Computational Analysis of Large-Scale Multi-Affine ODE Models

J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková, and D. Šafránek
Faculty of Informatics
Masaryk University
Brno, Czech Republic
Email: safranek@fi.muni.cz

***Abstract*—A biological system as considered in systems biology is understood in the form of a network of interactions among individual biochemical species. Complexity of these networks is inherently enormous, even for simple (e.g., procaryotic) organisms. When modeling and analyzing dynamics of these networks, i.e., exploring how the species evolve in time, we have to fight even another level of complexity – the enormous state space. In this paper we deal with a class of biological models that can be described in terms of multi-affine dynamic systems. First, we present a prototype tool for parallel (distributed) analysis of multi-affine systems discretized into rectangles that adapts the approach of Belta et.al. [1], [2]. Secondly, we propose heuristics that significantly increase applicability of the approach to large biological models. Effects of different settings of the heuristics is firstly compared on a set of experiments performed on small models. Subsequently, experiments on large models are provided as well.**

***Keywords*-biological networks; parallel model checking; dynamic systems; rectangular abstraction**

I. Introduction

The complexity of molecular mechanisms underlying the dynamics of living organisms is enormous. The main reason is not only the number of different biochemical species participating in huge number of biochemical reactions, but especially the implicit parallelism – the symphony in which the reactions compose and collaborate to drive the organism's physiology. Integrative study of these complex compositions is the main challenge of systems biology.

The central interest of systems biology is investigation of the response of the organism to environmental events (extra-cellular or intra-cellular signals). Even in procaryotic organisms, a single environment event causes a response induced by the interaction of several interwoven modules with complex dynamic behavior, acting on rapidly different time scales. In higher organisms, these modules form large and complex interaction networks. 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 interaction network with complex positive and negative regulatory feedback loops.

The most widely-used modeling frameworks for the analysis of the dynamics of biological systems are based on the deterministic continuous approach of ordinary differential equations [3] (ODE). The reduction of continuous models to discrete automata by a sequence of reductions, approximations, and abstractions allows formal methods for the automated analysis of temporal properties to be applied [2], [4]–[7]. When dealing with large models from systems biology, standard discrete state-space exploration techniques do not provide acceptable response times for answering user queries and high-performance parallel algorithms are required. Owing to dynamical dependencies among state variables, the *state-space explosion problem* arises during reduction to discrete automata. In general, there are two reasons for the state explosion – number of state variables and density of the discretization mesh. In fact, even relatively small ODE models containing tens of state variables lead to large automata having millions of states. In other words, models that represent dynamics of even smaller parts of complex biological networks usually appear to computer science as *large-scale* models.

In this paper we present a prototype tool for parallel analysis of biological models based on mass action kinetics. In particular, the tool adapts the rectangular abstraction approach of multi-affine ODEs mathematically introduced in [1] and algorithmically tackled in [2], [5]. We contribute to the domain by means of a scalable algorithm. In particular, our contribution is three-fold. First, we introduce a parallel on-the-fly state space generator for the rectangular abstraction. Second, we propose several heuristics for reducing the extent of approximation by guiding the state generator to avoid spurious simulations. Third, we show by means of experiments how parallelization fights the state space explosion. To the best of our knowledge, this is the first attempt to employ parallel techniques for analysis of rectangular abstractions of general multiaffine systems.

A. Related Work

Rectangular abstraction of multiaffine systems has been employed in [2] for reachability analysis. The result has been supported by experiments performed on a sequential implementation in Matlab. The provided experiments have showed that for models with 10 variables the reachability analysis ran out of memory after 2 hours of computation. This gave us the motivation to employ parallel algorithms. Moreover, we generalize the analysis method to general LTL model checking.

It is also worth noting that our abstraction slightly differs from the approach stated in [2]. In particular, we assume the abstracted states to be closed rectangles. We do not distinguish boundary regions (i.e., facets and their boundaries) as individual states which in turn gives us exponentially smaller state space while still preserving almost all trajectories. Such an approach to rectangular abstraction has been previously introduced in [8] in the context of piece-wise multiaffine systems. It is worth pointing out that this approach does not provide full overapproximation, but overapproximation up to the set of continuous trajectories of measure zero w.r.t. the entire solution space.

In our previous work [9] we have dealt with parallel model checking analysis of piece-wise affine ODE models [10]. The method allows fully qualitative analysis, since in the piece-wise affine approximation generating of the state space does not require to numerically enumerate the equations. Therefore that approach, in contrast to this one, is primarily devoted for models with unknown kinetic parameters. The price for this feature is higher time complexity of the state space generation. In particular, time appears there more critical than space while causing the parallel algorithms not to scale well. In the setting of this paper, the employed abstraction method relies on numerical evaluation of ODEs at intersection points of the partition mesh. In particular, the continuous function is evaluated many times in order to compute the transitions. This is the main reason why the enumerative method is inevitable.

We assume all the kinetic parameters to be numerically specified. In such a situation there is an alternative possibility to do LTL model checking directly on numerical simulations [6], [11]. However, in the case of unknown initial conditions there appears the need to provide large-scale parameter scans resulting in huge number of simulations. On the contrary, the analysis can be naturally generalized to arbitrary intervals of initial conditions by means of rectangular abstraction.

II. Preliminaries

A. Biochemical models as multi-affine systems

We can model a system of biochemical reactions in terms of an ODE system. We consider a special class of ODE systems in the form $\dot{x} = f(x)$ where $x = (x_1, \ldots, x_n)$ is a vector of variables and $f = (f_1, ..., f_n) : \mathbb{R}^n \to \mathbb{R}^n$ is a vector of multiaffine functions. A multiaffine function is a polynomial in the variables $x_1, ..., x_n$ where the degree of any variable is at most 1. Variables x_i represent concentrations of species. Multiaffine ODEs can express reactions in which the stoichiometry coefficients of all reactants are at most 1. ODEs can be constructed directly from the stoichiometric matrix of the biochemical system [12]. An example of a multi-affine system is given in Figure 1.

$$A + B \xrightarrow{k} C \qquad \begin{aligned} \frac{dA}{dt} &= -k \cdot A \cdot B \\ \frac{dB}{dt} &= -k \cdot A \cdot B \\ \frac{dC}{dt} &= k \cdot A \cdot B \end{aligned}$$

Figure 1. Example of a multi-affine system

B. Rectangular abstraction of multi-affine systems

Let us consider a bounded part of the phase space of the ODE system $\dot{x} = f(x)$ given by an n-dimensional rectangle $\prod_{i=1}^{n}[min_i, max_i] \subset \mathbb{R}^n$. *The ODE model*, denoted $\mathcal{M}$, is given by an ODE system, a *set of thresholds* $\theta_j^i \in \mathbb{R}$ satisfying $min_i = \theta_1^i < \theta_2^i < \ldots < \theta_{\zeta_i}^i = max_i$ for each variable x_i, and a *set of initial regions*. The number of tresholds specified for the variable x_i is denoted ζ_i, $\zeta_i \geq 2$.

Let us denote Ω the set $\prod_{i=1}^{n}\{1, \ldots, \zeta_i - 1\}$. Each multi-index $\alpha \in \Omega$ uniquely labels a rectangle in the phase space of $\mathcal{M}$. For each $\alpha \in \Omega$ we use several ways of realization of α in $\mathbb{R}^n$. The function $Re(\alpha)$ assigns a closed rectangular subset of $\mathbb{R}^n$ (a *rectangle*) to α, $Re(\alpha) \overset{\text{df}}{=} \prod_{i=1}^{n} \left[\theta_{\alpha_i}^i, \theta_{\alpha_i+1}^i\right]$. The notation $Re_{\mathcal{V}}(\alpha)$ then denotes the set of vertices of $Re(\alpha)$, $Re_{\mathcal{V}}(\alpha) \overset{\text{df}}{=} \prod_{i=1}^{n}\{\theta_{\alpha_i}^i, \theta_{\alpha_i+1}^i\}$. Finally, $Re_v(\alpha)$ denotes a unique point in $\mathbb{R}^n$ that represents $Re(\alpha)$, $Re_v(\alpha) \overset{\text{df}}{=} (\theta_{\alpha_1}^1, \ldots, \theta_{\alpha_n}^n)$.

The *set of initial regions* of $\mathcal{M}$, denoted $\text{Inset}(\mathcal{M})$, is defined as a finite set of k n-dimensional rectangles $[\gamma_\perp^j, \gamma_\top^j] \subset \mathbb{R}^n$ such that $(\gamma_\perp^j, \gamma_\top^j) \in \Omega \times \Omega$ for each j, $1 \leq j \leq k$.

Note that every pair $\alpha, \beta \in \Omega$ such that $\exists 1 \leq j \leq n. \beta_j = \alpha_j \pm 1$ and $\forall 1 \leq i \leq n, i \neq j. \beta_i = \alpha_i$ satisfies the equality $||\alpha - \beta|| = 1$ where $||x||$ denotes the Euclid norm of $x \in \mathbb{R}^n$. For such α, β we say that $Re(\alpha)$ and $Re(\beta)$ are *neighboring rectangles*. Neighboring rectangles satisfy $Re(\alpha) \cap Re(\beta) \neq \emptyset$. The intersection of their realization is a hyperrectangular facet of dimension $(n-1)$. Let us remind that the inequality $||\alpha - \beta|| \leq 1$ means either $\alpha = \beta$ or α, β are just neighboring rectangles.

Let $\alpha, \beta \in \Omega$ be such that $||\alpha - \beta|| = 1$ with $1 \leq i \leq n$, $j \in \{-1, 1\}$ satisfying $\beta_i = \alpha_i + j$. We denote $of(\alpha, i, j)$ the *outgoing facet of* α *along* x_i *in direction* j, $of(\alpha, i, j) \overset{\text{df}}{=} Re(\alpha) \cap Re(\beta)$. The notation $of_{\mathcal{V}}(\alpha, i, j)$ then denotes the set of vertices of the respective outgoing facet, $of_{\mathcal{V}}(\alpha, i, j) \overset{\text{df}}{=} Re_{\mathcal{V}}(\alpha) \cap Re_{\mathcal{V}}(\beta)$.

Rectangle $Re(\alpha)$ is called *transient* if for every solution of $\dot{x}(t) = f(x)$ satisfying $x(t_0) \in Re(\alpha)$ for some $t_0 \in \mathbb{R}$ there exists $t_1 > t_0$ such that $x(t_1) \notin Re(\alpha)$.

C. Rectangular abstraction transition system

The *rectangular abstraction transition system* (RATS) corresponding to an ODE model $\mathcal{M}$ is a triple (Q, T, I) where $Q \subseteq \Omega$ is the *set of states*, $I \subseteq Q$ is the *set of initial states*, $I \overset{\text{df}}{=} \{\alpha \in Q \mid \exists(\gamma_\perp^j, \gamma_\top^j) \in \text{Inset}(\mathcal{M}) : \forall 1 \leq i \leq$

$n.\gamma^j_{\perp i} \leq \alpha_i < \gamma^j_{\top i}\}$, and $T \subseteq Q \times Q$ is the *transition relation*.

The relation T contains only those pairs $\langle \alpha, \beta \rangle$ for which $||\alpha - \beta|| \leq 1$ and either of the following conditions holds:

1) $||\alpha - \beta|| = 1$ with $1 \leq i \leq n$, $j \in \{-1, 1\}$ such that $\beta_i = \alpha_i + j$ and there exists $v \in of_{\mathcal{V}}(\alpha, i, j)$ satisfying $f_i(v) \cdot j > 0$.
2) $||\alpha - \beta|| = 0$ and $\vec{0} \in hull\{f(v) \mid v \in Re_{\mathcal{V}}(\alpha)\}$ where *hull* denotes the convex hull of the given set.

We will often denote the fact $\langle \alpha, \beta \rangle \in T$ as $\alpha \rightarrow \beta$. Moreover, we use the notation $\alpha \rightarrow^{i,j} \beta$ to denote $\langle \alpha, \beta \rangle \in T$ such that $\beta_i = \alpha_i + j$.

Condition (1) states that there is a transition from α to β if there exists at least one vertex v of the facet $Re(\alpha) \cap Re(\beta)$ such that the sign of the respective component of the vector field agrees at v with the direction of the transition. Note that multiaffinity guarantees that vector field of any facet is a linear combination of the vector fields at its vertices. Therefore we consider just vertices of the outgoing facet.

Condition (2) implies inclusion of the self-loop $\langle \alpha, \alpha \rangle \in T$ to reflect the possibility of non-transiency of $Re(\alpha)$ if there exists a fixed point in $Re(\alpha)$. Convex hull, understood as the minimal convex set containing the vectors at the rectangle vertices, is used to obtain all possible linear combinations of the vector field in the rectangle. This is achieved by taking just the rectangle vertices as the base.

It is known that RATS represents an overapproximation with respect to trajectories of the original ODE model. For model checking of RATS we use Linear Temporal Logic (LTL). Note that LTL can be also directly interpreted on trajectories of ODE models [6], [11]. Given an ODE model $\mathcal{M}$ we can say that $\mathcal{M}$ satisfies a formula φ, written $\mathcal{M} \models \varphi$, only if all trajectories starting at the initial region satisfy φ. In the context of RATS, a formula φ is satisfied by RATS($\mathcal{M}$), written RATS($\mathcal{M}$) $\models \varphi$, only if each execution starting from any initial state satisfies φ. The following theorem characterizes the relation between validity of φ in the RATS and in the original ODE model.

Theorem 1: Consider an ODE model $\mathcal{M}$ and the respective RATS($\mathcal{M}$). If RATS($\mathcal{M}$) $\models \varphi$ then $\mathcal{M} \models \varphi$.

III. A Tool for RATS Analysis

In this section we describe a prototype implementation of the tool for analysis of rectangular abstraction transition systems. It is based on `DiVinE` [13] (Distributed Verification Environment), a parallel distributed-memory enumerative model-checking tool.

`DiVinE` employs aggregate power of network-interconnected workstations (nodes) to analyze large-scale state transition systems whose exploration is beyond capabilities of sequential tools. System properties can be specified either directly in LTL or alternatively as processes describing undesired behavior of systems under consideration (negative claim automata). From the algorithmic point of view, the tool implements a variety of parallel algorithms [14], [15] for cycle detection. By these algorithms, the entire state space is uniformly split into partitions and every partition is distributed to a particular computing node. Each node is responsible for generating the respective state-space partition on-the-fly while storing visited states into the local memory.

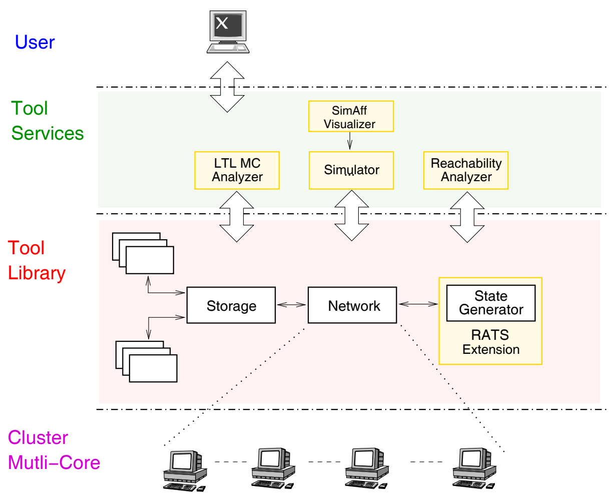

Figure 2. Tool Architecture

To adapt `DiVinE` algorithms for analysis of biological models, we have developed a prototype extension of `DiVinE` by introducing a new state-space generator. The new state generator constructs the RATS for a given multi-affine ODE model. The tool architecture is showed in Figure 2. Library-level components are responsible for constructing, managing and distributing the state space. They form the core of the tool, and with the only exception of state generator, they are taken directly from the `DiVinE` library. Services of the library are used by the application components allowing simulation as well as model-checking analysis of the reachable state space. The tool provides a graphical interface component *SimAff* allowing visualization of the simulation results.

The input model is specified by the following data:

- list of variables,
- list of (multi-affine) ODEs,
- list of partitioning thresholds given for each variable,
- list of initial rectangular subspaces (the union of these subspaces forms the initial condition),
- Büchi automaton representing an LTL property (this data is not needed for simulation).

An example of a simple three-variable model representing a single biochemical reaction $A + B \rightarrow C$ performed with rate $0.5\ M^{-1}s^{-1}$ is showed in Figure 3 on the left. For each variable there is specified the equation as well as the list of real values representing individual threshold positions. The initial condition is defined in this particular case by a single rectangular subspace stating $A \in \langle 6, 10\rangle, B \in \langle 4, 6\rangle$, and $C \in \langle 0.0001, 2\rangle$. As a consequence, the initial interval set is a singleton, $\mathrm{Inset} = \{\langle\langle 6, 4, 0.0001\rangle, \langle 10, 6, 2\rangle\rangle\}$.

A. State Space Generator

The state-space generator enumerates RATS on-the-fly starting in every initial state and therefore the state-space graph is encoded implicitly by a function that for a given state returns all its successor states. This function is the heart of the state generator. The set of initial states is given directly by all states satisfying the initial condition.

Problem 1: Let $\mathcal{M}$ be a multi-affine model with n variables and $\mathrm{Inset}(\mathcal{M})$ the initial condition. We want to construct the transition system $RATS(\mathcal{M}) \equiv \langle Q, T, I\rangle$ representing the dynamics of $\mathcal{M}$.

The procedure for computing the successor states for a state $q \in Q$ is the following:

```
**procedure** getSuccessorStates($q \equiv \langle l_1, ..., l_n\rangle \in Q$)
 1: init $Succs := \emptyset$ {initialize the set of successor states}
 2: for $i \in \{1, ..., n\}$ do
 3:     if decideSelfLoop($q$) then
 4:         $Succs := Succs \cup \{q\}$ {add selfloop to $q$}
 5:     end if
 6:     if $l_i + 1 < \zeta_i$ then {if not at maximal level}
 7:         init $q' := q$
 8:         init $\mathcal{F} := of_{\mathcal{V}}(q, i, 1)$
 9:         while $q' = q$ and $\mathcal{F} \neq \emptyset$ do
10:             choose $v \in \mathcal{F}$
11:             if $f_i(v) > 0$ then
12:                 $q'_i := l_i + 1$
13:                 $Succs := Succs \cup \{q'\}$ {transition increasing $x_i$}
14:                 break
15:             end if
16:             $\mathcal{F} := \mathcal{F} \setminus \{v\}$
17:         end while
18:     end if
19:     if $l_i - 1 > 0$ then {if not at minimal level}
20:         init $q' := q$
21:         init $\mathcal{F} := of_{\mathcal{V}}(q, i, -1)$
22:         while $q' = q$ and $\mathcal{F} \neq \emptyset$ do
23:             choose $v \in \mathcal{F}$
24:             if $f_i(v) < 0$ then
25:                 $q'_i := l_i - 1$
26:                 $Succs := Succs \cup \{q'\}$ {transition decreasing $x_i$}
27:                 break
28:             end if
29:             $\mathcal{F} := \mathcal{F} \setminus \{v\}$
30:         end while
31:     end if
32: end for
33: return $Succs$
```

The algorithm implements the abstraction introduced in Section II. However, there is one point that requires special discussion, in particular, the construction of selfloops on line 3. As it has been stated in Section II, selfloops are needed to ensure that trajectories converging to some fixed point are reflected by the abstraction procedure. A necessary condition for transiency of a rectangle requires computation of its convex hull. The complexity of optimal algorithm [16] for convex hull of 2^n points in $\mathbb{R}^n$ is $O(2^{n\lfloor n/2\rfloor})$. Executing of such a procedure would greatly increase the time complexity of state space generator. Since reachability and safety analysis do not rely on transiency, the guarding condition at line 3 can be relaxed to $True$, thus introducing selfloops to all states. For temporal safety and liveness analysis we compromise the decision by the following procedure:

```
**procedure** decideSelfLoop($q \in Q$)
 1: for $i \in \{1, ..., n\}$ do
 2:     init $\mathcal{F}_u := of_{\mathcal{V}}(q, i, 1)$
 3:     init $\mathcal{F}_l := of_{\mathcal{V}}(q, i, -1)$
 4:     init $po_u := 0$, $ne_u := 0$, $po_l := 0$, $ne_l := 0$
 5:     while $\mathcal{F}_u \cup \mathcal{F}_l \neq \emptyset$ do
 6:         choose $v_u \in \mathcal{F}_u$ and $v_l \in \mathcal{F}_l$
 7:         if $f_i(v_u) > 0$ then
 8:             $po_u := po_u + 1$
 9:         else if $f(v_u) < 0$ then
10:             $ne_u := ne_u + 1$
11:         end if
12:         if $f_i(v_l) > 0$ then
13:             $po_l := po_l + 1$
14:         else if $f_i(v_l) < 0$ then
15:             $ne_l := ne_l + 1$
16:         end if
17:         $\mathcal{F}_u := \mathcal{F}_u \setminus \{v_u\}$, $\mathcal{F}_l := \mathcal{F}_l \setminus \{v_l\}$
18:     end while
19:     if $(po_u \cdot po_l > 0 \wedge ne_u + ne_l = 0) \vee (ne_u \cdot ne_l > 0 \wedge po_u + po_l = 0)$
        then
20:         return $False$
21:     end if
22: end for
23: return $True$
```

The intuition behind this procedure relies on the fact that as soon as we detect a variable for which all vertices of both outgoing facets $\mathcal{F}_u$, $\mathcal{F}_l$ share the same derivation sign, we are sure that the state is transient. In all other cases we include the possibility of non-transiency by imposing the selfloop. More precisely, we test for a stronger necessary condition of transiency of $\alpha \in Q$: If there exists i such that $f_i(v) > 0$ for all vertices v of $Re(\alpha)$ or $f_i(v) < 0$ for all vertices v of $Re(\alpha)$ then α is transient.

It is worth noting that the algorithm never exceeds the minimal and maximal threshold levels of all variables (the interest box). To guarantee the overapproximation, the interest box must satisfy the sanity check requiring that derivations of all variables of the system evaluated in the vertices of the interest box must point inside the box [1]. If this is not true, the user is given a warning providing the information stating in which facet of the interest box the sanity is violated. However, as the effect of this "path cutting" is localized, for practical simulations it is still sensible to deal also with models that do not satisfy sanity. Moreover, it is often quite intricate to find a representative interest box that satisfies sanity.

In Figure 3 (on the right) there is an example of RATS generated from the model on the left. Each state $\alpha \in Q$ is displayed in the format $Re_v(\alpha)_1(\alpha_1), ..., Re_v(\alpha)_n(\alpha_n)$. In the implementation we use multiindices starting from 0 ($0 \sim min$).

The procedure getSuccessorStates() has the worst-case time complexity $O(n2^n)$ of generating successors for a given state in an n-dimensional system. The crucial point comes

```
VARS:A,B,C

EQ:dA = (-0.5)*A*B
EQ:dB = (-0.5)*A*B
EQ:dC = 0.5*A*B

TRES:A: 0.0001, 6, 10
TRES:B: 0.0001, 4, 6
TRES:C: 0.0001, 2, 4, 6

INIT: 6:10, 4:6, 0.0001:2
```

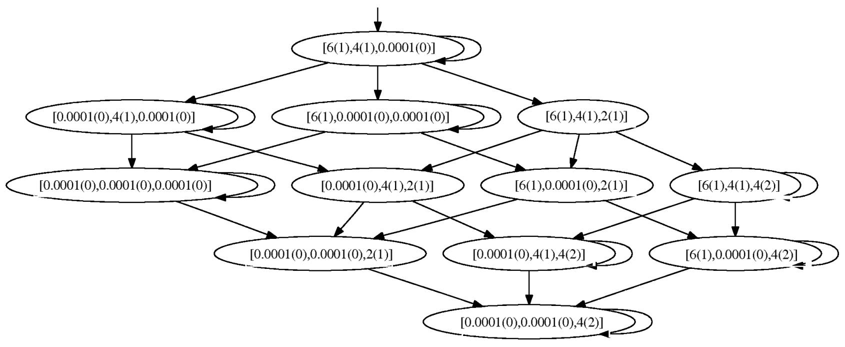

Figure 3. Input model example and resulting RATS representing overapproximation of its dynamics

at lines 9 and 22 where the while loop traverses all vertices of the respective facet. Similar situation occurs at line 5 of the procedure decideSelfLoop().

In the implementation of getSuccessorStates() we use the following observation. Let us fix the state $q \in Q$. For any outgoing facet $\mathcal{F} = of_{\mathcal{V}}(q, i, j)$, $j \in \{-1, 1\}$, we can define the relation $R_i \subseteq \mathcal{F} \times \mathcal{F}$ such that $\langle v, v' \rangle \in R_i \Leftrightarrow f_i(v) = f_i(v')$. Additionally, we consider the set of indices $\mathcal{D}_i \overset{\text{df}}{=} \{k \mid v_k \neq v'_k, \langle v, v' \rangle \in R_i\}$. Intuitively, $\mathcal{D}_i$ identifies those components of $v \in \mathcal{F}$ on which the value of $f_i(v)$ depends. Note that $\mathcal{D}_i$ is equivalent to the set of indices of all variables that appear in the right-hand side of the equation for $\frac{dx_i}{dt}$. Now to find all different values of f_i on $\mathcal{F}$, we have to check only the subset of $\mathcal{F}$ which consists of unique representatives of the equivalence classes given by R_i. That way we achieve $O(n2^d)$ where d is the maximal number of different variables appearing in right-hand side of each equation, $0 \leq d \leq n$. It is sensible to note that in biological models d is usually small since the typical chemical species interact with a restricted subset of other species.

When the condition at line 3 is relaxed to $True$, the computation does not necessarily need to traverse representatives of all equivalence classes of facet vertices. Both while loops are escaped as soon as the condition at line 11 (resp. 24) is satisfied. However, this is not true for the while loop of procedure decideSelfLoop() where representatives of all equivalence classes of vertices must be considered.

B. Abstraction Tuning

When considering the reachable state spaces generated by the algorithms of Section III-A, they typically include many trajectories that are not present in the original ODE model (we speak about *spurious* trajectories). We employ two different approaches to treat this problem.

1) Threshold Refinement: The first approach is conservative (in terms of Theorem 1) and decreases the level of overapproximation while increasing the state space. We employ the iterative algorithm of *threshold refinement* as defined in [2]. In a single iteration the algorithm searches for points in the state space where nullclines intersect the threshold lines. In particular, on every line $C_i = \{v \in \mathbb{R}^n \mid \forall j \neq i, 1 \leq j \leq n, \exists 1 \leq u \leq \zeta_j . v_j = \theta^j_u\}$, $i \leq n$, there is identified a point $w \in C$ where $f(w) = 0$. If such a point exists and moreover lies in the range of the interest box, a new threshold is added to variable x_i. Each effective refinement step decreases the level of overapproximation of the reachable set while increasing the number of rectangles. By iterating the refinement we can either reach a fixed point (the most precise grid), we can underflow the limit of floating point precision (thresholds are placed too near to each other), or we can get stuck in an infinite loop (the most precise grid cannot be reached). We refer the number of refinement iterations as the *refinement depth*.

2) Transition-reducing Approximations: The second approach is not conservative and it provides systematic reduction of the number of successor states by considering only the most significant ones. The method relies on reduction of the number of transitions (and therefore reduction of the reachable state spaces can be also observed). The quantitative measure by which we decide which transitions have to be excluded is the *transition magnitude*. Transition magnitude characterizes the average concentration increase in particular variable measured on the respective outgoing facet. The approach is motivated by the physical behavior of biochemical reactions running on different time-scales, in particular, fast reactions cause “immediate” transition of reaction energy among species while slower reactions appear to this time-scale as steady [17].

Therefore it is necessary not to loose fast transitions, but slow transitions can be, to some extent, abstracted out, while leaving simulations still fruitful. An advantage is a significant reduction of overapproximation of the reachable set, a disadvantage is the loss of conservativeness of the simulation. In particular, the slow transitions are overcharged by the fast ones causing the trajectories that rely on some slow transitions to be abstracted out.

For a state $\alpha \in Q$ the magnitude of the transition along variable x_i is determined by the magnitude function

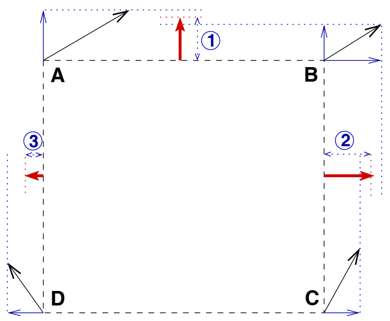

Figure 4. Magnitude function on a rectangle in 2D

$g(\alpha, i, j)$ where $j \in \{-1, 1\}$ denotes the transition direction:

$$g(\alpha, i, j) = avg_{v \in of_{\mathcal{V}}(\alpha,i,j)}(max\{0, j \cdot f_i(v)\}))$$

where avg denotes the arithmetic mean.

The function is computed as the upper bound of the absolute average rate $f_i(x)$ taken over the respective outgoing facet. The motivation for using the mean is the need to approximately characterize the tendency of flow through the outgoing facet. In Figure 4 there is showed an example of magnitude function computation for a 2×2 rectangle. At each vertex there is the vector field with the relevant projection vectors depicted. If we fix the horizontal axis for x_1 and the vertical axis for x_2 then the dimension (1) belongs to $g(\alpha, 2, 1)$, (2) to $g(\alpha, 1, 1)$, and (3) to $g(\alpha, 1, -1)$. The transition magnitude for the side DC is zero because the vector field at both vertices points towards the inside of the rectangle.

Let us denote $\chi_\alpha = max\{g(\alpha, i, j) \mid 0 < i \leq n, j \in \{-1, 1\}\}$ the maximal transition magnitude evolving from the source state α. Criterion for enabling the transition $\alpha \overset{i,j}{\rightarrow} \beta$ is then based on the function $h^s(\alpha, \beta, \sigma)$ where $\sigma \in \mathbb{R}$ is the *significance coefficient*, $0 < \sigma \leq 1$:

$$h^s(\alpha, \beta, \sigma) = \begin{cases} 1 & \text{if } \alpha \overset{i,j}{\rightarrow} \beta \text{ and } g(\alpha, i, j) \geq \sigma \cdot \chi_\alpha, \\ 0 & \text{otherwise.} \end{cases}$$

Generating successors by the decision procedure described above requires inevitably traversing through all vertices of every outgoing facet in order to compute the transition magnitudes. This must be done at the beginning of getSuccessorStates() in the way similar to decideSelfLoop(). Thus, we achieve the time complexity $O(n2^d)$ with n the model dimension and d as introduced in Section III-A. To comment on the space used, to smaller σ corresponds a smaller reachable state space while increasing the risk that we loose some behavior of the original continuous system. In Figure 5 there are depicted state spaces for the model consisting of two reactions $A \overset{k_1}{\rightarrow} B$ and $C \overset{k_2}{\rightarrow} D$ where $k_1 = 1$ and $k_2 = 0.1$, i.e., the former reaction is running on a faster time-scale than the latter. Thresholds of all the variables are set uniformly to $0.0001, 3, 7, 10$. In Figure $5(a)$, the vertical axis stands for A whereas the horizontal axis stands for C. Figure $5(b)$ shows dependency of A (vertical) on D (horizontal). The example demonstrates that the larger σ the smaller the state space. When compared with numerical simulations, both reduced state spaces credibly approximate the most of trajectories up to the segments colored in red.

However, there can be examples where some significant behavior is lost by the approximation. This can happen only when during a sequence of transitions imposed by fast changes at a particular variable, there occurs also a significant event in slow variables. In such a situation, the interesting "slow behavior" can be lost. To overcome this issue we employ the heuristics strategy that allows also the slow transitions to occur, while quantifying the occurrence by a probability distribution based on the magnitude of transitions outgoing from the given source state. Criterion of the heuristics is given by the function $h^p(\alpha, \beta, \sigma)$ which gives probability of enabling the transition $\alpha \overset{i,j}{\rightarrow} \beta$:

$$h^p(\alpha, \beta, \sigma) = \begin{cases} 1, & \text{if } \alpha \overset{i,j}{\rightarrow} \beta \text{ and } g(\alpha, i, j) \geq \sigma \cdot \chi_\alpha, \\ \frac{g(\alpha,i,j)}{\chi_\alpha}, & \text{otherwise.} \end{cases}$$

Using this heuristics, the fast (significant) transitions are always enabled whereas the slow ones are randomly chosen w.r.t. their magnitude. Note that the employed distribution depends only on local reaction rates. To make the distribution more predicative, we relativize the probability of enabling a particular transition against the maximal distance it has to pass through, in particular, the distance between the two adjacent thresholds bounding the source state on the variable affected by the transition. For transition $\alpha \overset{i,j}{\rightarrow} \beta$ affecting x_i we denote this distance Δx_i, $\Delta x_i = |\theta^i_{\beta_i} - \theta^i_{\alpha_i}|$. Criterion for the relativizing heuristics is given by the function $h^r(\alpha, \beta, \sigma)$:

$$h^r(\alpha, \beta, \sigma) = \begin{cases} 1, & \text{if } \alpha \overset{i,j}{\rightarrow} \beta \text{ and } g(\alpha, i, j) \geq \sigma \cdot \chi_\alpha, \\ \frac{g(\alpha,i,j)}{m \cdot \Delta x_i}, & \text{otherwise.} \end{cases}$$

where $m = max\{\frac{g(\alpha,i,j)}{\Delta x_i} \mid 0 < i \leq n, j \in \{-1, 1\}\}$ denotes the maximal relativized rate.

Both heuristics h^p and h^r result in nondeterministic behavior of the state space generator. To implement this randomness correctly in the distributed environment we use a hash function that returns a unique number reflecting the particular distribution. That way for each particular state it is guaranteed that the decision will be the same regardless on which machine the procedure is executed.

We stress that with heuristics based methods the time complexity for state space construction is the same as the deterministic method corresponding to h^s. On the other hand, space complexity is different. For the given σ, the size of the reachable state space of h^p and h^r fits between h^s (the smallest variant) and the full version of the algorithm that gives the largest reachable state spaces.

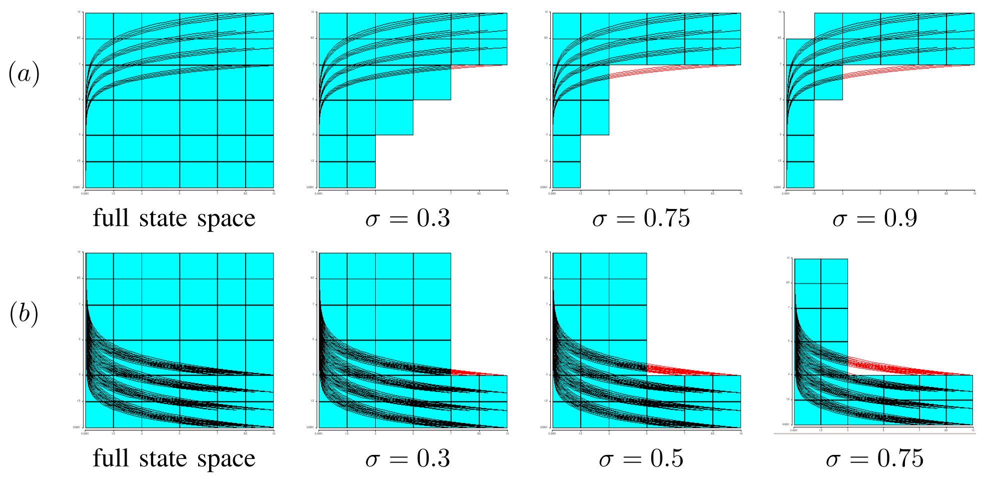

Figure 5. Simulation graphs showing reachable state spaces achieved for different settings of σ

C. Properties Analyzable on RATS

In the similar way like in [6]–[8], [11], we use LTL formulas with atomic propositions interpreted as constraints over real numbers to encode properties analyzed on RATS. In particular, we use propositions in the form $x_i \odot \theta^i_j$ where $\odot \in \{\leq, \geq\}$. The proposition is satisfied in a state $\alpha \in Q$ only if *all* points of $Re(\alpha)$ satisfy the constraint stated by the proposition. In such a setting it holds that $\neg(x_i \leq \theta^i_j) = x_i \geq \theta^i_j$ and vice-versa. Since the rectangles are closed sets, we cannot naturally interpret constraints expressing the relations $<, >, =, \neq$.

In the context of RATS, we usually use LTL model checking not universally to decide whether a particular property is satisfied by all paths, but rather existentially to find a path that satisfies the given property. E.g., one can explore the model in Figure 5 to check whether from the given initial condition it is possible to reach a state in which $A \leq 3$. For a single path, such a property can be expressed $\mathbf{F}(A \leq 3)$. We refer such a formula as the *observer*. When we model check a negation of the observer, the formula $\mathbf{G}(A \geq 3)$, the returned counterexample (if exists) gives us a path satisfying the observer.

The most elementary properties express *safety* properties, an example of which is, in fact, $\mathbf{G}(A \geq 3)$, stating that a state where the concentration of A is below 3 is not reachable from the given initial condition. Safety properties allow us to observe paths on which particular concentration levels are reached. Properties of this kind are not affected by the presence or absence of selfloops on particular states.

Another class of properties expresses *conditional safety*. An example of an observer for conditional safety is $(A \geq 3)\mathbf{U}(B \leq 5)$ expressing the property that until B falls below 5 the concentration of A keeps above the threshold level 3. In contrast to the previous properties, conditional safety properties provide run-time monitors which can express causality of particular events. Again in this case, the presence of a selfloop cannot violate verification of the observer. However, when interpreting the conditional safety observer formula universally, a selfloop can violate its validity.

Finally there remains a class of *liveness* properties, e.g., $\mathbf{FG}(A \leq 1)$, expressing the observation that A stabilizes within a certain level. Since liveness properties naturally require unbounded monotonic time progress, they strongly rely on the presence of selfloops at non-transient states.

IV. Comparison of the Heuristics

For the comparison of proposed heuristics we chose a small well known model of the Michaelis-Menten enzyme complex reaction. The model is based on the chemical equation $S + E \rightleftharpoons^{k_1}_{k_2} ES \rightarrow^{k_3} P + E$ where E denotes the catalyzing enzyme, S the substrate, P the product, and ES the intermediate product in which the reaction energy accumulates.

The reaction rates where set to $k_1 = k_3 = 1.0$, $k_2 = 0.01$. Initial conditions where chosen as $S = 50$, $E = 100$, $ES = P = 0$. By numeric simulation using the COPASI tool [18] the area of interest was estimated to be $E \in \langle 0.01, 108 \rangle$, $S \in \langle 0.01, 64 \rangle$, $P \in \langle 0.01, 72 \rangle$, $ES \in \langle 0.01, 16 \rangle$.

This area was partitioned into 4x4x4x3 intervals along the E, S, ES and P dimensions and further refined to partitions of 8x8x8x6 and 16x16x16x12 intervals.

With this partitioning the initial region was set to $E \in \langle 88, 96 \rangle$, $S \in \langle 32, 48 \rangle$, $ES \in \langle 0.01, 4 \rangle$, $P \in \langle 0.01, 12 \rangle$.

To evaluate the quality of each of the heuristics, two RATSs where generated as reference, a minimal and a maximal one. The minimal transition system $R_{min} = (Q_{min}, T_{min}, I_{min})$ was obtained manually (without using the state space generator) by numerically computing a dense set of 7^4 trajectories starting from the initial region with the COPASI tool and constructing a RATS from the data-points. The maximal transition system $R_{max} = (Q_{max}, T_{max}, I_{max})$ was obtained directly from our state space generator by applying the algorithm without any

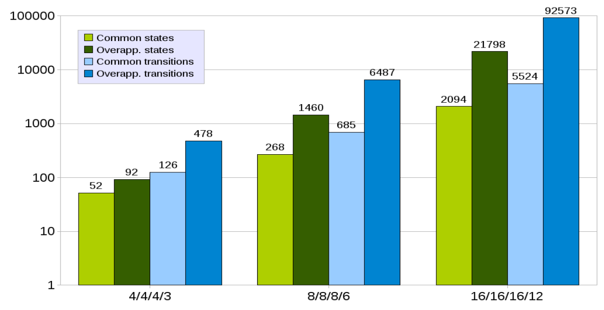

Figure 6. Exponential growth of states and transitions

abstraction tuning. Each of the heuristics h^s, h^p, and h^r was then used to construct a RATS $R_h = (Q_h, T_h, I_h)$ that was afterwords compared in terms of state and transition numbers. The minimal RATS R_{min} was used to compute the *overapproximated* and *common* sets of states and transitions as $Q_o = Q_h - Q_{min}$, $T_o = T_h - T_{min}$, $Q_c = Q_h \cap Q_{min}$, $T_c = T_h \cap T_{min}$. The maximal RATS R_{max} was then used to scale down the numbers into percentages.

To study the influence of refinement depth and parameter choice, these steps were repeated for each of the partition variants with values of $\sigma \in \{0.1, 0.2, \dots, 0.9\}$, for h^s, and $\sigma \in \{0.3, 0.6, 0.8, 0.9, 0.93, 0.96, 0.98, 0.99\}$, for h^p and h^r.

Experiment results are presented in Figures 6 to 9. Figure 6 shows how numbers of states and transitions of R_{min} and R_{max} keep on growing exponentially with further refining of the state space. In Figure 7 the scalable use of h^s is shown for parametric reduction of the overall state space, horizontal axis shows values of σ, vertical axis is the relativized number of states and transitions (relatively to R_{max}). Figure 8 further extends the results of Figure 7 by showing that with the growing amount of reduced states, the portion of underapproximated states (parts of R_{min} not included into the reduced R_h) keeps growing. The last Figure 9 shows results for h^p. As we can see this heuristics is less scalable and more conservative, the numbers of underapproximated states and transitions for this heuristics are around single percentage points (not shown in graphs) while for h^s these can reach to tens of percents.

V. Experiments on Model Checking

We have conducted several experiments on larger biological models to get the intuition to which extent the parallelization can provide a significant speed-up. Using our prototype tool we have performed analysis of an enzyme reaction modeled in terms of mass action kinetics to demonstrate how the computation scales with respect to the number of chemical substances in different settings of heuristics and threshold refinement depth. Moreover, we have performed an experiment on a real biological model (an ammonium assimilation module of E. Coli developed under the FP6 project EC-MOAN (`http://www.ec-moan.org`). The

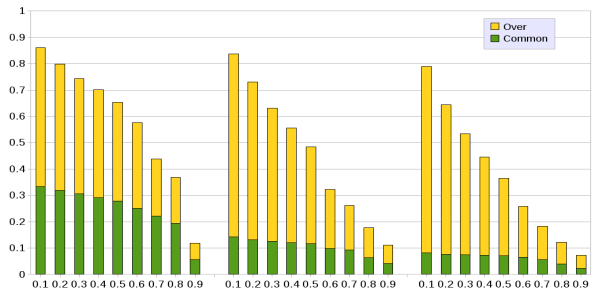

Figure 7. h^s - common and overapproximated states

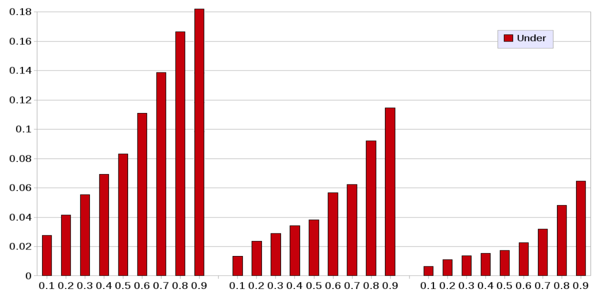

Figure 8. h^s - underapproximated states

module consists of 42 chemical substances involved in 56 reactions. All experiments have been performed on a homogeneous cluster allowing computation on up-to 22 nodes each equipped with 16GB of RAM and a quad-core processor Intel Xeon 2GHz.

A. Enzyme Kinetics Model

Let us recall the Michaelis-Menten model introduced in Section IV. The model can be refined by adding arbitrary number of intermediate products in the following way:

$$S + E \rightleftharpoons ES_1 \rightleftharpoons ES_2 \rightleftharpoons \cdots \rightleftharpoons ES_k \rightarrow P + E$$

When S and E are initially set to sufficiently high concentration levels (in our case $S = 50$ and $E = 100$)

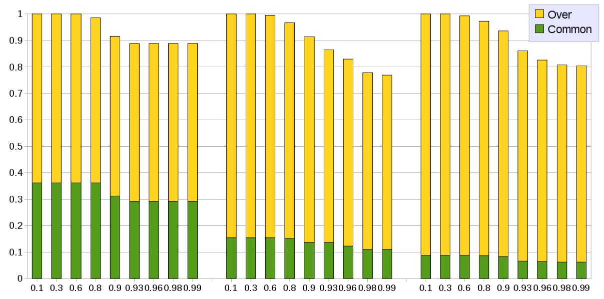

Figure 9. h^p - common and overapproximated states

Refin.	Method employed											
	full			h^s, $\sigma = 0.5$			h^p, $\sigma = 0.8$			h^r, $\sigma = 0.8$		
depth	States	Trans	Result	States	Trans	Result	States	Trans	Result	States	Trans	Result
0	179	668	spurious	124	338	spurious	175	540	spurious	176	554	spurious
1	1587	6534	spurious	468	1208	spurious	1451	4666	spurious	1546	5153	realistic
2	$9.8 \cdot 10^5$	$3.7 \cdot 10^6$	spurious	$2.3 \cdot 10^5$	$6.6 \cdot 10^5$	spurious	$9.7 \cdot 10^5$	$3.3 \cdot 10^6$	realistic	$9.5 \cdot 10^5$	$3.4 \cdot 10^6$	realistic

Table I
ENZYME OBSERVATION FOR DIFFERENT THRESHOLD REFINEMENT DEPTHS

			Time on particular number of cluster processor cores (s)																			
k	States	Trans	1	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	Result
5	$3 \cdot 10^4$	$8.5 \cdot 10^4$	15.4	4.9	2.7	1.8	1.4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	found
10	$9 \cdot 10^5$	$3.2 \cdot 10^6$	$\top$	$\top$	$\top$	161	119	107	85	72	63	55	53	46	44	40	38	38	$\perp$	$\perp$	$\perp$	found
15	$1.6 \cdot 10^6$	$6.5 \cdot 10^6$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	222	204	177	156	146	129	122	117	110	98	93	found
20	$3.2 \cdot 10^6$	$1.4 \cdot 10^7$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	202	180	173	found

Table II
ENZYME OBSERVATION SCALED FOR DIFFERENT NUMBERS OF INTERMEDIATES (USING HEURISTICS h^s, $\sigma = 0.5$)

while all the other substances are put to zero, then E and S are consumed by increasing concentration of the intermediates causing P to grow. Finally, when the whole S is consumed, E stabilizes back at the initial level. The behavior observed in the dynamics of E can be expressed by formula $\varphi_1 = E > 95 \wedge (E > 95\mathbf{U}(E =< 95 \wedge (E <= 95\mathbf{U}E > 95)))$. In the experiments, we have taken φ_1 as the observer. Note that since φ_1 is an observer for a temporal safety formula, observation of φ_1 is not distorted by the overapproximation of transiency. However, we may still observe a spurious behavior satisfying φ_1. Reduction of spurious paths can be realized by refining the threshold mesh. In Table I there are shown results for the single-intermediate model, different refinement depths, and different settings of heuristics. Refinement depth 1 had around 10 thresholds per variable whereas refinement depth 2 had around 20 thresholds per variable. The experiments have been performed on one processor. Note that numbers of states and transitions reflect the size of product automata (property automaton $\times$ model automaton). Identification of spurious paths has been realized by approximate comparison against numerical simulations. The experiments show that the effectiveness of both randomized heuristics is quite small. To this end, it can be useful to globally decrease the probability of slow reactions by multiplying it by a constant $\lambda \in (0, 1]$. We employed this approach in experiments provided in Table III.

Table II shows how the computation scales when distributed on a cluster (all experiments taken with heuristics h^s, $\sigma = 0.5$, and refinement depth 1). k denotes the number of intermediates. $\perp$ denotes the computations which did not provide better times, $\top$ stands for computations that took longer than $300s$ or ran out of memory. The largest feasibly analyzable model consisted of 23 variables. Computational limits of the full algorithm and both randomized heuristics are captured in Table III (experiments performed on 72 cluster cores, refinement depth set to 1).

σ	States	Trans	Mem	Time (s)	Result
0.75	$6 \cdot 10^4$	$4.6 \cdot 10^5$	13 GB	2.3	found
0.5	$6.2 \cdot 10^5$	$4.2 \cdot 10^6$	18 GB	9	found
0.3	$1.2 \cdot 10^7$	$1.3 \cdot 10^8$	198 GB	175	found

Table IV
E.COLI MODEL EXPERIMENTS PERFORMED WITH DIFFERENT SETTINGS OF THE HEURISTICS h^s

B. E. Coli Ammonium Assimulation Model

In the E.coli ammonium assimilation model we have employed the liveness property $\mathbf{FG}\varphi$ to observe whether a certain species stabilize at the particular concentration level. The threshold mesh was set to five thresholds per most of the species so that the initial region and the required property could be interpreted on the model. Table IV shows the best computation times we have achieved by using the heuristics h^s. We were unable to run the full algorithm, since the cluster memory has been exceeded. It is important to note that the path which has been found by the model checker can be a spurious behavior not present in the original system. To this end, it must be further analyzed. Here we face the main bottleneck of the model checking approach employed on rectangular abstractions. Because of the complexity of the model we cannot provide a suitable comparison of the returned paths against numerical simulations – as the number of the simulations required for this aim grows exponentially with the model dimension. Since we deal with a liveness property, the result has to be also analyzed on the existence of false-positive selfloops. In general, we leave for future work the development of methods which can help us to identify realistic paths among spurious ones.

VI. CONCLUSION

In this paper we have presented a prototype implementation of a tool for parallel LTL model checking of multiaffine ODE models of biological networks. We have introduced several heuristics that reduce the numbers of transitions and states needed for the analysis. The heuristics are based on the idea of considering only most significant transitions when generating successors of a given state. We have provided a set of experiments on which the effectiveness

k	Method employed								
	full			h^p, $\sigma = 0.8$, $\lambda = 0.5$			h^r, $\sigma = 0.9$, $\lambda = 0.5$		
	States	Trans	Time	States	Trans	Time	States	Trans	Time
5	$2.1 \cdot 10^6$	$1.5 \cdot 10^7$	15.7	$9 \cdot 10^5$	$3.9 \cdot 10^6$	10.2	$8.1 \cdot 10^5$	$3.6 \cdot 10^6$	10.3
10	$\top$	$\top$	$\top$	$7 \cdot 10^8$	$1.5 \cdot 10^8$	16163	$6.2 \cdot 10^8$	$1.1 \cdot 10^8$	14507

Table III
COMPUTATIONAL LIMITS OF ENZYME OBSERVATION EXPERIMENTS WITH HEURISTICS h^p, h^r

of the individual heuristics is compared against numerical simulations of the model. The most important result is the fact that in all experiments every heuristics has provided a scalable technique for reduction of the size of the reachable state space. Since the heuristics underapproximate the state spaces of the original continuous systems, we have also provided an analysis that gives us the information to which extent the underapproximation takes effect. In general, the probabilistic heuristics h^p, h^r have appeared to give very closed approximations. The heuristics h^s provides a powerful technique for reduction of the state space at the cost of more extensive underapproximations.

To show how the algorithms scale in parallel environment, we have showed several model checking experiments on large models. The results showed that when requiring a fast response on a common cluster, we are able to analyze up-to 20-variable models with around 10 thresholds per variable. When considering models with less thresholds per variable, we can satisfactorily analyze even larger models.

ACKNOWLEDGMENT

This work has been supported by the FP6 project No. NEST-043235 (EC -MOAN), by the Academy of Sciences of CR grant No. 1ET408050503, and by Czech Science Foundation grant No. 201/09/P497.

REFERENCES

[1] C. Belta and L. Habets, "Controlling a class of nonlinear systems on rectangles," *IEEE Transactions on Automatic Control*, vol. 51, no. 11, pp. 1749–1759, 2006.

[2] M. Kloetzer and C. Belta, "Reachability analysis of multi-affine systems," *Transactions of the Institute of Measurement and Control*, 2008, in press.

[3] T. Mestl, E. Plahte, and S.W.Omholt, "A mathematical framework for describing and analysing gene regulatory networks," *Journal of Theoretical Biology*, vol. 176, no. 2, pp. 291–300, 1995.

[4] 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 e. coli," in *Conference on Intelligent Systems for Molecular Biology*, 2005, pp. 19–28.

[5] G. Batt, C. Belta, and R. Weiss, "Model checking liveness properties of genetic regulatory networks," in *TACAS 2007*, ser. LNCS, vol. 4424. Springer, 2007, pp. 323–338.

[6] R. Donaldson and D. Gilbert, "A model checking approach to the parameter estimation of biochemical pathways," in *Computational Methods in Systems Biology CMSB 2008*, ser. LNBI, vol. 5307. Springer, 2008, pp. 269–287.

[7] 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*, ser. ENTCS, vol. 194, no. 3. Elsevier, 2007, pp. 35–50.

[8] G. Batt, C. Belta, and R. Weiss, "Model checking genetic regulatory networks with parameter uncertainty," in *Workshop on Hybrid Systems: Computation and Control*, ser. LNCS, vol. 4416. Springer, 2007, pp. 61–75.

[9] J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková, and D. Šafránek, "On Algorithmic Analysis of Transcriptional Regulation by LTL Model Checking," *Theoretical Computer Science*, vol. 410, pp. 3128–3148, 2009.

[10] H. de Jong, J. Geiselmann, C. Hernandez, and M. Page, "Genetic network analyzer: qualitative simulation of genetic regulatory networks," *Bioinformatics*, vol. 19, no. 3, pp. 336–344, 2003.

[11] A. Rizk, G. Batt, F. Fages, and S. Soliman, "On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology," in *Computational Methods in Systems Biology CMSB 2008*, ser. LNBI, vol. 5307. Springer, 2008, pp. 251–268.

[12] M. Feinberg, "Chemical reaction network structure and the stability of complex isothermal reactors i. the deficiency zero and the deficiency one theorems," *Chemical Engineering Science*, vol. 42, pp. 2229–2268, 1987.

[13] 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*, ser. LNCS, vol. 4144/2006. Springer, 2006, pp. 278–281.

[14] I. Černá and R. Pelánek, "Distributed explicit fair cycle detection (set based approach)," in *Model Checking Software. 10th International SPIN Workshop*, ser. LNCS, vol. 2648. Springer, 2003, pp. 49 – 73.

[15] J. Barnat, L. Brim, and J. Chaloupka, "From Distributed Memory Cycle Detection to Parallel LTL Model Checking," *Electronic Notes in Theoretical Computer Science*, vol. 133, no. 1, pp. 21–39, May 2005.

[16] B. Chazelle, "An optimal convex hull algorithm and new results on cuttings," *Symposium on Foundations of Computer Science*, vol. 0, pp. 29–38, 1991.

[17] R. Vallabhajosyula and H. Sauro, "Complexity reduction of biochemical networks," in *WSC '06: Proceedings of the 38th conference on Winter simulation*. Winter Simulation Conference, 2006, pp. 1690–1697.

[18] S. H. et.al., "Copasi – a complex pathway simulator," *Bioinformatics*, no. 22, pp. 3067–74, 2006.