On Algorithmic Analysis of Transcriptional Regulation by Parallel Model Checking

J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková and D. Šafránek [*]

Faculty of Informatics, Masaryk University, Czech Republic

Abstract

Studies of cells in silico can greatly reduce the need for expensive and prolonged laboratory experimentation. The use of model checking for the analysis of biological networks has attracted much attention recently. The practical limitations are the size of the model and the time needed to generate the state space. In the paper we present a detailed time complexity study of two different algorithms for construction of discrete transition systems for piecewise-linear models of genetic regulatory networks (GRNs), and we introduce a tool GeNeSim for parallel model checking of GRNs that implements the choice of both algorithms. Finally, we present experiments which give a comparison of both algorithms in model checking selected properties on a cluster. The results bring a new algorithmic insight into the qualitative modeling approach well-studied in computational biology.

Key words: genetic regulatory network, piecewise-linear approximation, parallel model checking

1 Introduction

Understanding of the molecular mechanisms underlying the behavior of complex living systems is the main challenge of systems biology [21]. In particular, investigation of how large and complex biochemical regulatory networks control the response of a living cell to its ever-changing environment is the central topic receiving recently much attention. The stress response to environmental events is induced by the interaction of several interwoven modules with complex dynamic behavior, acting on different time scales. These networks

[*] Faculty of Informatics, Botanická 68a, 602 00 Brno, Czech Republic, tel: +420-549 494 476, fax: +420-549 491 820

Email address: `safranek@fi.muni.cz` (D. Šafránek).

[1] This work has been supported by the FP6 project No. NEST-043235 (EC-MOAN) and by the Grant Academy of Czech Republic grant No. 201/06/1338.

are large and complex. For instance, a human cell contains in the order of 10,000 substances which are involved in 15,000 different types of reactions. This gives rise to a giant cellular network with complex positive and negative feedback loops. As all the functional species in cells are made of proteins, the most basic layer of biological networks is the layer of genetic (transcriptional) regulation that drives gene expression (protein synthesis). This layer of biological network is called *genetic regulatory network* (GRN) [1,21]. In Fig. 1 there is depicted an example of a two-gene network where protein A represses production of its own and protein B represses production of both A and B.

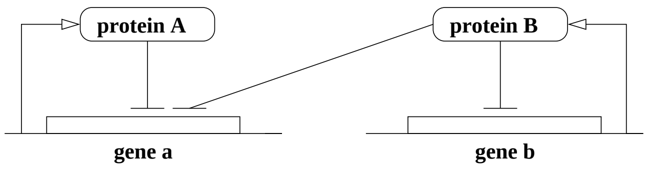

Fig. 1. Example of a simple GRN

In order to deal with the enormous intricacy of living organism functionality, experimental methods have to be supplemented with mathematical modeling and computer-supported analysis. One of the most critical limitations in applying current approaches to modeling and analysis is their scalability. Large models require powerful computational methods. The hardware infrastructure is nowadays already available in form of clusters, GRID, or multi-core computers, however, the parallel (distributed) algorithms for model analysis are still under development.

One of the most widely-used modeling frameworks for the analysis of biological networks, in particular the analysis of how concentration of species evolves in time, is based on the deterministic approach of *ordinary differential equations* (ODE) [23]. The reduction of continuous models to discrete automata by a sequence of reductions, approximations, and abstractions allows formal methods for the automated verification of properties of discrete transition systems to be applied [8]. However, when dealing with large models from systems biology, the standard model-checking techniques do not provide users with acceptable response times to answer their queries. This happens because the discrete transition systems contain thousands of millions of states even if the originating ODE model consists of a relatively small number of variables (less than 20). In other words, analysis of models that represent even small parts of complex networks is expensive in terms of computational resources.

Model checking is traditionally viewed as a verification procedure to verify that the given implementation exhibits properties given in the specification. On the other hand, model checking can also be considered as a flexible analysis tool—as long as the analyzed object is representable as a finite-state system and the analysis problem can be formulated in a suitable temporal logic. Consequently, model checkers are and always will be at the heart of many modeling and analysis tools. Recent studies on biologically relevant properties have identified the need for using both the branching-time temporal operators, that are able to express multi-stability properties (reachability of two or more different equilibrium states) and the linear-time operators, that are adequate to capture oscillations of protein concentration as well as temporal ordering of observable events. While substantial work on model-checking qualitative [6] as well as quantitative properties [20] of biochemical networks has been already achieved, no attempts to use parallel model checkers to analyze complex networks are known.

Motivation for employing the scalable model checking approach comes from our ongoing collaboration with biologists on modeling and analysis of carbon and nitrogen metabolism in Escherichia coli [16]. Basically, we are aware of at least two aspects motivating the application of the parallel model checking. First of all, tools for systems biology must be capable to satisfactorily handle specific networks and pathways containing at least tens of species. Especially, at the genetic regulatory level it means typically 10-20 crucial genes. In the language of ODE modeling it implies an analysis in 10-20-dimensional solution space. Moreover, with growing biological knowledge the number of species in analyzable networks will rapidly increase. Second aspect concerns the analysis of more complex properties than is possible by numeric and symbolic simulation methods. Additionally, there is also an important practical aspect of the analysis based on model-checking, namely the possibility to (semi)automatize the analysis.

1.1 Our Contribution

In this paper we deal with the discretization method as proposed in [15] and implemented in *Genetic Network Analyser (GNA)* [14]. The method is based on reduction of the original ODE system to a system of *piecewise-linear differential equations* (PLDE). Solutions of the reduced system are overapproximated by the qualitative finite state transition graph which can be analyzed by model checking. We bring new contribution to algorithmic analysis of genetic networks in the following directions. At first, we present a detailed time complexity study of two different algorithms for construction of discrete transition systems for PLDE models of GRNs, in particular, the original algorithm introduced in [15], and its simplified version. The simplified version has been already suggested in [15] at the intuitive level, however, its precise definition and the implementation are primarily realized in this work. At second, we introduce a tool GeNeSim for parallel linear temporal logic (LTL) model checking of GRNs, which implements both algorithms. Finally, we present experiments which give for both algorithms a comparison of times needed for model checking selected properties in different cluster settings.

GeNeSim is based on a distributed generator of qualitative transition systems. The generator is built as an inherent part of the DiVinE tool [3] allowing thus direct access to several efficient parallel on-the-fly LTL model-checking algorithms implemented in DiVinE. That way, GeNeSim can be used for qualitative analysis of biologically interesting properties on larger networks than it is possible with traditional sequential approach.

1.2 Related Work

As the biochemistry that controls cells of living organisms is a very complicated machinery, nontrivial physical properties have to be taken into account in models. There is a large scale of mathematical approaches for modeling of biochemistry [9]. They vary in amount of abstraction they provide. In this paper we deal with the traditional deterministic approach based on ODE. Most of tools provided for this approach are based on numeric simulation (quantitative analysis) of ODE solutions. Recently there appear specialized massively-parallel platforms [17,22,26] which can accelerate the numeric simulation. There is also a huge branch of quantitative simulation methods which incorporate stochastic modeling [19]. On the contrary, in this paper we deal with deterministic models, in particular, with a discrete approximation of ODE solutions. Such approximations are useful for analysis of GRNs with unknown exact values of reaction rates (qualitative analysis) [7,13]. Moreover, the abstraction provided by discrete models rapidly increases the possibility to predict not yet discovered biological mechanisms, and therefore it serves as a significant adviser for biological experiments [24].

The use of model checking for the analysis of biological networks has attracted much attention [6,8,25]. The individual approaches differ in models and model checking tools used. Our approach adds the parallel model checking analysis to the simulation features of the GNA tool [14]. For GNA, there was also developed a connection to CADP and NuSMV model checking tools [6]. However, these extensions currently employ only sequential model checking algorithms, and moreover, they rely on the need to explicitly generate the entire graph before passing it to the model checking tool. Besides GNA–CADP/NuSMV there are some other sequential approaches for model-checking GRNs. The BIOCHAM workbench [10] also provides an interface to NuSMV and CADP; the interface is based on a simple language for representing biochemical networks. The workbench provides mechanisms to reason about reachability, existence of partially described stable states, and some types of temporal behavior. Another tool is the Robust Verification of Gene Networks (RoVerGeNe) [4]. It complements the GNA by focusing on synthesis of quantitative parameters from PLDE models w.r.t. temporal properties of transcriptional dynamics. To the best of our knowledge, there is no implementation of the parallel on-the-fly model checking approach.

2 Preliminaries

In this section, we introduce formal notions used throughout this paper. At first, we present a simplified definition of PLDE systems, then we continue with definition of the qualitative state transition system (QSTS). As we focus on analysis of algorithms for QSTS construction, i.e, the computational complexity, the definition of QSTS presented here provides an algorithmic view of respective mathematical definitions given in [15].

Assume a network with n proteins. Let the concentration of ith protein in a time instant t, where $i \in \{1, ..., n\}$, be denoted by the variable $x_i(t)$. The respective PLDE system consists of a set of n equations. For each protein there is an equation describing how its concentration changes in time:

$$\frac{dx_i}{dt} = \sum_{l \in L} \kappa_{il} \varrho_{il}(\langle x_1, ..., x_n \rangle) - \gamma_i x_i$$

where

- L is a finite index set.
- For each $i \in \{1, .., n\}, l \in L$, κ_{il} is a constant expressing the rate of protein production.
- $\varrho_{il} : \mathcal{R}^n \to \{0, 1\}$ is a discrete input function that denotes the transcriptional regulation.
- γ_i is a constant expressing the rate of exponential decay of protein i.

In general, each equation of the system consists of two terms – the positive *production term* and the negative *degradation term*. The latter term is naturally nonzero and describes instability of proteins among other species in the cell. The former term describes the gene regulation, in particular, the intensity of protein production w.r.t. the current conditions in the cell. More precisely, the maximal production rate is given by the production constant κ which is additionally regulated by the input function ϱ.

The input function ϱ in general depends on the current concentration of all proteins in the system. In piecewise-linear approximation the (multidimensional) input function has a discrete range and is given by a product of elementary (one-dimensional) *step functions*. These step functions qualitatively characterize edges in GRNs, in particular, the activation or repression of the target gene by a certain transcriptional factor w.r.t. its given threshold concentration. In general, the input function has the form:

$$\varrho_{il}(\langle x_1, ..., x_n \rangle) = \prod_{j \in J^+} s^*(x_j, \theta^i_j) \cdot \prod_{k \in K} (1 - \prod_{j \in J^-_k} s^*(x_j, \theta^i_j))$$

where K is a finite index set, $J^+, J^-_k \subseteq \{1, ..., n\}$ such that $J^+ \cap \bigcup_{k \in K} J^-_k = \emptyset$, and $s^* : \mathcal{R} \to \{0, 1\}$, $* \in \{+, -\}$, denotes the step function defined for the protein concentration x_j and its threshold θ^i_j by the following expressions:

$$s^+(x_j, \theta^i_j) \overset{df}{=} \begin{cases} 1, \text{ if } x_j > \theta^i_j, \\ 0, \text{ if } x_j < \theta^i_j, \end{cases} \qquad s^-(x_j, \theta^i_j) \overset{df}{=} 1 - s^+(x_j, \theta^i_j)$$

s^+ characterizes activation, whereas s^- stands for repression.

For each variable there is a finite number of regulation configurations determined by particular evaluation of step functions. In each such configuration the transcriptional regulation tends towards an equilibrium state (the so-called *local equilibrium*) defined for some $L' \subseteq L$ by the equation:

$$\frac{dx_i}{dt} = 0 \Leftrightarrow x_i = \frac{\sum_{l \in L'} \kappa_{il}}{\gamma_i}$$

A crucial property of the PLDE model is the possibility of symbolic computation of qualitative solutions for given initial conditions. In particular, even when no quantitative data regarding the rate constants are available, evolution of the system can still be predicted by overapproximating the set of possible solution trajectories. The only additional information which must be specified (for each state variable) is the requested ordering of thresholds and the position of local equilibrium concentration values (given relatively to the thresholds). Therefore, by a *PLDE model* we always mean a PLDE system together with a set of threshold inequalities and a set of equilibrium inequalities. An example of a PLDE model of the two-gene network mentioned in Section 1 is given in Fig. 2.

Piecewise-Linear Diff. Equations	Threshold Inequalities	Equilibrium Ineqs
$\frac{dx_a}{dt} = \kappa_a\, s^-(x_a, \theta_a^1)\, s^-(x_b, \theta_b^1) - \gamma_a\, x_a$	$min_a < \theta_a^1 < max_a$	$\theta_a^1 < \frac{\kappa_a}{\gamma_a} < max_a$
$\frac{dx_b}{dt} = \kappa_b\, s^-(x_b, \theta_b^2) - \gamma_b\, x_b$	$min_b < \theta_b^1 < \theta_b^2 < max_b$	$\theta_b^2 < \frac{\kappa_b}{\gamma_b} < max_b$

Fig. 2. A two-gene PLDE model of the network from Fig. 1

2.2 Qualitative State Transition System

For a PLDE model and the given initial conditions, the set of all possible qualitative solutions is represented by a *qualitative state transition system* (QSTS). The algorithm for the construction of the QSTS relies on the fact that input functions given as products of step functions have discrete ranges. Up-to this approximation, concentration values of proteins are discretely abstracted into several open intervals between respective thresholds, and additionally, the discrete points equal just to the (symbolic) threshold values.

Moreover, from the inherent property of transcriptional regulation to keep concentration of each participating protein below some maximal level (each protein is naturally degraded), each solution of the PLDE system must stabilize below these maximal levels [15]. Therefore it is correct to deal only with qualitative concentration levels (symbolically) bounded by the maximal levels. Formally, the n-dimensional state space is bounded in the i-th dimension by values min_i and max_i for each $i \in \{1, \ldots, n\}$.

With respect to the discrete approximation, the solution space of the PLDE system can be divided into finite number of open rectangular partitions – so-called *regulatory domains*. In each regulatory domain all solutions of the PLDE system share the same qualitative behavior, i.e., they have the same qualitative phase. Boundaries among regulatory domains are called *switching domains*. In every switching domain at least one of the concentration variables has a constant value given by some relevant threshold. Each domain is uniquely represented by a corresponding qualitative state in QSTS.

In our approach, every domain is specified by its scope in each dimension. The scope can be an open interval between thresholds or a single point (a threshold which is in fact a closed interval). A regulatory domain can be written as a product of n open intervals. A switching domain is a product of m, $(0 < m \leq n)$ closed intervals and $n - m$ open intervals. We use notation $\dim(D)$ to denote the dimension of domain D.

Dimensions in which the projection of domain D is a single point are called switching (also *switching variables* in D), dimensions in which the projection is an open interval are called non-switching (also *non-switching variables* in D). The dimension of a switching domain is determined just by the number of non-switching variables. In particular, for a domain D with just m closed intervals, we have $\dim(D) = n - m$.

Let us assign each threshold and each interval between two adjacent thresholds a natural number according to Fig. 3 (a), in which domains of the two-gene example mentioned above are depicted (filled circles represent switching domains while non-filled circles regulatory domains). In each dimension every single point is followed by an open interval and vice-versa, so threshold points are assigned even numbers and intervals odd numbers.

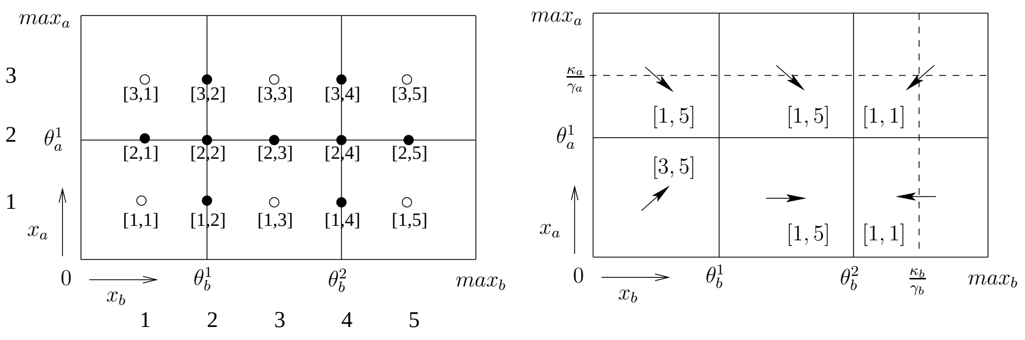

Fig. 3. (a) Domains/qualitative states (b) Focal directions

Definition 1 *A domain D is determined by an n-tuple of natural numbers (domain* coordinates*) $a_i \in \{1, \ldots, (2k+1)\}$, and denoted $D \equiv [a_1, \ldots, a_n]$. D_i denotes projection of the domain D in the i-th dimension, $D_i \overset{df}{=} a_i$.*

Domain D is regulatory *iff all its coordinates are odd. Domain D is* switching *iff at least one of the coordinates is a switching variable (an even number).*

Further we define neighbours $\mathrm{nbs}(D)$ of a domain $D \equiv [a_1, \dots, a_n]$ *as the set of all the domains* $D' \equiv [b_1, \dots, b_n]$ *satisfying exactly one of the following conditions:*

(1) There is at least one variable x_i switching in D that is not switching in D' with $b_i = a_i \pm 1$ for all such x_i. All the other coordinates of D' are the same as the coordinates of D.
(2) There is at least one variable x_i non-switching in D that becomes switching in D' with $b_i = a_i \pm 1$ for all such x_i. All the other coordinates of D' are the same as the coordinates of domain D.

Each domain identifies a unique state of the QSTS. We can now formally introduce the qualitative state transition system.

Definition 2 *For a PLDE model $\mathcal{M}$ define its* qualitative state transition system $\mathrm{QSTS}(\mathcal{M})$ *as a tuple $\langle S, T, S_0 \rangle$ where S is a finite set of qualitative states each of which is denoted QS^i and uniquely associated with a particular domain D^i, $T \subseteq S \times S$ a finite transition relation, and $S_0 \subseteq S$ a set of initial states representing the* initial condition.

Following definitions lead to computational specification of the transition relation of QSTS. Especially, at first we recall the original algorithm and then we introduce its simplified underapproximative modification (obtained by omitting some of the transitions).

Since in a particular domain each solution can either remain or it can proceed to some neighboring domain, each transition from state denoting domain $D \equiv [a_1, \dots, a_n]$ to state denoting (neighboring) domain $D' \equiv [b_1, \dots, b_n]$ is determined by a direction vector $w \equiv \langle w_1, \dots, w_n \rangle \in \{-1, 0, 1\}^n$ such that $w_i = b_i - a_i$.

The transition relation is mathematically defined in [15] (page 17) by two transition rules which decide (coordinatewisely) if a transition between any two neighboring domains is allowed or forbidden. Here we reformulate those rules in terms of our simplified language. Inside the rules there appears a set of qualitative directions – the so-called *direction set*, denoted $V(D)$, which is a central notion for determining the transitions evolving from the corresponding qualitative state. We will now gradually define this notion.

For every regulatory domain D we define its *focal point* $\mathrm{Fp}(D)$ as the position of the local equilibrium for domain D, $\mathrm{Fp}(D) \equiv [p_1, \dots, p_n] \in \{1, \dots, 2k+1\}^n$. The focal point $\mathrm{Fp}(D)$ specifies the direction in which every substrate (variable) concentration will evolve over time inside D. For every possible combination of production and degradation parameters of substrate i, the position of its local equilibrium is given as part of the PLDE model, in particular, its equilibrium inequalities. Thus the coordinate p_i of focal point $\mathrm{Fp}(D)$ is given by the specific configuration of production and degradation parameters of substrate i which is characteristic for D. We use the notation $\mathrm{Fp}(D)_i$ to denote i-th coordinate of the focal point.

The relative position of a focal point $\mathrm{Fp}(D)$ w.r.t. the domain D is specified by *focal direction of* D, denoted $F(D)$, and defined $F(D) \equiv \langle f_1, \ldots, f_n \rangle \in \{-1, 0, 1\}^n$, where for each $i \in \{1, ..., n\}$, $f_i \overset{df}{=} sgn(\mathrm{Fp}(D)_i - D_i)$. We use notation $F(D)_i$ to denote projection of the focal direction of D to the i-th dimension. Example of focal directions is showed by arrows in Fig. 3 (b). In the figure, there is also the respective focal point attached to each arrow.

In every switching domain D, values of all those step functions that depend on switching variables are undefined. Hence the values of switching variable derivatives are undefined, and so is direction of system solution. Therefore, there is a mechanism which overapproximates possible solution direction by analyzing the behavior in neighboring domains. In particular, this analysis is realized by a so-called *direction hyperrectangle* $\mathrm{dh}(D)$ determined by focal directions of all neighboring domains in $\mathrm{nbs}(D)$. Formally, direction hyperrectangle $\mathrm{dh}(D)$ of domain D is given by a pair of two n-tuples $\mathrm{dh}(D) \overset{df}{=} [\mathrm{Min}(D), \mathrm{Max}(D)]$, $\mathrm{Min}(D), \mathrm{Max}(D) \in \{-1, 0, 1\}^n$, that contain coordinatewisely its lower and upper boundary, respectively. In particular, the n-tuples are defined by the following equations:

$$\mathrm{Min}(D) \overset{df}{=} \langle w_1^\perp, \ldots, w_n^\perp \rangle, \text{ where } \forall i \in \{1, ..., n\}.\, w_i^\perp \overset{df}{=} min\, \{F(D')_i | D' \in \mathrm{nbs}(D)\}$$
$$\mathrm{Max}(D) \overset{df}{=} \langle w_1^\top, \ldots, w_n^\top \rangle, \text{ where } \forall i \in \{1, ..., n\}.\, w_i^\top \overset{df}{=} max\, \{F(D')_i | D' \in \mathrm{nbs}(D)\}$$

Now we can introduce the *direction set* $V(D)$ of domain D. We distinguish two cases – when D is a switching domain and when it is not. In [15], the direction set of a switching domain with m switching variables is geometrically constructed by intersection of the direction hyperrectangle with the $(n - m)$-dimensional hyperplane C containing D[1]. In our simplified language, $V(D)$ is represented by a pair of two n-tuples $V(D) \equiv [V_\perp(D), V_\top(D)]$, $V_\perp(D), V_\top(D) \in \{-1, 0, 1\}^n$, which contain coordinatewisely its lower and upper boundary, respectively. Coordinates of $V_\perp(D)$ and $V_\top(D)$ are denoted for all $i \in \{1, ..., n\}$ as $v_i^\perp$ and $v_i^\top$, respectively. The intersection of the direction hyperrectangle $\mathrm{dh}(D)$ with the hyperplane C is computed coordinatewisely by the following rules:

- If i is a non-switching variable in D then for the i-th coordinate of every point of the intersection the inequality $w_i^\perp \leq x_i \leq w_i^\top$ must be satisfied, therefore $v_i^\perp \overset{df}{=} w_i^\perp, v_i^\top = w_i^\top$.
- If i is a switching variable in D then for the i-th coordinate of every point of the intersection the inequality $w_i^\perp < x_i < w_i^\top$ must be satisfied, therefore if $w_i^\perp = -1 \wedge w_i^\top = 1$ then $v_i^\perp = v_i^\top = 0$, else the whole intersection is empty (this fact is indicated by the notation $\mathrm{dsp}(D) = 1$ defined further on).

If D is a regulatory domain D then the direction set $V(D)$ is defined as the set of all unitary vectors included in the smallest hyperrectangular region bounded

[1] The hyperrectangular set of all focal directions intersected by the $(n - m)$-dimensional hyperplane containing D is labeled as $V(D, \Psi(D))$ in [15].

by $F(D) \equiv \langle f_1, ..., f_n \rangle$ and the zero vector $0^n \equiv \langle 0, \dots, 0 \rangle$. In particular, $V(D)$ is computed coordinatewisely (for each $i \in \{1, ..., n\}$) by the following rules:

- If $f_i = -1$ then $v_i^\perp \overset{df}{=} -1$ else $v_i^\perp \overset{df}{=} 0$.
- If $f_i = 1$ then $v_i^\top \overset{df}{=} 1$ else $v_i^\top \overset{df}{=} 0$.

Henceforth, $v \in V(D)$ will denote the case where a vector $v \in \{-1, 0, 1\}^n$ satisfies $\forall i \in \{1, \dots, n\} \,.\, v_i^\perp \leq v_i \leq v_i^\top$ for a given $V(D)$. Note that according to the rules also $0^n \in V(D)$, which reflects the fact that the qualitative state corresponding to the regulatory domain D is a *steady state* of the PLDE system (there exists a solution which will never leave D).

For the sake of simplicity, we introduce a multi-valued predicate $\mathrm{dsp}(D)$ (*direction set property*) that characterizes the direction set for a domain D by the cumulative information about possible directions included in $V(D)$. For a given domain D, the value of $\mathrm{dsp}(D)$ is defined as follows:

- $\mathrm{dsp}(D) = 0 \Leftrightarrow V(D)$ has not been computed yet
- $\mathrm{dsp}(D) = 1 \Leftrightarrow V(D) = \emptyset$
- $\mathrm{dsp}(D) = 2 \Leftrightarrow V(D) \neq \emptyset$, but $\langle 0, \dots, 0 \rangle \notin V(D)$
- $\mathrm{dsp}(D) = 3 \Leftrightarrow V(D) \neq \emptyset$ and $\langle 0, \dots, 0 \rangle \in V(D)$

The case when $\mathrm{dsp}(D) = 1$ can arise only when D is a switching domain and it means that no solution can remain in D for arbitrarily small time instant. The cases $\mathrm{dsp}(D) = 2$ and $\mathrm{dsp}(D) = 3$ are meaningful for both kinds of domains. The former case indicates that the state related with D is not steady, but solutions can remain in D for a finite time interval. The latter case denotes that the state related with D is a steady state.

Finally, we can reformulate the transition rules determining whether there exists a transition from D to some $D' \in \mathrm{nbs}(D)$.

Transition rule 1 Let $D \equiv [a_1, \dots, a_n]$ be a domain, $D' \equiv [b_1, \dots, b_n] \in \mathrm{nbs}(D)$ be its lower dimension neighbor domain (i.e., $\dim(D) > \dim(D')$ and some variables in D' are switching), and $w = D' - D$ be the direction vector from D to D'. Then, there exists a transition from D to D' iff the two following conditions are satisfied:

(1) $V(D) \neq \emptyset$ (equivalently written as $\mathrm{dsp}(D) > 1$)
(2) $\exists v \in V(D)$ such that $\forall i \in \{1, ..., n\}.\ a_i \neq b_i \Rightarrow v_i w_i = 1$

More intuitively, a transition from D to a lower dimension domain D' is possible only if the direction set of the initial domain D contains a direction heading towards D'.

Transition rule 2 Let $D \equiv [a_1, \dots, a_n]$ be a domain, $D' \equiv [b_1, \dots, b_n] \in \mathrm{nbs}(D)$ be its higher dimension neighbor domain (i.e., $\dim(D) < \dim(D')$ and some variables in D' are non-switching), and $w = D' - D$ be the direction vector from D to D'. Then, there exists a transition from D to D' iff the two following conditions are satisfied:

(1) $V(D') \neq \emptyset$ (equivalently written as $\mathrm{dsp}(D') > 1$)
(2) $\exists v \in V(D')$ such that $\forall i \in \{1, ..., n\}.\ a_i \neq b_i \Rightarrow v_i w_i \neq -1$

More intuitively, a transition from D to a higher dimension domain D' is possible only if the direction set of the destination domain D' does not contain a direction heading against the considered transition, i.e., towards D.

2.2.1 Construction of Original QSTS

Now we can present our refined version of the original algorithm scheme developed by de Jong et. al. [15] for construction of QSTS($\mathcal{M}$)$\equiv \langle S, T, S_0 \rangle$ for an arbitrary PLDE model $\mathcal{M}$:

Algorithm 1 *QSTS construction – a refined version of the original algorithm*

```
01 for every regulatory $D$ compute $F(D)$, $V_\perp(D)$, $V_\top(D)$, $\mathrm{dsp}(D)$
02 for every switching $D$ compute $V_\perp(D)$, $V_\top(D)$, $\mathrm{dsp}(D)$
03 for each initial qualitative state $QS^i \in S_0$ do
04  push(stack, $QS^i$)
05  while not stack empty do
06      current qualitative state $QS^i \leftarrow$ pop(stack)
07      determine candidate successors (using $\mathrm{nbs}(D^i)$) of lower dimension
08      determine candidate successors (using $\mathrm{nbs}(D^i)$) of higher dimension
09      for every candidate successor $QS^j$ do
10          if $D^i$ and $D^j$ satisfy rule 1 or rule 2 then
11              update the transition relation by $T := T \cup \{\langle QS^i, QS^j \rangle\}$;
12              if not $QS^j$ reached before then
13                  mark $QS^j$ as reached;
14                  push(stack,$QS^j$);
15              fi
16          fi
17      end
18  end
19 end
```

The main point of interest in Algorithm 1 comes on lines 9–17, when we are forced to determine all actual successor states QS^j of the current qualitative state QS^i. To do this we must be able to decide which transitions to neighboring domains (qualitative states) are possible according to the two transition rules mentioned above. To decide the two rules for every transition from domain D to its neighbor D', $V_\perp(D), V_\top(D)$, and $\mathrm{dsp}(D)$ must be computed if D' is of lower dimension, or $V_\perp(D'), V_\top(D')$, and $\mathrm{dsp}(D')$ must be computed if D' is of higher dimension than D. For our complexity analysis presented further on, we let this computation to be performed at the beginning, preliminary to the construction of the transition relation. It is also worth noting that in current version of GeNeSim this computation is realized on-the-fly with respect to the implicit (and distributed) approach for representation of QSTS.

However, as it will be deduced in the following section, this difference has no significant impact on the complexity results. The QSTS constructed for the example PLDE model is depicted in Fig. 4a. The steady state is circled.

2.2.2 Construction of Underapproximated QSTS

Even from the intuitive point of view, it is easy to see that the number of all candidate successors of an arbitrary state (lines 7–8 in Algorithm 1) is in worst cases exponential w.r.t. the model dimension. More precisely, for a state associated to any one-dimensional switching domain there are exactly $3^n - 1$ candidate successors. In order to avoid this exponential blow up, we employ the idea suggested by de Jong et.al. (conclusion section in [15]) to underapproximate the resulting QSTS by removing all transitions among domains which differ in dimension by more than one. By such a reduction, the number of candidate successors that we get in the worst case appears to be significantly lower (at least by half). Number of states is not reduced. Detailed analysis is given in the next section.

Qualitative solutions which are omitted by this reduction correspond to situations which are, from the physical point of view, significantly less probable than the retained solutions. In particular, the omitted transitions correspond exactly to situations when at least two or more species change their concentration in just the same time instant. Of course, this abstraction can be in specific cases inappropriate, however, it can still be satisfactorily used for general analysis of most models. In particular, when model checking of a particular formula on an underapproximated QSTS returns a negative answer, we are sure that the formula is not satisfied in the original QSTS. Nonetheless, we cannot make such a conclusion for positive answers.

The only change needed to our algorithm in order to implement the reduction introduced above involves a slight refinement of the lines 7–8 as follows. The respective QSTS for the example PLDE model is depicted in Fig. 4b.

Algorithm 2 *QSTS construction – reduced version of the original algorithm*

```
07  determine candidate successors (using nbs($D^i$)) of dim. lower by one than $D^i$
08  determine candidate successors (using nbs($D^i$)) of dim. higher by one than $D^i$
```

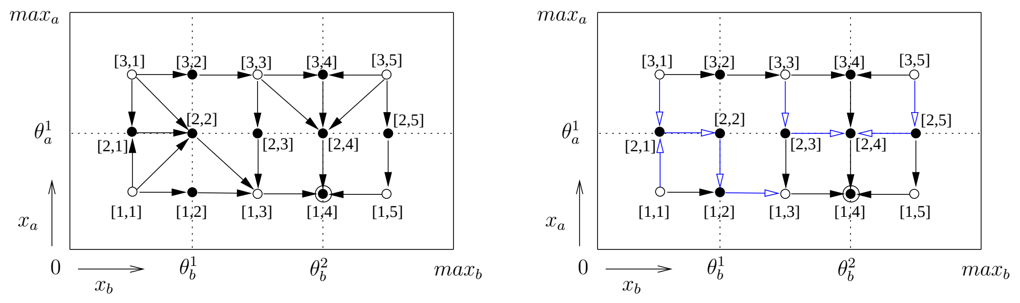

Fig. 4. (a) QSTS by Algorithm 1 (b) QSTS by Algorithm 2

3 Complexity Analysis

In this section we aim to analyze the time complexity of the original algorithm and compare it to the complexity of the underapproximative algorithm introduced in the previous section.

Throughout this section, for the sake of simplicity we suppose there are k thresholds $\theta_i^1, \ldots, \theta_i^k$ between min_i and max_i in each of the n dimensions.

3.1 Complexity of Algorithm 1

In all following claims we target the worst case complexity. The worst case happens when reaching all the qualitative states in the model.

Numbers of domains (representing qualitative states) are the following:

- the total number of all domains is $(2k+1)^n$,
- there are $(k+1)^n$ regulatory domains (all coordinates must be odd),
- thus there are $(2k+1)^n - (k+1)^n$ switching domains in total,
- and $\binom{n}{m} k^m (k+1)^{n-m}$ switching domains with exactly m switching variables.

The number of all domains grows exponentially with the number of variables (substrates) of the model. This is an inherent property of the model. Together with the fact that the number of candidate successors of a qualitative state is in general also exponential it is the basis for a complexity of $O(n(4k+3)^n)$ which is explained in this section and proved in Section 3.

In the worst case we have to compute $F(D)$, $V(D)$ (i.e. $V_\perp(D)$, $V_\top(D)$) and $\mathrm{dsp}(D)$ for all regulatory domains and $V(D)$, $\mathrm{dsp}(D)$ for all switching domains assuming that all domains are reachable from the initial condition. Thus we can compute these values at the beginning (lines 01–02 of Algorithm 1). We also have to identify all candidate successors D' of every regulatory domain D (D' is always of lower dimension than D — line 07 of Algorithm 1) and all candidate successors of every switching domain (both higher and lower dimension neighbors are included in this case — lines 07–08 of Algorithm 1). These are the basic parts into which the complexity analysis is decomposed.

3.1.1 Computing $V(D)$

Computing $\mathrm{Fp}(D)$, $F(D)$, and $V(D)$ for a regulatory domain D can be performed in time $O(ns)$ where s is the upper bound of the number of step-functions appearing on the right sides of the PLDE system. Therefore computing these sets for all regulatory domains can be performed in time

$$O(ns(k+1)^n) \tag{1}$$

Computing $V(D)$ for a switching domain D with m switching variables consists of constructing the direction hyperrectangle $\mathrm{dh}(D)$ (i.e. $[\mathrm{Min}(D), \mathrm{Max}(D)]$) and intersecting it with the $(n-m)$-dimensional hyperplane C containing D.

By definition $\mathrm{dh}(D)$ is constructed from focal directions $F(D')$ of all neighboring regulatory domains $D' \in \mathrm{nbs}(D)$. Since the number of these is in general 2^m the complexity of constructing $\mathrm{nbs}(D)$ is $O(n2^m)$. However this can be improved to linear time as we demonstrate on the following example of constructing $\mathrm{dh}(D_{2,2} \equiv [2,2])$ from Fig. 3a.

Example 3 (construction of $\mathrm{dh}(D_{2,2}) \equiv [\mathrm{Min}(D_{2,2}), \mathrm{Max}(D_{2,2})]$**)**
$\mathrm{Min}(D_{2,2}) \overset{df}{=} \langle w_1^\perp, w_2^\perp \rangle$ *is given coordinatewisely by the following expressions:*
$\forall i \in \{1,2\}. w_i^\perp = min\{F(D_{1,1})_i, F(D_{3,1})_i, F(D_{1,3})_i, F(D_{3,3})_i\} = min\{min\{F(D_{1,1})_i, F(D_{3,1})_i\}, min\{F(D_{1,3})_i, F(D_{3,3})_i\}\} = min\{\mathrm{Min}(D_{2,1})_i, \mathrm{Min}(D_{2,3})_i\}$.
The analogous result can be inferred for $\mathrm{Max}(D_{2,2})$ *and hence* $\mathrm{dh}(D_{2,2})$ *can be computed by analyzing only two domains from* $\mathrm{nbs}(D_{2,2})$ *instead of all.*

In general, $\mathrm{dh}(D)$ of any switching domain D with $m > 1$ switching variables can be computed in linear time from any two direction hyperrectangles $\mathrm{dh}(D^-)$, $\mathrm{dh}(D^+)$ of neighboring domains $D^+, D^- \in \mathrm{nbs}(D)$ with $m-1$ switching variables satisfying $D - D^+ = -(D - D^-)$. For switching domains with one switching variable, $\mathrm{dh}(D)$ must be computed directly according to the definition. However, since in this case the considered domains have exactly two neighboring regulatory domains, this step is linear too.

After gradualy constructing $\mathrm{dh}(D)$ for all switching domains D (from highest dimension to lowest) the intersection of the direction hyperrectangles with the respective smallest hyperplanes containing the considered domains is computed coordinatewisely in linear time. This results in complexity $O(n)$ for computing $V(D)$ of any switching domain D, and hence for all switching domains we get:

$$O(n((2k+1)^n - (k+1)^n)) \tag{2}$$

3.1.2 Computing successors

All candidate successors of a regulatory domain D are placed relatively to D in the directions specified by $V(D)$, there are generally 2^n directions in $V(D)$ for a regulatory domain D. Candidate successors of a switching domain with m switching variables are of two types. The lower dimension successors must lie in the direction $v \in V(D)$, where the size of $V(D)$ is in general 3^{n-m}. The higher dimension successors must be all tested w.r.t. the transition rules, the number of such successors is 3^m.

Checking Transition rule 1 and 2 (TR1, TR2) for any candidate successor D' of D can be done in time $O(n)$ by comparing the coordinates of $V(D)$ resp. $V(D')$ in which D and D' differ. This gives us a complete estimation of how long it will take to compute actual successors for any single domain and for all the domains of a system (for proofs see Section 3.3):

$O(n2^n)$	computing the transitions for a regulatory domain	(3)
$O(n2^n(k+1)^n)$	for all regulatory domains in the system	(4)

$$O(n(3^{n-m} + 3^m)) \quad \text{for a switching domain with } m \text{ switching variables} \tag{5}$$
$$O(n(4k+3)^n) \quad \text{for all the switching domains in the system} \tag{6}$$

Computing the actual successors for all the regulatory and switching domains in the system requires time

$$O(n(4k+3)^n) \tag{7}$$

Total complexity of Algorithm 1 is the sum of (1), (2) and (7) which is

$$O(n(4k+3)^n) \tag{8}$$

3.2 Complexity of Algorithm 2

Since the main idea of Algorithm 2 is to allow only transitions between domains (resp. qualitative states) which differ in their dimension by exactly one, the speed-up is determined by the number of transitions that must be checked. Computing $F(D)$, $V(D)$, and $\operatorname{dsp}(D)$ requires the same time as in Algorithm 1, since we have not reduced the state space, but only the transitions.

Computing actual successors (existing transitions) requires time

$$O(n^2) \quad \text{for a regulatory domain } D \tag{9}$$
$$O(n^2(k+1)^n) \quad \text{for all regulatory domains in the system} \tag{10}$$
$$O(n^2) \quad \text{for an arbitrary switching domain } D \tag{11}$$
$$O(n^2((2k+1)^n - (k+1)^n)) \quad \text{for all switching domains in the system} \tag{12}$$

Computing the actual successors for all the domains in the system takes time

$$O(n^2(2k+1)^n) \tag{13}$$

Complexity of Algorithm 2 is obtained by summing up (1), (2) and (9) which results in

$$O(n^2(2k+1)^n) \tag{14}$$

Summary 1 (Comparison of the results) *By omitting transitions between domains that differ in dimension by more than 1, we can reduce the complexity asymptotically from $O(n(4k+3)^n)$ to $O(n^2(2k+1)^n)$.*

In following sections proofs of above mayor complexity estimations are given. Each proof refers to a corresponding complexity expression stated above.

3.3 Proofs of complexity results for Algorithm 1

3.3.1 Computing $V(D)$ — proofs of (1) and (2)

Transitions from one state to another are determined by TR1 and TR2 from section 2.2. Since both of these rules depend on the knowledge of $V(D)$ and $\operatorname{dsp}(D)$, they must be computed for every regulatory and switching domain.

Proof of (1) **(computing $V(D)$ of regulatory domain D)** The focal point $\mathrm{Fp}(D)$ of a regulatory domain D is computed by coordinates. For each coordinate i the differential equation for variable i is computed giving the actual production and degradation parameters present on the right side of the equation in the conditions of domain D. Since the number of step functions is bounded by s it takes $O(s)$ time to decide which parameters are present. The position of the local equilibrium for this specific combination is then retrieved from model input (in time $O(1)$ if proper indexing is used). Therefore it takes time $O(ns)$ to compute focal point $\mathrm{Fp}(D)$ from all the n variables of domain D.

The transformation of a focal point $\mathrm{Fp}(D)$ to a focal direction $F(D)$ takes linear time $O(n)$. The computation of $V(D)$ (i.e. $V_{\perp}(D)$, $V_{\top}(D)$) is done coordinate-wise in linear time $O(n)$ according to the two definition rules. For a regulatory domain D only two values of $\mathrm{dsp}(D)$ are possible. $\mathrm{dsp}(D) = 3 \Leftrightarrow \forall i \in \{1, \ldots, n\}.(v_i^{\perp} = 0 \vee\ v_i^{\top} = 0)$ else $\mathrm{dsp}(D) = 2$. This can be checked in linear time $O(n)$.

Therefore computing $F(D)$, $V(D)$, $\mathrm{dsp}(D)$ for all $(k+1)^n$ regulatory domains can be done in time $O(ns(k+1)^n)$. $\square$

Proof of (2) **(computing $V(D)$ of switching domain D)** As is shown in section 3.1.1 the computation of $V(D)$ for a switching domain D consist of constructing $\mathrm{dh}(D)$ and intersecting it with a hyperplane C.

For every switching domain D of dimension $n-1$, $\mathrm{dh}(D)$ (i.e. $[\mathrm{Min}(D), \mathrm{Max}(D)]$) is computed as $\mathrm{Min}(D) \overset{df}{=} \langle w_1^{\perp}, \ldots, w_n^{\perp}\rangle$, $\forall i \in \{1, \ldots, n\}.w_i^{\perp} \overset{df}{=} min\{F(D^+)_i,\ F(D^-)_i\}$, where $D^+, D^- \in \mathrm{nbs}(D)$ are regulatory domains of dimension n. Similarly for $\mathrm{Max}(D)$.

For every switching domain D of dimension $m < n-1$, $\mathrm{dh}(D)$ (i.e. $[\mathrm{Min}(D), \mathrm{Max}(D)]$) is computed as $\mathrm{Min}(D) \overset{df}{=} \langle w_1^{\perp}, \ldots, w_n^{\perp}\rangle$, $\forall i \in \{1, \ldots, n\}.w_i^{\perp} \overset{df}{=} min\{\mathrm{Min}(D^+)_i,\ \mathrm{Min}(D^-)_i\}$, where $D^+, D^- \in \mathrm{nbs}(D)$ are any switching domains of dimension $m+1$ such that $D - D^+ = -(D - D^-)$. Similarly for $\mathrm{Max}(D)$.

In this way the $\mathrm{dh}(D)$ of each switching domain D is progressively computed, proceeding from highest dimension domains to lowest in linear time $O(n)$.

According to the rules for intersecting $\mathrm{dh}(D)$ with the $(n-m)$-dimensional hyperplane C containing switching domain D with m switching variables, $V(D)$ can be computed coordinate-wise in linear time $O(n)$. During this process $\mathrm{dsp}(D)$ is acquired without adding to overall complexity.

Therefore computing $V(D)$, $\mathrm{dsp}(D)$ for a switching domain requires time $O(n)$, for all the $(2k+1)^n - (k+1)^n$ switching domains it takes time $O(n((2k+1)^n - (k+1)^n))$. $\square$

3.3.2 Computing successors — proofs of (3), (5) *and* (7)

When computing transitions from domain D (m switching variables) to its neighboring domains D' there are two cases:

If domain D' is of lower dimension (some of the $n-m$ non-switching variables of D become switching in D') then the TR1 says that the transition exists iff

(1) $V(D) \neq \emptyset \Leftrightarrow \mathrm{dsp}(D) > 1$ and
(2) $w = D' - D \in V(D)$

The second condition allows to consider only domains D' such that $D' - D \in V(D)$. This can be efficiently done coordinate-wise by computing all the vectors v that satisfy $\forall i \in \{1, \dots, n\}. v_i^\perp \leq v_i \leq v_i^\top$. In general this takes time $O(|V(D)|) = O(3^{n-m})$. Since putting each successor of D on the stack takes $O(n)$ time the whole step requires time $O(n3^{n-m})$.

If domain D' is of higher dimension (some of the m switching variables of D become non-switching in D') then TR2 says that the transition exists iff

(1) $V(D') \neq \emptyset \Leftrightarrow \mathrm{dsp}(D') > 1$ and
(2) $\exists v \in V(D')$ such that $v_i.w_i \neq -1$ for all i that become non-switching in D' ($w = D' - D$).

The difference from the previous case is that the outcome of TR2 depends entirely on the destination domain D'. Therefore we cannot efficiently reduce the number of candidate successors to check and must test all domains of higher dimension. These are obtained by altering some of the switching variables of D by ± 1 giving us generally $(3^m - 1)$ possible successors. For every such successor we have to test $\mathrm{dsp}(D') > 1$ and whether for each variable i that becomes non-switching in D' the condition: $v_i^\perp \leq w_i \leq v_i^\top \vee v_i^\perp \leq 0 \leq v_i^\top$ holds. Since the condition can be verified in linear time $O(m)$ it takes in general time $O(m3^m)$ to decide all higher dimension successors of D and time $O(n3^m)$ to put them all onto the stack.

Proof of (3) **(computing actual successors of regulatory domain** D**)**
All the neighboring domains of a regulatory domain D are of lower dimension, therefore TR1 will be used. As discussed above this step should take $O(n3^n)$ time ($m = 0$). However the structure of the $V_\perp(D)$, $V_\top(D)$ assures $|v_i^\perp - v_i^\top| \leq 1$ for every i (see def. of $V(D)$ for regulatory domains in section 2.2), which leads to a better complexity $O(n2^n)$. $\square$

Proof of (5) **(computing actual successors of switching domain** D**)**
Because in general a switching domain with m switching variables has both higher dimension and lower dimension neighbors we must incorporate both cases. By summing the number of transitions of both types we get complexity $O(n(3^m + 3^{n-m}))$. $\square$

Proof of (7) (computing actual successors for all domains)
Let us summarize already proved facts:

- $(k+1)^n$ regulatory domains in system
- $O(n2^n)$ time to stack successors of every regulatory domain
- $O(n2^n(k+1)^n)$ time to stack successors of all regulatory domains
- $O(n(3^m + 3^{n-m}))$ time to stack successors of a switching domain with m switching variables
- $\binom{n}{m}k^m(k+1)^{n-m}$ switching domains of dimension m
- $O(\binom{n}{m}k^m(k+1)^{n-m}n(3^m+3^{n-m}))$ time to stack successors of all the switching domains with m switching variables

To compute transitions from all switching domains we must sum the number of transitions from switching domains of all dimensions:

$$\begin{gathered}\sum_{m=1}^{n}\binom{n}{m}k^m(k+1)^{n-m}n(3^{n-m}+3^m) = \\ n\sum_{m=1}^{n}\binom{n}{m}\left(k^m(3k+3)^{n-m} + 3k^m(k+1)^{n-m}\right) = \\ \Big(n\sum_{m=0}^{n}\binom{n}{m}\left(k^m(3k+3)^{n-m} + 3k^m(k+1)^{n-m}\right)\Big) - n\Big((3k+3)^n + (k+1)^n\Big) = \\ n\Big((4k+3)^n + (4k+1)^n - (3k+3)^n - (k+1)^n\Big)\end{gathered}$$

Computing actual successors of all regulatory and switching domains in the system requires time:

$$\begin{gathered}O\left(n2^n(k+1)^n + n((4k+3)^n + (4k+1)^n - (3k+3)^n - (k+1)^n)\right) = \\ O(n(4k+3)^n)\end{gathered}$$

□

3.3.3 Total complexity of Algorithm 1 — proof of (8)

Overall Algorithm 1 complexity is obtained by summing up (1), (2), and (7):

$$\begin{gathered}O\left(ns(k+1)^n + n((2k+1)^n - (k+1)^n) + n(4k+3)^n\right) = \\ O(ns(k+1)^n + n(4k+3)^n)\end{gathered}$$

Since the upper bound of the number of step functions s is usually relatively small (units to tens) and the number of thresholds in average is larger then 1 or 2, we can bound this by $O(n(4k+3)^n)$.

3.4 Proofs of complexity results for Algorithm 2

These proofs differ from those of the previous subsection only in the number of neighboring domains we have to investigate. The simplification provided by the algorithm is based on omitting all the transitions between domains which differ in dimension by more than 1. So we restrict our choice of successors of a domain D to domains D' that have one more or one less switching variable than D. This leads to the upper bound of the number of successors $2n$ since in each transition exactly one variable can change by no more than ± 1.

3.4.1 Computing successors — proofs of (9), (11) *and* (13)

Proof of (9) **(computing successors of regulatory domain D)** Each regulatory domain has at most n successors of dimension $n-1$ (each variable can only change in the direction to the focal point). Stacking them takes time $O(n^2)$, for all regulatory domains $O(n^2(k+1)^n)$. $\square$

Proof of (11) **(computing successors of switching domain D)** For a domain with m switching variables we can change each variable x_i by ± 1. The transition to D' in the direction $w = D' - D$ then exists iff

- for x_i non-switching in $D(\mathrm{dsp}(D) > 1) \wedge (v_i^\perp \leq w_i \leq v_i^\top)$ holds
- for x_i switching in $D(\mathrm{dsp}(D') > 1) \wedge \big((v_i^\perp \leq w_i \leq v_i^\top) \vee (v_i^\perp \leq 0 \leq v_i^\top)\big)$ holds.

This leads to at most $2(n-m) + 2m = 2n$ transitions tested in time $O(2n)$ and stacked (successor must be pushed onto the stack) in time $O(2n^2) = O(n^2)$. $\square$

Proof of (13) **(computing successors for all domains)** Since it takes time $O(n^2)$ to compute and stack all successors for every regulatory and switching domain it takes $O(n^2(2k+1)^n)$ for all the domains in the system. $\square$

3.4.2 Total complexity of Algorithm 2 — proof of (14)

Overall complexity of Algorithm 2 is given by summing up (1), (2), and (13):

$$O\left(ns(k+1)^n + n\left((2k+1)^n - (k+1)^n\right) + n^2(2k+1)^n\right) =$$
$$O\left(ns(k+1)^n + n^2(2k+1)^n\right)$$

It can be approximated due to the relation between s and k to $O(n^2(2k+1)^n)$.

4 GeNeSim

DiVinE Tool (`http://anna.fi.muni.cz/divine`) is a parallel, distributed-memory enumerative model-checking tool for verification of concurrent systems. The tool employs aggregate power of network-interconnected workstations to verify systems whose verification is beyond capabilities of sequential tools. System properties can be specified either directly in Linear Temporal Logic (LTL) or alternatively as processes describing undesired behavior of systems under consideration (negative claim automata). In fact, the tool can check properties expressible in Linear Time Mu-calculus (Büchi automata). From the algorithmic point of view, the tool is quite unique. It implements a variety of novel parallel algorithms for cycle detection (LTL model checking).

GeNeSim is build on the top of the DiVinE library that offers common functions needed to develop a parallel or distributed enumerative model checker. The only extension to the library that was necessary, was the extension of the state generator to a state generator tailored for the specific input provided by GeNeSim GUI. In particular, there are two versions of the state generator – one implements Algorithm 1 and the other implements Algorithm 2 presented in Section 2.2.1. More detailed information on GeNeSim is given in [2].

GeNeSim GUI (`http://anna.fi.muni.cz/genesim`) is an online web application in PHP using MySQL as data storage. Models can be input manually through structured forms or imported automatically from GNA [14]. GUI output is in the form of XML or GNA. The XML form can be directly read by the GeNeSim state generator embedded to DiVinE.

5 Experiments

To compare practicability of the two GeNeSim implementation variants based on the two algorithms for QSTS construction analyzed in this paper, we have conducted experiments checking two different kinds of properties in a random extension of a real model example, and in an automatically generated artificial model with uniformly interconnected genes. We have employed the OWCTY algorithm [11], a distributed algorithm for state space exploration implemented in DiVinE, which is an extended enumerative version of the One Way Catch Them Young Algorithm [18]. We have model checked whether QSTSs of both PLDE models satisfy two different LTL properties. In all cases, a single qualitative state has been considered as the initial condition, in particular, a state with minimal concentration of all protein species.

In DiVinE, LTL properties are encoded as Büchi automata. A source of LTL properties which are of interest for analysis of regulatory networks can be found, e.g., in [5]. We have selected two properties – stability of a particular protein concentration and an example of concentration oscillation. An example of a stability property is description of a system behavior such that for the specified initial condition concentration of a substrate x in all solutions will finally never exceed the threshold θ^1. Such a property can be expressed as LTL formula $\mathrm{FG}(x \leq \theta^1)$ which we denote (c). The atomic proposition of this formula states that the actual concentration level of the state variable x is less or equal than the threshold level θ^1. Negation of this formula, written as $\mathrm{GF}(x > \theta^1)$ and denoted (a), expresses the property that in each solution the concentration of x does not stabilize at level $x \leq \theta^1$.

Oscillation properties express cyclic behavior and are of high importance when analyzing and validating models of biochemical processes (e.g., the circadian rhythm). In Fig. 5, there is an example of a Büchi automaton describing a property of this kind. The rightmost state of the automaton is an accepting state. The automaton states that whenever in the solution the concentration of x exceeds the threshold θ^1, there must immediately follow some oscillation of x around θ^1. Negation of this property guarantees non-existence of permanent oscillation of x around θ^1. We will refer this negated property as (b).

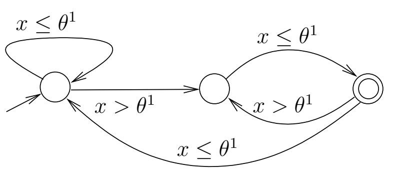

Fig. 5. A Büchi automaton expressing oscillation

For the purpose of our experiments, we have randomly modified a four-variable model of genetic regulation in bacteriophage lambda (thth) (according to Thieffry and Thomas [27]) providing high mutual interaction of genes. In particular, we have considered random modifications of this model with 6 and 8 state variables. Moreover, we have generated a scalable artificial model (chain) with uniform interconnection of genes (a linear chain in which each gene negatively interacts with itself and with the succeeding gene by AND-composition of the respective step functions). For this model we have considered settings with 10, 12, 15, 17, and 20 variables. We have run the experiments on a cluster which offers up-to 20 homogeneous workstations, in particular, dual core Pentium 4 2×2.6 Hz with 4GB RAM. The experiments have been scaled for 1, 5, 10, 15, and 20 nodes.

	Algorithm 1						Algorithm 2					
Exp.	**States**	#1	#5	#10	#15	#20	**States**	#1	#5	#10	#15	#20
thth6a	13230	28.4	7.1	5.4	4.7	4.6	138	0.1	$\perp$[1]	$\perp$	$\perp$	$\perp$
thth8a	117390	1249.9	309.3	190.7	163.7	151.3	24166	17.6	4.8	2.7	$\perp$	$\perp$
chain10a	152191	3972.2	1003	683.6	555.8	494.9	34503	30.7	7.9	5.0	$\perp$	$\perp$
chain12a	1878527	$\top$[2]	$\top$	$\top$	$\top$	11.3 h	230827	238.6	68.8	37.8	27.5	$\perp$
chain15a	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	3977969	4866.7	1021.9	546.7	371.9	355.7
chain17a	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	12518979	6.8 h	5132.2	2675.3	1840.3	1779.8
thth6b	7996	20.2	5.1	3.4	2.5	2.5	153	0.1	$\perp$	$\perp$	$\perp$	$\perp$
thth8b	69410	870	206.1	131.1	110.1	108.6	14769	14.1	3.7	2.4	$\perp$	$\perp$
chain10b	109008	1807.4	524.6	354.4	285.2	253.4	22486	5.2	1.5	$\perp$	$\perp$	$\perp$
chain12b	1267000	$\top$	$\top$	$\top$	$\top$	7.2 h	154022	57.1	12.8	6.8	$\perp$	$\perp$
chain15b	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	2651946	1949	412	212.7	150.7	124.5
chain17b	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	8345888	9842.5	2090.8	1069	738.3	625.7
chain20b	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	283742998	$\top$	$\top$	$\top$	$\top$	11.5 h

Table 1
Comparison of results achieved with Algorithm 1 and Algorithm 2

[1] $\perp$ denotes values which do not significantly improve results achieved on less nodes.
[2] $\top$ stands for values greater than 12 hours.

The experiments based on Algorithm 1 showed that the property (a) is satisfied in both variants of the thth model (the same gene variable has been substituted for x). Using Algorithm 2 we have achieved faster computation and lower numbers of states explored. However, because of the underapproximation the latter positive results cannot be used to argue about the respective PLDE models. To this end, we have additionally run the model checking of the negative property (c) using Algorithm 2. For both models the results have been found negative, hence we can conclude that in both cases there *exists* a path in the original QSTS satisfying the formula (a). However, such an existential result cannot be interpreted universally. Moreover, due to the overapproximation inherent to both algorithms, the counterexample must be further analyzed to ensure that it represents a real solution of the PLDE system.

In the case of the 10 and 12-gene variants of the chain model, the formula (a) was found false by using both algorithms. For all the higher variants, we were unable to get the results in reasonable time by using Algorithm 1. By employing Algorithm 2 we were able to tackle up-to 17 genes. All the achieved answers have been positive, which as explained above does not say anything about the PLDE models. Therefore also in these cases we checked for the negative property (c) using Algorithm 2. The results have been found negative, hence we have the same conclusion as in model checking of thth. Summary of the experiments is presented in the first half of Table 1 and in Table 2. There are showed numbers of states which were explored and times of computation. Each result is taken from a single run of model checking using a particular state generation algorithm as described in the table heading.

Exp.	**States**	**#1**	**#5**	**#10**	**#15**	**#20**
thth6c	275	0.2	$\perp$	$\perp$	$\perp$	$\perp$
thth8c	16579	18.9	5.1	3.6	$\perp$	$\perp$
chain15c	4034980	14902.2	3154.2	1620.8	1126.4	960.4
chain17c	12690022	$\perp$	15897.4	8277.2	5659.4	4815.2

Table 2
Additional results achieved with Algorithm 2 for property (c)

In the case of the oscillation property (b), we checked for non-existence of the requested oscillation in concentration of a selected variable. By employing Algorithm 1, the oscillation was not found in any experimented model (we got positive results for (b)). By employing Algorithm 2, we also got positive results, and moreover, we have been capable of checking the 20-gene model till the limit of 12 hours. Nevertheless, the positive results achieved by Algorithm 2 cannot be directly interpreted from the same reasons as mentioned above. Summary of the experiments is given in the second half of Table 1.

Concerning the memory consumption the experiments showed that, in average, by using Algorithm 1 seven times less states were generated in contrast to Algorithm 2. The cluster memory needed for the worst case (chain20b) was 48 GB. Every computation that was finished before the time limit did not run out of memory. Much more critical was time. We encountered that both algorithms have similar scaling. Algorithm 2 speeded up the answers more than a hundred times in average. In the case of the chain model, this allowed to complete (before the time limit) checking of models having five more genes (property (*a*)) and eight more genes (property (*b*)) than was possible with Algorithm 1. However, because of the positive results in the cases of models having more than 12 genes, we were still unable to decide if both properties are fully satisfied by the PLDE model.

6 Conclusions

In this paper, we have presented time complexity analysis of two algorithms for generating of approximative discrete solution space of piecewise-linear models of transcriptional regulation. We believe that the analysis brings a new algorithmic insight into the qualitative modeling approach well-studied in computational biology. In particular, we started by refining the algorithm introduced by de Jong et.al. [15] into a more detailed version which includes construction of all the necessary computational structures. In order to fight the inevitable exponential complexity of the algorithm, we have introduced a simplified version which produces underapproximation of the piecewise-linear solution space by reducing the number of possible transitions among piecewise-linear partitions. We have compared the complexity of both algorithms. As both algorithms enable us to view the complex biological process of transcriptional regulation as a discrete concurrent system, we have adapted the parallel on-the-fly LTL model-checking, which is a well-known method in the concurrency community, for analysis of biologically interesting properties of GRNs.

Our results extend the previously achieved results on sequential model checking [6,12,20,25] of biological networks. With respect to the same level of modeling abstraction, our approach is comparable with [6]. Other approaches deal with more abstract models like Boolean networks or they deal with metabolic and signalling pathways which require different approaches to qualitative modeling. As in [6], we deal with piecewise-linear models of GRNs. We add the possibility of the on-the-fly parallel approach which has the following advantages. First of all, by generating the state space on-the-fly, we avoid dealing with the entire state space that is exponentially large w.r.t. number of variables. At second, by using the scalable distribution of the computation we fight against running out of memory. Finally, as time of successor generation becomes critical when using on-the-fly approach, we introduce an underapproximative algorithm which may be used when faster analysis is required.

Owing to the fact that the scaling achieved is not ideal, we leave for future work taking advantage of locality of the qualitative state space. Especially, we aim to equip computing nodes with hashed information regarding the local aspects of successor generation. We believe that such improvements will speed up the computation. Other improvements which we plan to consider to reduce the model checking time and memory resources are model-driven and property-driven reduction methods. With respect to the fact that time seems to be more critical than memory, we aim to employ multi-core version of DiVinE.

References

[1] U. Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman & Hall/CRC, 2006.

[2] J. Barnat, L. Brim, I. Černá, S. Dražan, and D. Šafránek. Parallel Model Checking Large-Scale Genetic Regulatory Networks with DiVinE. In *Proc. of From Biology to Concurrency and Back*, volume 194 of *ENTCS*, pages 35–50. Elsevier, 2007.

[3] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *Computer Aided Verification*, volume 4144/2006 of *LNCS*, pages 278–281. Springer, 2006.

[4] G. Batt, C. Belta, and R. Weiss. Model checking genetic regulatory networks with parameter uncertainty. In *Workshop on Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, pages 61–75. Springer, 2007.

[5] G. Batt, C. Belta, and R. Weiss. Model checking liveness properties of genetic regulatory networks. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 323–338. Springer, 2007.

[6] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider. Analysis and verification of qualitative models of genetic regulatory networks: A model-checking approach. In *Conference on Artificial Intelligence*, pages 370–375, 2005.

[7] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider. Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in escherichia coli. In *Conference on Intelligent Systems for Molecular Biology*, pages 19–28, 2005.

[8] 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.

[9] James M. Bower and Hamid Bolouri. *Computational modeling of genetic and biochemical networks*. Cambridge : Bradford Book, 2001.

[10] L. Calzone, F. Fages, and S. Soliman. BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.

[11] I. Černá and R. Pelánek. Distributed explicit fair cycle detection (set based approach). In *Model Checking Software. 10th International SPIN Workshop*, volume 2648 of *LNCS*, pages 49 – 73. Springer, 2003.

[12] N. Chabrier and F. Fages. ”symbolic model checking of biochemical networks”. In *Computational Methods in Systems Biology (CMSB’03)*, volume 2602 of *LNCS*, pages 149–162. Springer, 2003.

[13] H. de Jong, J. Geiselmann, G. Batt, C. Hernandez, and M. Page. Qualitative simulation of the initiation of sporulation in Bacillus subtilis. *Bulletin of Mathematical Biology*, 66(2):261–300, 2004.

[14] H. de Jong, J. Geiselmann, C. Hernandez, and M. Page. Genetic network analyzer: qualitative simulation of genetic regulatory networks. *Bioinformatics*, 19(3):336–344, 2003.

[15] H. de Jong, J-L. Gouz, C. Hernandez, T. Sari, M. Page, and J. Geiselmann. Qualitative simulation of genetic regulatory networks using piecewise-linear models. Technical Report RR-4407, INRIA-Helix, 2002.

[16] *FP6 Project EC-MOAN*, 2007. `http://www.ec-moan.org`.

[17] S. Williams et.al. The Potential of the Cell Processor for Scientific Computing. In *Proc. of Conference on Computing Frontiers*, pages 9–20. ACM Press, 2006.

[18] K. Fisler, R. Fraer, G. Kamhi, M. Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In *Proc. of Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 420–434. Springer, 2001.

[19] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(No. 25):2340–2381, 1977.

[20] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In *Computational Methods in Systems Biology*, volume 4210 of *Lecture Notes in Bioinformatics*, pages 32–47. Springer, 2006.

[21] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–4, 2002.

[22] J.S. Meredith, S.R. Alam, and J.S. Vetter. Analysis of a Computational Biology Simulation Technique on Emerging Processing Architectures. In *Proc. of Parallel and Distributed Processing Symposium*, pages 1–8. IEEE, 2007.

[23] T. Mestl, E. Plahte, and S.W.Omholt. A mathematical framework for describing and analysing gene regulatory networks. *Journal of Theoretical Biology*, 176(2):291–300, 1995.

[24] D. Ropers, H. de Jong, M. Page, D. Schneider, and J. Geiselmann. Qualitative simulation of the carbon starvation response in escherichia coli. *Biosystems*, 84(2):124–52, 2006.

[25] M. Schaub, T. Henzinger, and J. Fisher. Qualitative networks: a symbolic approach to analyze biological signaling networks. *BMC Systems Biology*, 2007.

[26] C. P. Sosa, T. Milledge, J. McAllister, and G. Z. Milledge. Life Sciences Molecular Dynamics Applications on the IBM System Blue Gene Solution: Performance Overview. Technical report, IBM, 2006.

[27] D. Thieffry and R. Thomas. Dynamical Behavior of Biological Regulatory Networks. *Bulletin of Mathematical Biology*, 57(2):277–297, 1995.