour. Since the interaction graph we are using in our case study is already heavily restricted due to edge constrains we took from the study of Whade and Hertz, we expected the set of parametrizations allowing for reproduction of the time series to be quite felicitous. The behaviour map in Figure 5 as well as high resulting robustness suggests that our expectations were legitimate.

Complementarily, it seems reasonable to expect that a time series that is in conflict with the anticipated behaviour will cause the possible models to score poorly. To test this assumption, we have conducted an experiment using the rat model with the time series having order of its measurements reverted. Such change basically means inversion in the purpose of the network - from the growth we obtain decay. A robust model should forbid such a behaviour or at least restrict probability of its occurrence significantly.

This expectations were correct as can be seen from the comparison of the best scores of parametrizations when tested with the original and the reverted time series:

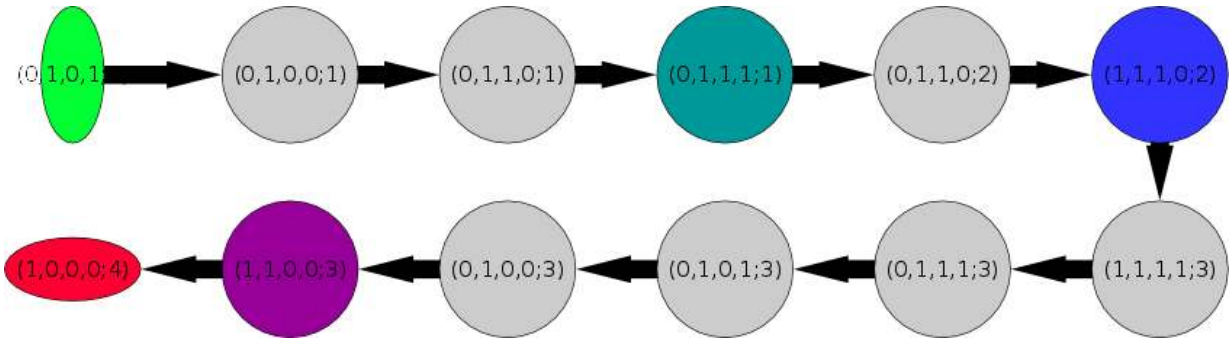


Figure 6: Behavioural map for all parametrizations.

	parametrizations count	lowest cost	highest robustness
original	108	6	25%
reverted	81	12	0.78%

It can be seen, that our expectations were met. Even though the decrease in size of the set of feasible parametrizations is not that significant, the ranking has changed greatly. It is also worth noting that unlike in the case of the original time series, the parametrization with the lowest cost is actually not the one with the greatest robustness.

To provide a more detailed picture, in Fig. 6 we also present a behaviour map for the parametrization having the highest robustness from those with the cost of 12. Mainly between the third and fourth measurement one can see that the model must undergo vast amount of changes just to switch off a single component, which behaviour is probably not natural.

6 Implementation and evaluation

In this section we briefly describe methodology of synthesis and analysis as well as the 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.

4. Plotting - finally, for the time series walks produced in the previous step we use the BehaviourMapper tool. The tool creates a behaviour map for a given Parsybone output file. Such a map can be than viewed using the Cytoscape [6] tool.

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 [14]). 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.

6.2.1 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.

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

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* $\mathtt{Code} : 2^{\mathcal{P}} \to \{0,1\}^{|\mathcal{P}|}$ where $\mathtt{Code}(\mathcal{P}') = b_1 b_2 ... b_{|\mathcal{P}|}, \forall i (b_i = 1 \Leftrightarrow P^i \in \mathcal{P}')$. 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 $\mathtt{Code}(\{P | x \xrightarrow{P} x'\})$.

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

6.2.2 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 [14]. The most important is the procedure of splitting.

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).

6.2.3 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.

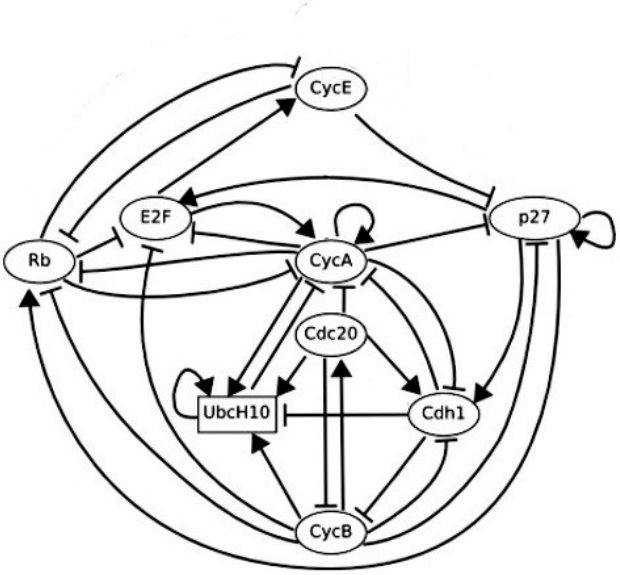
Each independent worker does its own parsing and pre-computation and then goes through the procedure of parameter identification with a subset of parametrization space that is disjunctive with subsets of other workers.

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

In this section we present performance measurements of our tool using two different models. First we evaluate overall performance on a model of mammalian cell cycle with more than half a billion parametrizations - this evaluation has mainly the purpose of showing what is actually computable. In the second part we use the bacteriophage model from Section 5 which is quite fast to compute and show that even on such a small model our algorithm scales well with number of processes.

6.3.1 Mammalian cell cycle



(a) Regulatory graph of Mammalian cell cycle with edge constraints.

Rb	E2F	CycE	CycA	p27	Cdc20	Cdh1	UbcH10	CycB
0	0	0	0	0	1	1	1	0
0	1	0	0	0	0	1	1	0
0	1	1	0	0	0	1	0	0
0	1	1	1	0	0	1	0	0
0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	1	1
0	0	0	1	0	1	0	1	1
0	0	0	0	0	1	1	1	0

(b) Tested time series.

Figure 7: Mammalian cell cycle - known data.

To test capabilities of our algorithm, we had it analyze a model of mammalian cell cycle [10] with 9 components depicted in Figure 7a. For this model we have defined partial parametrizations given as valuations of following logical formulas:

- $\mathrm{CycA} := \overline{\mathrm{Rb}} \wedge \overline{\mathrm{Cdc20}} \wedge \overline{\mathrm{Cdh1} \wedge \mathrm{UbcH10}} \wedge (\mathrm{E2F} \vee \mathrm{CycA})$
- $\mathrm{UbcH10} := \overline{\mathrm{Cdh1}} \vee (\mathrm{UbcH10} \wedge (\mathrm{Cdc20} \vee \mathrm{CycA} \vee \mathrm{CycB}))$

reducing size of parametrization space to final number of $675,584,064$ parametrizations. As a guide for the analysis we have used a time series with 8 measurements as depicted in Figure 7b.

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 Figure 8. 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

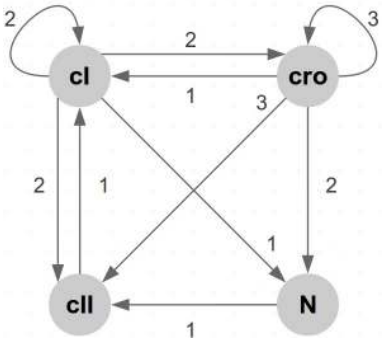
Figure 8: Results of distributed analysis of Mammalian cell cycle.

As can be seen, there are slight differences in run times up to 20%, some of which probably has been caused by background noise of the server. Other than that, we can see that parametrizations space has been partitioned to sets with almost identical numbers of acceptable parametrizations.

6.3.2 Bacteriophage

To demonstrate the improvement in performance since the old version, we have used an another, smaller example - the network of Bacteriophage λ infection [24]. Since our old tool has many functional restrictions, i.e. absence of edge constrains interface, we had to use the model in a very general form as depicted in Fig. 9a. With the model we have conducted analysis using the time series described in Fig. 9b marking the lysogenic fate of the cell.

We have also imposed some restriction on the initial set of parametrizations, obtaining the initial parametrizations space of $589,824$ parametrizations, out of which $90,148$ were acceptable. This model has then been analysed five times using each version of the tool. Analysis using the old version took on the average 967 seconds and used at



(a) Regulatory graph of bacteriophage λ with edge constraints.

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

(b) Lytic time series. The last three measurements indicate an oscillation.

Figure 9: Bacteriophage λ - known data.

max 50 MB of RAM, whereas the new version needed only 6 seconds and did not need more than 3 MB RAM.

Not only is our new tool usually significantly faster, it also provides almost linear speedup - there is some overhead in pre-computation that must be conducted by each of the processes, but this procedure is very fast - for example in analysis of mammalian cell cycle it takes less than second which is absolutely insignificant in comparison with tens of hours of following model checking.

To demonstrate scalability we have analyzed the bacteriophage model using up to 8 independent processes. In Fig. 10 we show average runtime of all the processes used. The resulting numbers are again produced as an average of three independent experiments conducted on the same platform. As can be seen from the graph our algorithm scales linearly wrt number of processes used.

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

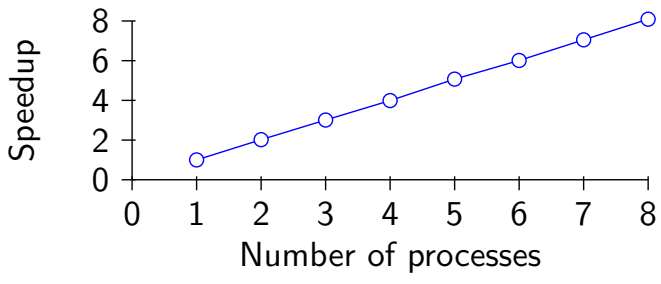


Figure 10: Scalability evaluation using the Bacteriophage λ model.

7 Algorithms

In Section 6.2.1 we present a method for concise representation of a parametrization space. This representation further allows for computation of all necessary operations on multiple parametrizations at once, using only fast bitwise operations on integers. Using this operations, we can compute all forms of analysis presented in this article in three successive steps:

1. coloring procedure and computation of the cost value,
2. time series walks search,
3. computation of the robustness value.

In this section we present methods for all of above. Each of these is based on the idea of *coloring* - labeling of the transition graph with subsets of parametrization space, which is described in the Section 2.3.2. To correspond with definition of function $\mathtt{Code}$ in Section 6.2.1 we will represent this labeling as a word from the language $\{0,1\}^{|\mathcal{P}|}$, where $\mathcal{P}$ is a parametrization space of some regulatory network. For a state x of the transition graph of the network we denote the coloring of the state as a word $\mathtt{Col}_x \in \{0,1\}^{|\mathcal{P}|}$. We also employ the usual notation $\mathtt{Col}_x(n), 1 \leq n \leq |\mathcal{P}|$ for an n-th letter of the word $\mathtt{Col}_x$, which corresponds to the n-th parametrization in the $\mathcal{P}$, denoted $\mathcal{P}(n)$. For a coloring of a state x, $\mathtt{Col}_x(n) = 1$ marks the presence and $\mathtt{Col}_x(n) = 0$ the absence of the parametrization $\mathcal{P}(n)$.

We also need to extend a semantics of bitwise operators to apply to such words. For words $w_1, w_2 \in \{0,1\}^{|\mathcal{P}|}$ and number $n : 1 \leq n \leq |\mathcal{P}|$ we use the following operations:

- $w_1 \& w_2 \in \{0,1\}^{|\mathcal{P}|}$, $(w_1 \& w_2)(n) = 1$ *iff* $w_1(n) = 1$ and $w_2(n) = 1$,
- $w_1 | w_2 \in \{0,1\}^{|\mathcal{P}|}$, $(w_1 | w_2)(n) = 1$ *iff* $w_1(n) = 1$ or $w_2(n) = 1$.

7.1 Coloring

According to the method presented in Section 2.3 a regulatory network $\mathcal{R}$ with parametrization P satisfies some property described by a Büchi automaton $\mathtt{BA}(\varphi)$ if and only if their parametrized product $\mathtt{BA}(\mathcal{R}, \varphi)_P$ contains a path from a initial to a final state and a cycle containing that state. For a proof of this claim refer to [12]. Here we restrict ourself to a method for computing the exact set of parametrizations that satisfy this condition.

Algorithm 1 Color

Require: X_0 the set of initial states from X, $\mathcal{P}$ the initial coloring

Ensure: $\mathtt{Col}$ is a coloring of X from initial states X_0

$\mathtt{Updates} \leftarrow X_0$
$\mathtt{Updates}' \leftarrow \emptyset$
$\mathtt{Col} \leftarrow \emptyset$
for all $x \in X_0$ **do**
$\mathtt{Col}_x \leftarrow \mathtt{Code}(\mathcal{P})$
end for
while $\mathtt{Updates} \neq \emptyset$ **do**
while $\mathtt{Updates} \neq \emptyset$ **do**
$x \in \mathtt{Updates}$
$\mathtt{Pass}(x, \mathtt{Col}, \mathtt{Updates}, \mathtt{Updates}')$
$\mathtt{Updates} \leftarrow \mathtt{Updates}/\{x\}$
end while
$\mathtt{Updates} \leftarrow \mathtt{Updates}'$
$\mathtt{Updates}' \leftarrow \emptyset$
end while

Algorithm 2 $\mathtt{Pass}(x, \mathtt{Col}, \mathtt{Updates}, \mathtt{Updates}')$

Require: $\mathtt{Updates}$ set of states scheduled for an update, $x \in \mathtt{Updates}$, $\mathtt{Col}$ coloring

Ensure: $\mathtt{Col}$ is coloring after updating all neighbours of x, $\mathtt{Updates}'$ contains all states that have received a new coloring

$\mathtt{Neigh} \leftarrow \{y | \exists P(x \xrightarrow{P} y)\}$
for all $y \in \mathtt{Neigh}$ **do**
$P_{new} \leftarrow \mathtt{Col}_x \& \mathtt{Code}(\{P | x \xrightarrow{P} y\})$
if $(P_{new} | \mathtt{Col}_y) \neq \mathtt{Col}_y$ **then**
$\mathtt{Updates}' \leftarrow \mathtt{Updates}' \cup \{y\}$
$\mathtt{Col}_y \leftarrow \mathtt{Col}_y | P_{new}$
end if
end for

The whole process is executed in three steps:

1. compute the coloring of the state space from the initial states using Algorithm 1,
2. for each of the final states, take the resulting coloring and run Algorithm 1 again from that final state and store the resulting coloring of that state,
3. for $f_1, f_2, ..., f_n$ final states, compute a word $w = Col_{f_1}|Col_{f_2}|...|Col_{f_n}$ from the stored colorings.

Proposition 7.1. *If w is the result of the procedure above, then $\forall n, 1 \leq n \leq |\mathcal{P}| : w(n) = 1 \Leftrightarrow \exists \pi^0_{\mathcal{P}(n)}$ in $BA(\mathcal{R}, \varphi)_{\mathcal{P}(n)}$.*

We now need to prove that Algorithm 1 correctly computes coloring of the state space, formally that the condition $\forall x \in X, \forall n \in \{1, ..., |\mathcal{P}|\} : Col_x(n) = 1 \Leftrightarrow \exists x_0 \in X_0, x_0 \xrightarrow{P}{}^* x$ is satisfied.

Lemma 7.2. *After m-th execution (round) of the inner while loop in Algorithm 1, $Col_x(n) = 1 \Leftrightarrow \exists x_0 \in X_0, x_0 \xrightarrow{P}{}^{\leq m} x$.*

Proof. Before the first execution, only the initial states are colored.

In the m-th round, $Updates$ contains all states that have been colored in the previous round and each of these states is colored with parametrizations that allow to reach it in at most $m-1$ rounds. For each of these the $Pass$ function is called. $Pass$ takes a coloring of a state and for each of the neighbours:

1. removes those parametrizations that do not allow transition to that neighbour - $Col(x)\&Code(\{P|x \xrightarrow{P} y\})$,
2. tests if there is any parametrization that is not present in the coloring of that neighbour,
3. if the test succeeds, the succesor state is colored.

If the test fails it means that either the state has been already colored with these colors, therefore the coloring has been passed from the state in previous round and the reachability is preserved, or the transition is not allowed for any of parametrizations present in coloring of the source state.

□

Theorem 7.3. *Algorithm 1 computes the coloring $cl^{\mathcal{P}}_X$.*

Proof. According to Lemma 7.2, after at most $|BA(\mathcal{R}, \varphi)|$ rounds, every state is colored with all parametrizations that allow a path in at most $|BA(\mathcal{R}, \varphi)|$ transitions. Because the structure has size of $|BA(\mathcal{R}, \varphi)|$ there is no possibility for a path longer than that, therefore every state is colored by P *iff* P allows a path to this state.

As a corollary, after at most $|BA(\mathcal{R}, \varphi)|$ rounds, the test in function $Pass$ fails for all states in $Updates$ set, therefore $Updates$ and $Updates'$ remain empty and the algorithms finishes. □

Theorem 7.4. *Complexity of the algorithm is* $\mathcal{O}(|BA(\mathcal{R}, \varphi)|^2 * |\mathcal{R}| * |\mathcal{P}|)$*, where* $|\mathcal{R}|$ *is number of components of the regulatory network.*

Proof. According to Theorem 7.3 the outer while loop is executed at most $|BA(\mathcal{R}, \varphi)|$ times. $Updates$ contains at most $|BA(\mathcal{R}, \varphi)|$ states, therefore the inner while loop can be also executed always at most $|BA(\mathcal{R}, \varphi)|$ times. Each node can have at most $|\mathcal{R}| * 2$ neighbours, each of them differing by +/-1 in at most one component. Finally, the complexity of bitwise operations is linear in the size of the parametrization space.

Note that even though operations with coloring vectors are expensive their computation is in practice very fast due to the usage of bitwise operations. Also splitting of the parametrization space loweres the complexity in an average case, because the size of the reachable state space is reduced. □

Remark 7.5. *If the property belongs to a class of reachability properties, like a time series walk, it is not necessary to conduct a cycle detection [1].*

Algorithm 1 also allows for easy computation of the cost function. Before each round of the outer cycle we go through the states in the $Updates$ set, find final ones and check if they are colored with parametrizations whose cost is not yet known. If we find such, their cost is set equal to the number of this round.

7.2 Analysis

Knowing the cost value, we can compute all time series walks allowed by a parametrization. We have tried multiple approaches and the most efficient one seems to be a depth-first traversal with marking of states, which we briefly describe here. This method requires a structure that has transitions labeled with colors that have been actually passed during the computation. This information, however, can be easily stored within the coloring procedure.

The search starts from final states using a coloring of the individual states. From each of these states we then recursively descend to its predecessors, using only the coloring that has been passed through the transition originating in an individual predecessor. During the procedure the cost of each parametrization is compared to the current depth of the search. If the depth is greater, the parametrization is removed from the coloring, because the current path is then certainly not a part of the shortest time series walk. The procedure descends until the current coloring is not empty or until an initial state is found. In this case all transitions from the root of the search are stored for all parametrizations that remain in the coloring.

This algorithm can be improved by adding an instruction to store the current coloring and depth of the search during every visit of a state. When the state is visited again, this information is retrieved and parametrizations that have been already passed from the state at a depth lower or equal to the current depth are removed from the coloring because a shortest walk passing through this state, if it exists, has been already found. Note that this basically corresponds to the standard idea of marking the visited states in depth-first traversal.

Having a set of transitions for each parametrization, we can also easily compute the robustness value. There are many possibilities of how to compute such a value, for a reference see [1]. We have settled for a simple iterative algorithm that attaches a new floating point variable Prob : $0.0 \leq \text{Prob} \leq 1.0$ to each state, and works in rounds, adjusting and passing the current Prob value for each transition from its source to its sucessor.

To be correct, the algorithm has to distribute the initial value evenly between initial states that are sources of some transition. The sum of Prob values for these states must be equal to 1.0. The Prob value is then for each state, that is source of some transition divided, by a number of outcoming edges and passed to the successor. Values that have been passed are then summed at each state and used as a new Prob value in the following round. This procedure is repeated as many times as is the greatest cost value, which assures that all shortest paths have been traversed. The sum of Prob values from all final states then gives robustness value for each parametrization.

8 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 visualization of obtained results.

By evaluating our algorithms on several biological models, we have demonstrated that the computation scales well w.r.t. number of parallel processes, and moreover, that it copes with larger parameter spaces. Comparing these results with our previous results [2, 13], capabilities of parameter identification by model checking have been improved.

On the methodological side, our achievement brings new insights into applying discrete modeling frameworks to gene regulatory networks. The case study shows that the approach can help researchers identify reasonable parametrizations thus allowing for more elaborate approach to reverse engineering of regulatory networks.

In future work we want to focus on additional methods of raking as well as on extending the knowledge we can extract from the values we already have. This, we hope, could give us more certainty when reasoning about the properties of the network.

References

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[2] J. Barnat, L. Brim, A. Krejci, A. Streck, D. Safranek, M. Vejnar, and T. Vejpustek. On Parameter Synthesis by Parallel Model Checking. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(3):693–705, 2012.

[3] G. Batt, M. Page, I. Cantone, G. Goessler, P. Monteiro, and H. de Jong. Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. *Bioinformatics*, 26(18):i603–i610, 2010.

[4] G. Bernot, J.-P. Comet, A. Richard, and J. Guespin. 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] C. Chaouiya, E. Remy, B. Mossé, and D. Thieffry. Qualitative analysis of regulatory graphs: A computational tool based on a discrete formal framework. In *Positive Systems*, volume 294 of *Lecture Notes in Control and Information Sciences*, pages 830–832. Springer Berlin, 2003.

[6] M. Cline et al. Integration of biological networks and gene expression data using Cytoscape. *Nat. Protocols*, 2(10):2366–2382, 2007.

[7] F. Corblin, E. Fanchon, L. Trilling, C. Chaouiya, and D. Thieffry. Automatic inference of regulatory and dynamical properties from incomplete gene interaction and expression data. In *Information Processign in Cells and Tissues*, volume 7223 of *Lecture Notes in Computer Science*, pages 25–30. Springer, 2012.

[8] F. Corblin et al. A declarative constraint-based method for analyzing discrete genetic regulatory networks. *Biosystems*, 98(2):91 – 104, 2009.

[9] H. de Jong. Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. *Journal of Computational Biology*, 9(1):67–103, 2002.

[10] A. Fauré, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. In *ISMB (Supplement of Bioinformatics)'06*, pages 124–131, 2006.

[11] T. Helikar, J. Konvalina, J. Heidel, and J. A. Rogers. Emergent decision-making in biological signal transduction networks. *Proceedings of the National Academy of Sciences*, 105(6):1913–1918, 2008.

[12] E. M. C. Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[13] H. Klarner, H. Siebert, and A. Bockmayr. Time series dependent analysis of unparametrized thomas networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99(PrePrints), 2012.

[14] H. Klarner, A. Streck, D. Safranek, J. Kolcak, and H. Siebert. Parameter identification and model ranking of thomas networks. Technical Report FIMU-RS-2012-03, Masaryk University, 2012.

[15] R. Laubenbacher and P. Mendes. A discrete approach to top-down modeling of biochemical networks. In A. Kriete and R. Eils, editors, *Computational Systems Biology*, pages 229–247. Elsevier Academic Press, 2005.

[16] W.-P. Lee and W.-S. Tzou. Computational methods for discovering gene networks from expression da ta. *Briefings in Bioinformatics*, 10(4):408–423, 2009.

[17] C. Müssel, M. Hopfensitz, and H. A. Kestler. Boolnet – an r package for generation, reconstruction and analysis of boolean networks. *Bioinformatics*, 26(10):1378–1380, 2010.

[18] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theor. Comput. Sci.*, 412(21):2207–2218, 2011.

[19] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[20] A. Richard. SMBioNet-1.4 User manual, 2005.

[21] L. Sánchez and D. Thieffry. A logical analysis of the drosophila gap-gene system. *Journal of Theoretical Biology*, 211(2):115 – 141, 2001.

[22] I. Shmulevich and W. Zhang. Binary analysis and optimization-based normalization of gene expression data. *Bioinformatics*, 18(4):555–565, 2002.

[23] H. Siebert and A. Bockmayr. Incorporating time delays into the logical analysis of gene regulatory networks. In *Proceedings of the 2006 int. conf. on Computational Methods in Systems Biology*, CMSB'06, pages 169–183. Springer-Verlag, 2006.

[24] D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks II. Immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology*, 57:277–297, 1995.

[25] R. Thomas. Regulatory networks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology*, 153(1):1–23, 1991.

[26] M. Wahde and J. Hertz. Modeling genetic regulatory dynamics in neural development. *Journal of Computational Biology*, 8:442, 2001.

[27] X. Wen, S. Fuhrman, G. S. Michaels, D. B. Carr, S. Smith, J. L. Barker, and R. Somogyi. Large-scale temporal gene expression mapping of central nervous system development. *PNAS*, 95(1):334–339, 1998.