Taming the Complexity of Biochemical Models through Bisimulation and Collapsing: Theory and Practice

M. Antoniotti[1], B. Mishra[1,2], C. Piazza[3], A. Policriti[4], and M. Simeoni[3]

[1] Courant Institute of Mathematical Science, NYU, New York, U.S.A.
[2] Watson School of Biological Sciences, Cold Spring Harbor, New York, U.S.A.
[3] Dipartimento di Informatica, Università *Ca' Foscari* di Venezia, Italy
[4] Dipartimento di Matematica e Informatica, Università di Udine, Italy
`marcoxa@cs.nyu.edu, mishra@nyu.edu piazza@dsi.unive.it, policrit@dimi.uniud.it simeoni@dsi.unive.it`

Abstract. Many biological systems can be modeled using systems of ordinary differential algebraic equations (e.g., S-systems), thus allowing the study of their solutions and behavior automatically with suitable software tools (e.g., PLAS, Octave/Matlab^{tm}). Usually, numerical solutions (*traces* or *trajectories*) for appropriate initial conditions are analyzed in order to infer significant properties of the biological systems under study. When several variables are involved and the traces span over a long interval of time, the analysis phase necessitates automation in a scalable and efficient manner. Earlier, we have advocated and experimented with the use of automata and temporal logics for this purpose (XS-systems and Simpathica) and here we continue our investigation more deeply.
We propose the use of *hybrid automata* and we discuss the use of the notions of *bisimulation* and *collapsing* for a "qualitative" analysis of the temporal evolution of biological systems. As compared with our previous approach, hybrid automata allow maintenance of more information about the differential equations (S-system) than standard automata. The use of the notion of bisimulation in the definition of the *projection operation* (restrictions to a subset of "interesting" variables) makes possible to work with reduced automata satisfying the same formulae as the initial ones. Finally, the notion of collapsing is introduced to move toward still simpler and equivalent automata taming the complexity of the automata whose number of states depends on the level of approximation allowed.

Keywords: Biochemical Models, Hybrid Automata, Bisimulation, Collapsing.

1 Introduction

The emerging fields of *system biology* [30], and its sister field of bioinformatics, focuses on creating a finely detailed and "mechanistic" picture of biology at the cellular level by combining the part-lists (genes, regulatory sequences, other objects from an annotated genome, and known metabolic pathways), with observations of both transcriptional states of a cell (using micro-arrays) and translational states of the cell (using proteomics tools).

Recently, the need has arisen for more and more sophisticated and mathematically well founded computational tools capable of analyzing the models that are and will be at the core of *system biology.* Such computational models should be implemented in software packages faithfully while exploiting the potential trade-offs among usability, accuracy, and scalability dealing with large amounts of data. The work described in this paper is part of a much larger project still in progress, and thus only provides a partial and evolving picture of a new paradigm for computational biology.

Consider the following scenario. A biologist is trying to test a set of hypotheses against a corpus of data produced in very different ways by several *in vitro*, *in vivo*, and *in silico* experiments. The system the biologist is considering may be a piece of a *pathway* for a given organism. The biologist can access the following pieces of information:

– raw data stored somewhere about the temporal evolution of the biological system; this data may have been previously collected by *observing* an in vivo or an in vitro system, or by *simulating* the system in silico;
– some mathematical model of the biological system[5].

The biologist will want to formulate *queries* about the evolution encoded in the data sets. For example, he/she may ask: *will the system reach a "steady state"?*, or *will an increase in the level of a certain protein activate the transcription of another?* Clearly the set of numerical *traces* of very complex systems rapidly becomes unwieldy to wade through for increasingly larger numbers of variables.

Eventually, many of these models will be available in large public databases (e.g. [7, 27–29, 39, 35]) and it is not inconceivable to foresee a biologist to test some hypotheses *in silico* before setting up expensive wet-lab experiments. The biologist will mix and match several models and raw data coming from the public databases and will produce large datasets to be analyzed.

To address this problem, we have proposed a set of theoretical and practical tools, XS-systems and Simpathica, that allow the biologist to formulate such queries in a simple way [4–6]. The computational tool Simpathica derives its expressiveness, flexibility, and power by integrating in a novel manner many commonly available tools from numerical analysis, symbolic computation, temporal logic, model-checking, and visualization. In particular, an *automaton-based* semantics of the temporal evolution of complex biochemical reactions starting from their representations as sets of differential equations is introduced. Then *propositional temporal logic* is used to qualitatively reason about the systems. When we speak of "qualitative reasoning," as in the preceding sentence, we do not intend to describe an abstracted reasoning process devoid of all quantitative information—rather, we focus on the relation among several basic properties

[5] We note that simulating a system *in silico* actually requires a mathematical model. However, we want to consider the case when such mathematical model is unavailable to both the biologist and the software system.

(each described by an atomic proposition), where each one may involve some quantitative information, e.g., "property of a protein concentration reaching half of it initial value."

In this paper we continue our research on the computational models at the core of our approach. We bring in several techniques from the fields of Verification, Logic and Control Theory, while maintaining a trade off between the need to manipulate large sets of incomplete data and the requirements arising from the needs to provide a mathematically well founded system. In particular, we propose the use of *hybrid automata* together with the notions of *bisimulation* and *collapsing*. Hybrid automata are equipped with states embodying time-flow, initial and final conditions, and therefore allow maintenance of more information about the differential equations (S-system). The use of the notion of bisimulation in the definition of the *projection operation* (restrictions to a subset of "interesting" variables) provides a way to introduce *reduced* automata satisfying the same formulae as the initial ones. Notice that the idea behind and potential of this notion of bisimulation can be exploited just as fruitfully here as in the context of standard automata. Finally, the notion of collapsing, we introduce, serves a dual purpose: first, it provides a natural approach for qualitative reasoning of the automata extracted from the analysis of traces summarizing the behavior of biomolecules; second, it tames the otherwise unruly complexity of the automata in terms of their size as a function of the levels of approximation allowed.

The cellular and biochemical processes analyzed using XS-systems and Simpathica [5, 4] provide a large set of application examples for the framework we present here. In order to motivate the choices of our modeling framework, the paper focuses on a detailed examination of one such example: namely, the repressilator system described by Elowitz and Leibler in [21]. Later, in Section 7, we study two more complex and natural systems: the purine metabolism, first, described in [38] Chapter 10 and fully analyzed in [15, 16]; the quorum sensing process in *Vibrio fischeri* which has been studied in [26, 32, 37, 1].

We conclude pointing out that the analysis presented in this paper is not limited to XS-Systems, but could be extended to more general hybrid system models.

2 Related Works

A survey on the different approaches for modeling and simulating genetic regulatory systems can be found in [18]: the author takes into consideration different mathematical methods (including ordinary and partial differential equations, qualitative differential equations and others) and evaluates their relative strengths and weaknesses.

The problem of constructing an automaton from a given mathematical model of a general dynamical system has been previously considered in the literature. In particular, it has been investigated by Brockett in [8]: our approach in [5] is certainly more focused, since it deals with specific mathematical models (i.e. S-systems). Here we move farther away from purely discrete models, and adapt hybrid automata to describe the underlying biochemical behavior instead of standard automata. Consequently, we are able to take advantage of the continuous component of hybrid automata for allowing quantitative information in addition to qualitative reasoning.

The use of hybrid automata for the modeling and simulation of biomolecular networks has been proposed also by Alur et al. in [1] and by Chabrier et al. in [10]. In [1] the discrete component of an hybrid automaton is used to switch between two different behaviors (models) of the considered biological system, (for example) depending on the concentration of the involved molecules. The hybrid automaton is then implemented in Charon. In our case, the continuous component is used to model the permanence on a given state depending on the values of the involved variables (reactants), and the discrete component is used for enabling the transition to another state. Moreover, we do not only model the biological systems, but we also query them using temporal logics. A similar approach is considered in [10], where a variant of Euler's method is applied in order to obtain a symbolic representation of the system. Then the authors show how to use symbolic model checkers, such as NuSMV [11] and DMC [19], to study the system.

Moreover, in [1], as well as in other formalisms modeling biochemical systems (e.g., [36, 14, 13, 17]), the notion of concurrency is explicitly used since the involved reactants are represented as processes running in parallel. In our case this kind of concurrency becomes implicit since in all the states of the automaton representing an S-system the values of all the reactants and their evolutions are represented.

3 Setting the Context

3.1 S-systems

We begin presenting the basic definitions and properties of S-systems. The definition of S-systems we use in this paper is basically the one presented in [38] augmented with a set of *algebraic constraints*. The constraints characterize the conditions that must be additionally satisfied for the system to obey conservation of mass, stoichiometric relations, etc.

Definition 1 (S-system). *An S-system is a quadruple $S = (DV, IV, DE, C)$ where:*

- *$DV = \{X_1, \ldots, X_n\}$ is a finite non empty set of* dependent variables *ranging over the domains $D_1, \ldots, D_n$, respectively;*
- *$IV = \{X_{n+1}, \ldots, X_{n+m}\}$ is a finite set of* independent variables *ranging over the domains $D_{n+1}, \ldots, D_{n+m}$, respectively;*
- *DE is a set of* differential equations*, one for each dependent variable, of the form*

$$\dot{X}_i = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}}$$

with $\alpha_i, \beta_i \geq 0$ *called* rate constants*;*

– C *is a set of* algebraic constraints *of the form*

$$C_j(X_1, \ldots, X_{n+m}) = \sum(\gamma_j \prod_{k=1}^{n+m} X_k^{f_{jk}}) = 0$$

with γ_j *called* rate constraints*.*

In what follows we use $\vec{X}$ to denote the vector $\langle X_1, \ldots, X_n, X_{n+1}, \ldots, X_{n+m}\rangle$ of variables and $\vec{d}$ ($\vec{a}$, $\vec{b}$,...) to denote the vector $\langle d_1, \ldots, d_n, d_{n+1}, \ldots, d_{n+m}\rangle \in D_1 \times \ldots \times D_n \times D_{n+1} \times \ldots \times D_{n+m}$ of values. Similarly given a set of variables $U = \{X_{U_1}, \ldots, X_{U_u}\} \subseteq DV \cup IV$ we use $\vec{X} \upharpoonright U$ to denote the vector of variables of U, while $\vec{d} \upharpoonright U$ denotes the vector of values $\langle d_{U_1}, \ldots, d_{U_u}\rangle \in D_{U_1} \times \ldots \times D_{U_u}$.

The dynamic behavior of an S-system can be simulated by computing the approximate values of its variables at different time instants (*traces*). To determine a trace of an S-system it is necessary to fix an initial time (t_0), the values of the variables at the initial time ($\vec{X}(t_0)$), a final time (t_f), and a step (s).

Definition 2 (Trace). *Let* $S = (DV, IV, DE, C)$ *be an S-system. Let* $\vec{f}(t) = \langle f_1(t), \ldots, f_{n+m}(t)\rangle$ *be a (approximated) solution for the S-system* S *in the time interval* $[t_0, t_f]$ *starting with initial values* $\vec{X}(t_0)$ *in* t_0*. Let* $s > 0$ *be a time step such that* $t_f = t_0 + j * s$*. The sequence of vectors of values*

$$tr(S, t_0, \vec{X}(t_0), s, t_f) = \langle \vec{f}(t_0), \vec{f}(t_0 + s), \ldots, \vec{f}(t_0 + (j-1) * s), \vec{f}(t_0 + j * s)\rangle$$

is a trace *of* S*. When we are not interested in the parameters defining the trace we use the notation* tr*.*

Notice that $\vec{f}(t_0) = \vec{X}(t_0)$. A trace is nothing but a sequence of values of $D_1 \times \ldots \times D_{n+m}$ representing a solution of the system in the time instants $t_0, t_0 + s, \ldots, t_0 + j * s$. By varying the initial values of the variables, we obtain different system traces, for the same parameters t_0, s and t_f. Notice moreover that it is not restrictive to consider traces having a fixed time step: the theory we develop can be straighforwardly adapted to variable time steps. Simulations of the behavior of an S-system can be automatically obtained by using the tool PLAS (see [38]). In fact, PLAS takes in input an S-system and approximates the values of the system variables, once the parameters in Definition 2 have been specified. The output is exactly a trace describing the behavior of the given system.

Example 1. The following feedback system is taken from [38], Chapter 6, and can be found in PLAS (see `\Book_Examples\Feedback.plc`). It represents a system in which the reactant X_1 is inhibited by X_2, while X_3 is an independent input variable and X_4 an independent inhibitor for the degradation of X_2. Hence, we have $DV = \{X_1, X_2\}$, $IV = \{X_3, X_4\}$, and

$$\dot{X}_1 = 0.5X_2^{-2}X_3^{0.5} - 2X_1 \qquad \dot{X}_2 = 2X_1 - X_2^{0.5}X_4^{-1}$$

Let $t_0 = 0$ be the initial time, $\vec{X}(t_0) = \langle 1, 1, 4, 2\rangle$ be the initial values of the reactants, $s = 1$ be the time step, and $t_f = 18$ be the final time. By simulating the system in PLAS with these values and setting the Taylor method with tolerance $1E - 16$ we obtain the following trace

$$\langle\ \langle 1, 1, 4, 2\rangle, \langle 0.33, 1.59, 4, 2\rangle, \langle 0.22, 1.48, 4, 2\rangle, \ldots$$
$$\ldots, \langle 0.28, 1.31, 4, 2\rangle, \langle 0.28, 1.31, 4, 2\rangle, \langle 0.28, 1.31, 4, 2\rangle\ \rangle$$

where, due to lack of space, we have only presented the values at low precision (two decimal places) and omitted the description of some states. In this trace, for instance, we can observe that the quantity of X_1 is 0.28 in the last tree steps.

The solutions of an S-system have some nice properties. First of all they admit all the derivatives everywhere except when they intersect one of the hyperplane $X_i = 0$, for $i = 1, \ldots, n+m$. There could be problems when $X_i = 0$ for $i \in \{1, \ldots, m+n\}$ in the case one of the exponent is, for instance, of the form 0.5. As noticed in [1], this corresponds to the fact that at reasonably high molecular concentrations, one can adopt continuum models which lend themselves to deterministic models, while at lower concentrations, the discrete molecular interactions become important and deterministic models are more difficult to obtain. However, the existence of all the derivatives implies that if at a given instant t_1 all the X_i, for $i = 1, \ldots, n+m$, are different from 0, then there exists a unique solution in an interval $[t_1, t_1 + \epsilon]$ and this solution can be extended if it still holds that all the variables are different from 0. Moreover, if two solutions $\vec{f}(t)$ and $\vec{g}(t)$, obtained with different initial values, pass both in a point $\vec{d}$, possibly at different times, i.e., there exist two instants t_1 and t_2 such that $\vec{f}(t_1) = \vec{g}(t_2) = \vec{d}$, then from those instants on they always coincide, i.e., for all $p \geq 0$, $\vec{f}(t_1 + p) = \vec{g}(t_2 + p)$. This is a consequence of the fact that the *variable* time does not explicitly occur in the differential equations. What we have just stated in mathematical terms can be restated from the biological point of view saying that if the biological system modeled by the S-system reaches a state $\vec{d}$, its evolution does not depend on the states in which the system was before reaching $\vec{d}$ (i.e., the system is *without memory*). In particular, on a set of traces this last property has the following consequence.

Proposition 1. *Let $\langle \vec{a}_0, \ldots, \vec{a}_j\rangle$ and $\langle \vec{b}_0, \ldots, \vec{b}_i\rangle$ be two traces of an S-system S obtained by using the same time step s. If there exist h and k such that $\vec{a}_h = \vec{b}_k$, then for all $r \geq 0$ it holds $\vec{a}_{h+r} = \vec{b}_{k+r}$.*

Obviously in the above proposition we are assuming that we are using the same approximation method to obtain both traces. Moreover, it can be the case that the two traces are equal. This property of sets of traces of an S-system implies what is known in the area of Model Checking as *fusion closure* (see [22]). We anticipate here that all the results we present in the rest of this paper are consequences of Proposition 1, i.e., they hold every time we deal with a set of traces satisfying it. We formalize this as follows.

Definition 3 (Convergence). *A set of traces Tr is* convergent *if for all the traces $\langle \vec{a}_0, \ldots, \vec{a}_j \rangle$ and $\langle \vec{b}_0, \ldots, \vec{b}_i \rangle$ belonging to Tr, if there exist h and k such that $\vec{a}_h = \vec{b}_k$, then for all $r \geq 0$ it holds $\vec{a}_{h+r} = \vec{b}_{k+r}$.*

Corollary 1. *If Tr is a set of traces of an S-system S obtained by using the same time step s, then Tr is convergent.*

Example 2. Let us consider again the simple feedback system described in Example 1. If we simulate it using $\vec{X}(t_0) = \langle 0.33, 1.59, 4, 2 \rangle$, i.e., $\vec{X}(t_1)$ of the trace in Example 1, we obtain
$\langle\ \langle 0.33, 1.59, 4, 2 \rangle, \langle 0.22, 1.48, 4, 2 \rangle, \ldots, \langle 0.28, 1.31, 4, 2 \rangle, \langle 0.28, 1.31, 4, 2 \rangle\ \rangle$
which is exactly the trace we had before without the first state.

3.2 XS-systems

The basic idea of XS-systems (introduced in [5]) is to associate an S-system S with a finite automaton, obtained by suitably encoding a set of traces on S. Essentially, each trace on S can be encoded into a simple automaton, where states correspond to the trace elements (i.e., the values of the system variables observed at each step), and transitions reflect the sequence structure of the trace itself (i.e., there exists a transition from a state v_i to a state v_j if they are consecutive in the trace). When more than one trace is involved in the process, coinciding elements of different traces correspond to the same state in the automaton.

Consider an S-system and a set of traces on it. The automaton derived from the system traces is defined as follows.

Definition 4 (S-system Automaton). *Let S be an S-system and Tr be a set of traces on S. An* S-system automaton *is $\mathcal{A}(S, Tr) = (V, \Delta, I, F)$, where*

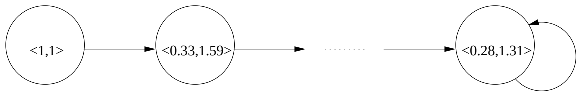
- $V = \{\vec{v} = \langle v_1, \ldots, v_{n+m} \rangle \mid \exists tr \in Tr : \vec{v} \text{ is in } tr\} \subseteq D_1 \times \ldots \times D_{n+m}$ *is the set of states;*
- $\Delta = \{(\vec{v}, \vec{w}) \mid \exists tr \in Tr : \vec{v}, \vec{w} \text{ are consecutive in } tr\}$ *is the transition relation;*
- $I = \{\vec{v} \mid \exists tr \in Tr : \vec{v} \text{ is initial in } tr\} \subseteq V$ *is the set of initial states;*
- $F = \{\vec{v} \mid \exists tr \in Tr : \vec{v} \text{ is final in } tr\} \subseteq V$ *is the set of final states.*

Automata can be equipped with labels on nodes and/or edges (see [25]). Labels on the nodes maintain information about the properties of the nodes, while labels on the edges are used to impose conditions on the action represented by the edge (see [12]). In the case of S-system automata edges are unlabeled, while the label we assign to each node is actually the name (identifier) of the node itself, i.e. the concentrations of the reactants for that state. In this way S-system automata maintain qualitative information about the system only in the instants corresponding to the steps.

We say that an automaton is *deterministic* if each node has at most one outgoing edge for each edge-label, i.e., in our case, at most one outgoing edge. From Proposition 1 we get the following result.

Proposition 2. *Let S be an S-system and Tr be a convergent set of traces on S. The automaton $\mathcal{A}(S, Tr)$ is deterministic.*

Example 3. The trace shown in Example 1 gives us the following automaton, where we omit the values of the independent variables.



The initial state is the one on the left, while final state is the one on the right. By using both the trace of Example 1 and the trace of Example 2 we obtain the same automaton, but with two initial states. The automaton represents the fact that in it, the steady state with values $X_1 = 0.28$ and $X_2 = 1.31$ is globally reachable. That is, all the simulations of this system reach this steady state independent of which initial values (equal to the values in some state of the automaton) of the reactants are assumed.

In [5], a language called ASySA (*Automata S-systems Simulation Analysis* language) has been presented to inspect and formulate queries on the simulation results of XS-systems. The aim of this language is to provide the biologists with a tool to formulate various queries against a repository of simulation traces. ASySA is essentially a *Temporal Logic* language (see [22]) (an English version of CTL) with a specialized set of predicate variables whose aim is to ease the formulation of queries on numerical quantities. The fusion closure of sets of traces (see Proposition 1 and Corollary 1) is necessary in order to reflect the behavior of the set of traces with temporal logic semantics (see [22]). This means that a formula is true on the S-system automaton if and only if it is true in the set of traces. Intuitively, the behavior of the traces is not approximated in the automaton because two traces which reach the same state always coincide in the future.

Example 4. The automaton in Example 3 satisfies the formula

$$\textsc{Eventually}(\textsc{Always}(X_2 > 1))$$

which means that the system admits a trace such that, from a certain point on, X_2 is always greater than 1. Similarly, it does not satisfies the formula

$$\textsc{Always}(\textsc{Eventually}(X_1 > X_2))$$

since it reaches a steady state in which X_1 is less than X_2.

Since the notion of steady state plays a fundamental role in biological systems, a predicate STEADY_STATE has been introduced in the ASySA language. This predicate is satisfied by a system (S-system automaton) if there exists an instant (a state) after which all the derivatives will always be equal to zero, i.e. the system ends in a loop involving only one state.

Unfortunately, in the practical cases the automata built from sets of traces have an enormous number of states. In [5] two techniques have been proposed to reduce the number of states of an S-system automaton, namely *projection* and *collapsing*.

Definition 5 (Projection). *Let S be an S-system and U be a subset of the set of variables of S. Given a trace $tr = \langle \vec{a}_0, \ldots, \vec{a}_j \rangle$ of S the* projection over U *of tr is the sequence $tr \upharpoonright U = \langle \vec{a}_0 \upharpoonright U, \ldots, \vec{a}_j \upharpoonright U \rangle$. Given a set of traces Tr the* projection over U *of Tr is the set of projected traces $Tr \upharpoonright U = \{tr \upharpoonright U \mid tr \in Tr\}$. The* U-projected S-system automaton *from Tr and S is $\mathcal{A}(S, Tr \upharpoonright U)$.*

The automaton $\mathcal{A}(S, Tr \upharpoonright U)$ has usually less states than $\mathcal{A}(S, Tr)$. However, the set of traces $Tr \upharpoonright U$ does not always satisfy either convergence or fusion closure. Furthermore, the automaton $\mathcal{A}(S, Tr \upharpoonright U)$ can be non-deterministic. This can introduce an approximation, i.e., the formulae satisfied by the automaton $\mathcal{A}(S, Tr \upharpoonright U)$ are not the same satisfied by the set of traces $Tr \upharpoonright U$.

Example 5. As a simple yet very interesting example, consider the repressilator system constructed by Elowitz and Leibler [21]. First the authors constructed a mathematical model of a network of three interacting transcriptional regulators and produced a trace of the interaction using a traditional mathematical package (Matlab$^{\mathsf{tm}}$). Subsequently, they constructed a plasmid with the three regulators and collected data from in vivo experiments in order to match them with the predicted values. In particular, this contains three proteins, namely lacl (which we refer to as X_1), tetR (X_2), and cl (X_3). The protein lacl represses the protein tetR, tetR represses cl, whereas cl represses lacl, thus completing a feedback system. The dynamics of the network depend on the transcription rates, translation rates, and decay rates. Depending on the values of these rates the system might converge to a stable limit circle or become unstable. The following S-system represents[6] the repressilator system: rate values have been set in such a way that the system converges to a stable limit circle.

$$\begin{aligned}
\dot{X}_1 &= X_4 X_3^{-1} - X_1^{0.5} \\
\dot{X}_2 &= X_5 X_1^{-1} - X_2^{0.578151} \\
\dot{X}_3 &= X_6 X_2^{-1} - X_3^{0.5}
\end{aligned}$$

If we simulate it in PLAS, with $t_0 = 0$, $\vec{X}(t_0) = \langle 0.01, 0.2, 0.01, 0.2, 0.2, 0.2 \rangle$, $s = 0.05$, and $t_f = 30$, we obtain a trace whose automaton reaches the loop shown on the left of Figure 1: we omit the independent variables and we use dotted lines to represent the fact that there are other intermediate states.

The automaton does not satisfy $\textsc{Eventually}(\textsc{Always}(X_1 \geq 0.3))$. In fact in the limit cycle reached by the repressilator, the values of X_1 range in the interval

[6] To be precise the system described in [21] is not an S-system. However, it can be reasonably approximated through an S-system, as proved by the general theory presented in [38]. Notice that our automaton-model can be built using directly traces of the system in [21].

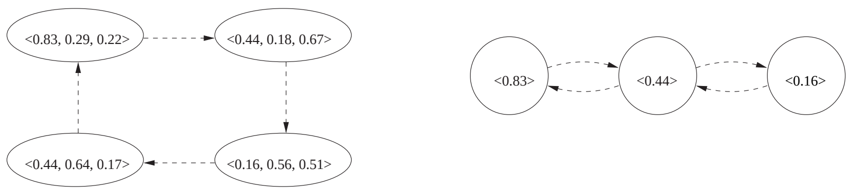


Fig. 1. Repressilator: automaton and projected automaton.

$[0.16, 0.83]$. Hence, the formula is false also in the projected trace. However, the formula is satisfied by the projected automaton, partially depicted on the right of Figure 1. In fact, the projected automaton represents a system in which it is possible that after a certain instant the variable X_1 assumes values in the interval $[0.44, 0.83]$.

The collapsing operation is defined in such a way that a state is removed from a trace when it behaves similarly to the previous one, i.e., when the derivatives computed in it can be approximated by the derivatives computed in the previous state (see [5] for the formal definition). Also this operation can introduce approximation as shown in the following example.

Example 6. Let S be an S-system with dependent variables X_1 and X_2. Let us assume that S admits a trace of the form $\langle\ \langle 1,5\rangle, \langle 2,4\rangle, \langle 3,3\rangle, \langle 4,2\rangle, \langle 5,1\rangle\ \rangle$.

We also assume that the derivative $\dot{X}_1$ is 1 in all the states except the last one, and, similarly, $\dot{X}_2$ is -1 in all the states except the last one. By applying the definitions presented in [5] we can collapse some of the states obtaining the reduced trace $\langle\ \langle 1,5\rangle, \langle 5,1\rangle\ \rangle$. The formula $\textsc{Eventually}(|X_1 - X_2| \leq 3)$ is true in the trace of S, but is false in the collapsed one.

Consider again the repressilator system of Example 5, whose automaton is partially represented on the left of Figure 1. If all the intermediate states on the dotted lines are collapsed, then we obtain an automaton with 4 states which does not satisfy the formula $\textsc{Eventually}(|X_1 - X_2| \leq 0.1)$, while it is easy to check that the same formula is satisfied by the repressilator system.

In order to avoid these approximations and to obtain a more powerful and flexible framework in the next sections we propose the use of hybrid automata together with a reformulation of projection and collapsing.

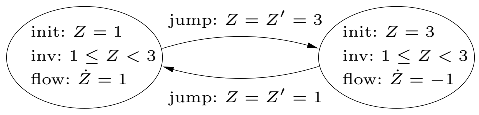
4 Hybrid Automata to model S-systems

The notion of hybrid automata was first introduced in [2] as a model and specification language for hybrid systems, i.e., systems consisting of a discrete-valued program (with finitely many modes) within a continuously changing environment.

Definition 6 (Hybrid automata). *A* hybrid automaton $H = (Z, V, \mathit{\Delta}, I, F,$ $\mathit{init}, \mathit{inv}, \mathit{flow}, \mathit{jump})$ *consists of the following components:*

– $Z = \{Z_1, \ldots, Z_k\}$ *a finite set of variables;* $\dot{Z} = \{\dot{Z}_1, \ldots, \dot{Z}_k\}$ *denotes the first derivatives during continuous change;* $Z' = \{Z'_1, \ldots, Z'_k\}$ *denotes the values at the end of discrete change;*
– (V, Δ, I, F) *is an automaton; the nodes of* V *are called* control modes*, the edges of* Δ *are called* control switches*;*
– *each* $v \in V$ *is labeled by* $init(v)$*,* $inv(v)$*, and* $flow(v)$*; the labels* $init(v)$ *and* $inv(v)$ *are constraints with free variables in* Z*; the label* $flow(v)$ *is a constraint with free variables in* $Z \cup \dot{Z}$*;*
– *each* $e \in E$ *is labeled by* $jump(e)$*, which is a constraint whose free variables are in* $Z \cup Z'$*.*

Example 7. Consider the following simple hybrid automaton.



The initial state is the left one, with $Z = 1$. In this state Z grows with constant rate 1. After 3 instants we have $Z = 3$ and we jump on the right state. In this second state Z decreases and when Z becomes 1 we jump again in the state on the left.

The usefulness of hybrid automata has been widely proved in the area of verification (see, e.g., [33]). In order to exploit the expressive power of hybrid automata their properties have been deeply studied (see [23]), specification languages have been introduced to describe them, and model checkers have been developed to automatically verify temporal logic properties on them. Among the specification languages and the model checkers which deal with hybrid automata we mention SHIFT (see [3]) and HyTech (see [24]) developed at Berkeley University, and Charon (see [1]) developed at the University of Pennsylvania.

In the S-system automata introduced in the previous section the only quantitative information maintained is the values of the variables in the instants corresponding to the steps. The values at instants between two steps are lost. This situation becomes particularly dangerous when we apply a reduction operation such as collapsing. The novelty of our approach is in the way it circumvents this problem by using the continuous component of hybrid automata to maintain also some approximate information about the values of the variables between two steps.

Let us introduce some notations which simplify the definition of a hybrid automaton modeling a convergent set *Tr* of traces of an S-system. Given the vectors $\vec{X} = \langle X_1, \ldots, X_{n+m} \rangle$ and $\vec{v} = \langle v_1, \ldots, v_{n+m} \rangle$ we use the notation $\vec{X} = \vec{v}$ to denote the conjunction $X_1 = v_1 \wedge \ldots \wedge X_{n+m} = v_{n+m}$. The notation $\vec{v} \le \vec{X} < \vec{w}$ has a similar meaning, while $\dot{\vec{X}} = (\vec{w} - \vec{v})/s$ stands for $\dot{X}_1 = (w_1 - v_1)/s \wedge \ldots \wedge \dot{X}_{n+m} = (w_{n+m} - v_{n+m})/s$.

Definition 7 (S-system Hybrid Automaton). *Let* S *be an S-system and Tr be a convergent set of traces on* S*. Consider the S-system automaton* $\mathcal{A}(S, Tr)$*.*

The S-system hybrid automaton *built on* $\mathcal{A}(S, Tr)$ *is* $\mathcal{H}(S, Tr) = (X, V, \Delta, I, F, init, inv, flow, jump)$*, where:*

- $X = \{X_1, \ldots, X_{n+m}\} = DV \cup IV$*;*
- (V, Δ, I, F) *is the automaton* $\mathcal{A}(S, Tr)$*;*
- *for each* $\vec{v} \in V$ *let* $init(\vec{v}) = \vec{X} = \vec{v}$*;*
- *for each* $\vec{v} \in V$ *such that* $(\vec{v}, \vec{w}) \in \Delta$ *let*[7] $inv(\vec{v}) = \vec{v} \leq \vec{X} < \vec{w}$*;*
- *for each* $\vec{v} \in V$ *such that* $(\vec{v}, \vec{w}) \in \Delta$ *let* $flow(\vec{v}) = \dot{\vec{X}} = (\vec{w} - \vec{v})/s$*;*
- *for each* $(\vec{v}, \vec{w}) \in \Delta$ *let* $jump((\vec{v}, \vec{w})) = \vec{X} = \vec{X}' = \vec{w}$*.*

Notice from the above definition that being in a state $\vec{v}$ does not necessarily mean that the values of the variables are exactly $\vec{v}$: they can in fact assume values between $\vec{v}$ and $\vec{w}$. In particular, they grow linearly in this interval and when they reach $\vec{w}$ the system jumps to a new state.

The automaton $\mathcal{H}(S, Tr)$ is a rectangular singular automaton and the temporal logic CTL is decidable for this class of automata (see [23]). The model checker HyTech can be used to check whether a temporal formula is satisfied by $\mathcal{H}(S, Tr)$. Moreover, $\mathcal{H}(S, Tr)$ is deterministic, since we require Tr to be convergent and hence $\mathcal{A}(S, Tr)$ is deterministic. Notice also that all the information needed to build $\mathcal{H}(S, Tr)$ is already encoded in $\mathcal{A}(S, Tr)$, i.e., it is possible to work on $\mathcal{H}(S, Tr)$ by only maintaining in memory $\mathcal{A}(S, Tr)$.

Example 8. From the traces of the feedback system of Examples 1 and 2 we obtain the hybrid automaton shown in Figure 8. In the first state (the one on

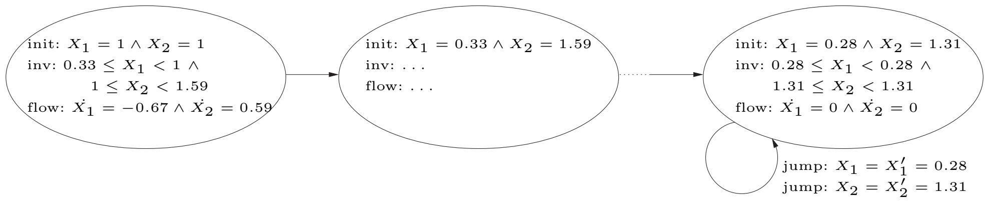

init: $X_1 = 1 \wedge X_2 = 1$
inv: $0.33 \leq X_1 < 1 \wedge$
$1 \leq X_2 < 1.59$
flow: $\dot{X_1} = -0.67 \wedge \dot{X_2} = 0.59$
init: $X_1 = 0.33 \wedge X_2 = 1.59$
inv: . . .
flow: . . .
init: $X_1 = 0.28 \wedge X_2 = 1.31$
inv: $0.28 \leq X_1 < 0.28 \wedge$
$1.31 \leq X_2 < 1.31$
flow: $\dot{X_1} = 0 \wedge \dot{X_2} = 0$
jump: $X_1 = X_1' = 0.28$
jump: $X_2 = X_2' = 1.31$

Fig. 2. Feedback hybrid automaton.

the left) variable X_1 starts with value 1 and decreases until it reaches value 0.33, while variable X_2 starts with value 1 and grows until it reaches value 0.59. Then, we jump to the second state. When we reach the last state the values of the variables become stable and the system loops forever.

The additional quantitative information stored in each state of an S-system hybrid automaton allows one to deeply investigate the behavior of the system during any individual step. As we will see in Section 6, this process assumes an additional relevance when we apply a collapsing technique to reduce the number of states.

[7] We invert the interval when $w_i < v_i$.

5 Bisimulation and Projection

As pointed out in Example 5 the projection operation can lead to incorrect prediction since we only use a reduced automaton. In order to avoid this problem, we define in this section a projection operator based on the notion of bisimulation. Since bisimulation is an equivalence relation preserving temporal logic formulae (see, e.g., [9, 31]), the obtained projected automata will satisfy the same formulae as the original one.

Let us introduce the following notations. Given a condition $\mathit{init}(\vec{v})$ ($\mathit{inv}(\vec{v})$, $\mathit{flow}(\vec{v})$, resp.) and $U \subseteq DV \cup IV$ we use $\mathit{init}(\vec{v}) \restriction U$ ($\mathit{inv}(v) \restriction U$, $\mathit{flow}(v) \restriction U$, resp.) to denote that we consider only the conditions relative to the variables in U.

Definition 8 (U-bisimulation). *Let $\mathcal{H}(S, Tr)$ be an S-system hybrid automaton. Let $U \subseteq \{X_1, \ldots, X_{n+m}\}$ be a subset of variables. A relation $R \subseteq V \times V$ is a U-*bisimulation *if*

- *if $\vec{v}R\vec{w}$, then $\mathit{init}(\vec{v}) \restriction U = \mathit{init}(\vec{w}) \restriction U \wedge \mathit{inv}(\vec{v}) \restriction U = \mathit{inv}(\vec{w}) \restriction U \wedge \mathit{flow}(\vec{v}) \restriction U = \mathit{flow}(\vec{w}) \restriction U$;*
- *if $\vec{v}R\vec{w}$ and $(\vec{v}, \vec{v}') \in \Delta$, then $(\vec{w}, \vec{w}') \in \Delta$ and $\vec{v}'R\vec{w}'$;*
- *if $\vec{v}R\vec{w}$ and $(\vec{w}, \vec{w}') \in \Delta$, then $(\vec{v}, \vec{v}') \in \Delta$ and $\vec{v}'R\vec{w}'$.*

Intuitively, two states $\vec{v}$ and $\vec{w}$ are U-bisimilar if it is the case that not only do the variables in U have the same values in $\vec{v}$ and $\vec{w}$, but additionally, from $\vec{v}$ and $\vec{w}$, the system evolves in the same way with respect to the variables in U. In fact, for instance,it is possible that there are two states in which the variables in U have the same values, but the first state evolves into a state in which the variables are incremented while the second one evolves into a state in which the variables are decremented; in this case, we do not wish to identify these two states as equivalent.

Lemma 1. *There always exists a unique maximum U-bisimulation $\approx_U$ which is an equivalence relation. Moreover, if $\vec{v} \approx_U \vec{w}$ and $(\vec{v}, \vec{v}') \in \Delta$, then $(\vec{w}, \vec{w}') \in \Delta$ and $\mathit{jump}((\vec{v}, \vec{v}')) \restriction U = \mathit{jump}((\vec{w}, \vec{w}')) \restriction U$.*

Proof. The first part follows immediately from the fact that a U-bisimulation on $\mathcal{H}(S, Tr)$ is nothing but a strong bisimulation on $\mathcal{A}(S, Tr)$ whose nodes have been labeled using part of the conditions defining the hybrid automaton $\mathcal{H}(S, Tr)$.

The second fact is a consequence of the fact that *jump* is uniquely defined once we know *init* and *inv*. □

Definition 9 (Projected Hybrid automaton $\mathcal{H}(S, Tr, U)$). *Let $\mathcal{H}(S, Tr) = (X, V, \Delta, I, F, \mathit{init}, \mathit{inv}, \mathit{flow}, \mathit{jump})$, be an S-system hybrid automaton and U be a subset of variables. The* projected hybrid automaton *$\mathcal{H}(S, Tr, U) = (U, V_U, \Delta_U, I_U, F_U, \mathit{init}_U, \mathit{inv}_U, \mathit{flow}_U, \mathit{jump}_U)$ is defined as follows:*

- *$V_U = V/\approx_U$;*
- *$\Delta_U = \{([\vec{v}], [\vec{w}]) \mid \exists \vec{v}' \in [\vec{v}], \vec{w}' \in [\vec{w}] : (v, w) \in \Delta\}$;*

– *for each* $[\vec{v}] \in V_U$ *let* $init_U([\vec{v}]) = init(\vec{v}) \restriction U$*;*
– *for each* $[\vec{v}] \in V_U$ *let* $inv_U([\vec{v}]) = inv(\vec{v}) \restriction U$*;*
– *for each* $[\vec{v}] \in V_U$ *let* $flow_U([\vec{v}]) = flow(\vec{v}) \restriction U$*;*
– *for each* $([\vec{v}], [\vec{w}]) \in \Delta_U$ *such that* $(\vec{v}', \vec{w}') \in \Delta$ *let* $jump_U(([\vec{v}], [\vec{w}])) = jump((v', w')) \restriction U$*.*

The above definition does not depend on the representative element of each class. This is a consequence of the definition of $\approx_U$ as far as the *init*, *inv*, and *flow* conditions are concerned, and of Lemma 1 as far as the *jump* conditions are concerned. Those familiar with automata and bisimulation reductions will immediately recognize that the hybrid automaton $\mathcal{H}(S, Tr, U)$ is nothing but the hybrid automaton built on the bisimulation reduced automaton $\mathcal{A}(S, Tr)/\approx_U$ with conditions defined only on the variables of U.

The automaton $\mathcal{H}(S, Tr, U)$ is still a rectangular singular automaton, hence CTL is still decidable on it. Moreover, $\mathcal{H}(S, Tr, U)$ is deterministic, since bisimulation preserves determinism. The fact that we are working on deterministic automata implies that the bisimulation relation $\approx_U$ can be computed in linear (see [20]) time using the procedure defined in [34].

As far as the correctness of the reduction is involved, we have the following result.

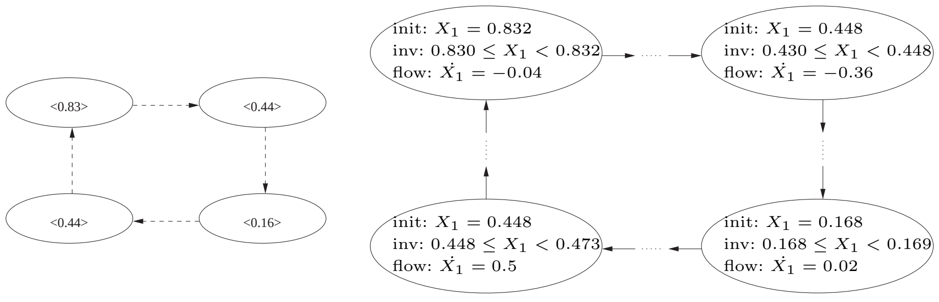
Proposition 3. *Let TL be a temporal logic which is a fragment of the μ-calculus. Let φ a formula of TL involving only the variables in U. $\mathcal{A}(S, Tr)$ satisfies φ if and only if $\mathcal{A}(S, Tr)/\approx_U$ satisfies φ. $\mathcal{H}(S, Tr)$ satisfies φ if and only if $\mathcal{H}(S, Tr, U)$ satisfies φ.*

Proof. The first part is a consequence of the fact that $\approx_U$ is a strong bisimulation and strong bisimulations preserve all the formulae of the μ-calculus (see [9, 31]).

The second part is a consequence of the first part and of the fact that $\mathcal{H}(S, Tr, U)$ is basically the hybrid automaton built on $\mathcal{A}(S, Tr)/\approx_U$. □

In the following example we show the difference between $\mathcal{A}(S, Tr \restriction U)$ and $\mathcal{A}(S, Tr)/\approx_U$. This difference is at the basis of the correctness of $\mathcal{H}(S, Tr, U)$.

Example 9. Consider again the repressilator system of Example 5. Part of the projected automaton we obtain by applying bisimulation is shown on the left of Figure 3. The two states in which $X_1 = 0.44$ do not coincide when we use bisimulation. In fact, the first state in which X_1 is 0.44 evolves to a state in which X_1 is 0.43 (the protein concentration is decreasing), while the second state in which X_1 is 0.44 evolves to a state in which X_1 is 0.47 (the protein concentration is increasing). Hence, the projected automaton fails to satisfy the formula Eventually(Always($X_1 \geq 0.3$)). This conclusion is correct, since the repressilator system we have simulated reaches a steady loop in which X_1 oscillates between 0.16 and 0.83. Part of the projected hybrid automaton is shown on the right of Figure 3.



<0.83>
<0.44>
<0.44>
<0.16>
init: $X_1 = 0.832$
inv: $0.830 \leq X_1 < 0.832$
flow: $\dot{X_1} = -0.04$
init: $X_1 = 0.448$
inv: $0.430 \leq X_1 < 0.448$
flow: $\dot{X_1} = -0.36$
init: $X_1 = 0.448$
inv: $0.448 \leq X_1 < 0.473$
flow: $\dot{X_1} = 0.5$
init: $X_1 = 0.168$
inv: $0.168 \leq X_1 < 0.169$
flow: $\dot{X_1} = 0.02$

Fig. 3. Repressilator: bisimulation quotiented automata.

6 Collapsing States

In this section we introduce the definition of collapsing of a trace. The definition we present is similar but not equivalent to the one given in [5]. In fact, we do not consider the difference between the derivatives calculated in the states, but only the degree of growth within a step. This reformulation was inspired by hybrid automaton in which in the *flow* condition of a state we do not use the derivatives calculated at the beginning of a time step, but the coefficients of the lines connecting the values at the beginning to the ones at the end of a time step. In the following collapsing definition we use a compact notation similar to the one already introduced in Section 4.

Definition 10 (Collapsing). *Let $\vec{\delta} = \langle \delta_1, \ldots, \delta_{n+m} \rangle$ be a $(n+m)$-vector of values. Let $tr = \langle \vec{a}_0, \ldots, \vec{a}_j \rangle$ be a trace obtained by simulating the S-system S with time step s. A $\vec{\delta}$-collapsing of tr is a partition of the states of tr such that:*

- *the blocks are sub-traces of tr;*
- *if a block is formed by the states from $\vec{a}_i$ to $\vec{a}_{i+h}$, and $\vec{a}_j, \vec{a}_{j+1}$ belong to the block, then $|(\vec{a}_{j+1} - \vec{a}_j)/s - (\vec{a}_{i+1} - \vec{a}_i)/s| \leq \vec{\delta}$.*

The collapsing operation in [5] is based on the difference between the first derivatives computed in the elements of the trace. Here, instead, we consider as collapsing criterion the degree of growth within a step. In practice the definition requires that the lines connecting $\vec{a}_i$ to $\vec{a}_{i+1}$ are good approximations of the lines connecting $\vec{a}_j$ to $\vec{a}_{j+1}$. As a consequence we obtain that the lines connecting $\vec{a}_i$ to $\vec{a}_{i+h}$ are good approximations of all the small lines. In particular, the following result holds.

Lemma 2. *If a block of a $\vec{\delta}$-collapsing is formed by the sequence of states from $\vec{a}_i$ to $\vec{a}_{i+h}$ and $\vec{a}_j, \vec{a}_{j+1}$ belong to the block, then $|(\vec{a}_{j+1} - \vec{a}_j)/s - (\vec{a}_{i+h} - \vec{a}_i)/(h*s)| \leq 2 * \vec{\delta}$.*

Proof. It is not restrictive to prove that the result holds on the first component. Let $(a_{i+r+1,1} - a_{i+r,1})/s = coef_{r+1}$ for $r = 0, \ldots, h-1$. It is easy to prove that $(a_{i+h,1} - a_{i,1})/(h*s) = (1/h) * ((a_{i+h,1} - a_{i+h-1,1})/s + (a_{i+h-1,1} - a_{i+h-2,1})/s +$

$\ldots + (a_{i+1,1} - a_{i,1})/s) = (1/h) * \sum_{r=0}^{h-1} coef_{r+1}$. By hypothesis we have $coef_1 - \delta_1 \leq coef_{r+1} \leq coef_1 + \delta_1$, hence we obtain $(1/h) * h * (coef_1 - \delta_1) \leq (a_{i+h,1} - a_{i,1})/(h * s) \leq (1/h) * h * (coef_1 + \delta_1)$, i.e. $coef_1 - \delta_1 \leq (a_{i+h,1} - a_{i,1})/(h * s) \leq coef_1 + \delta_1$. From this last observation, we get $(coef_1 - \delta_1) - (coef_1 + \delta_1) \leq coef_{r+1} - (a_{i+h,1} - a_{i,1})/(h * s) \leq (coef_1 + \delta_1) - (coef_1 - \delta_1)$, i.e. $-2 * \delta_1 \leq coef_{r+1} - (a_{i+h,1} - a_{i,1})/(h * s) \leq 2 * \delta_1$, which is equivalent to our thesis. $\square$

Given the trace tr and the vector $\vec{\delta}$ the partition in which each state constitutes a singleton class is a $\vec{\delta}$-collapsing.

Definition 11 (Maximal Collapsing). *Let C_1 and C_2 be two $\vec{\delta}$-collapsing of tr. We say that C_1 is* coarser *than C_2 if each block of C_2 is included in a block of C_1. We say that the $\vec{\delta}$-collapsing C_1 is* maximal *if there does not exist another $\vec{\delta}$-collapsing coarser than C_1.*

The uniqueness of a coarsest $\vec{\delta}$-collapsing is not guaranteed. However, we can give an algorithm to find a maximal $\vec{\delta}$-collapsing. The algorithm performs the following steps: it starts from $\vec{a}_0$, it check if $\vec{a}_1$ can be collapsed with $\vec{a}_0$, if this is the case it goes on with $\vec{a}_2$, and so on. Assume that $\vec{a}_i$ is the first state which does not collapse to the same state as $\vec{a}_0$, then the algorithm starts another block from $\vec{a}_i$ and it goes on in the same way.

The following proposition states that if we use maximal δ-collapsing, then two traces which match in one state always match in the future.

Proposition 4. *Let Tr be a convergent set of traces of an S-system S. Let $Tr/\vec{\delta}$ be the set of collapsed traces obtained by applying to each trace of Tr a maximal $\vec{\delta}$-collapsing. The set $Tr/\vec{\delta}$ is convergent.*

This property is sufficient to guarantee that taking a set of traces and collapsing them using maximal δ-collapsing, the set of collapsed traces can be used to build automata and hybrid automata as defined in the previous sections. In fact, as pointed out earlier in the paper, the correctness of our framework holds whenever we use convergent sets of traces. Nonetheless, this statement does not imply that the automaton we build from a set of collapsed traces satisfies the same formulae as the original set of traces, but only that it satisfies the same formulae as the set of collapsed automata derived from the traces.

Example 10. Consider again the S-system and the trace of Example 6. The collapsed trace we obtain is again $\langle\ \langle 1, 5\rangle, \langle 5, 1\rangle\ \rangle$. The hybrid automaton we build from this trace has two states. In the first state, let us call it $\vec{v}$, we have the conditions

$$inv(\vec{v}) = 1 \leq X_1 \leq 5 \wedge 1 \leq X_2 \leq 5 \qquad flow(\vec{v}) = \dot{X}_1 = 1 \wedge \dot{X}_2 = -1$$

which make $\textsc{Eventually}(|X_1 - X_2| \leq 3)$ true in the automaton, as it was in the original trace.

Similarly, we can safely collapse the states of the repressilator system and obtain a hybrid automaton with four states which correctly satisfies the formula $\textsc{Eventually}(|X_1 - X_2| \leq 0.1)$.

This last example shows that the additional information maintained in the hybrid automaton is particularly useful when we use techniques as collapsing to reduce the number of states.

7 Case Studies

In this section we present two case studies for our framework. The first one concerns the purine metabolism pathway, whose complexity makes it a good candidate for reasoning with the computational tools we have developed. The second one is the quorum sensing process in *Vibrio fischeri*, which allows us to discuss an extension of our framework admitting a system description based on more than one S-system.

7.1 Purine Metabolism

We now revisit the example of purine metabolism described in [38] Chapter 10 and fully analyzed in [15, 16]. The pathway for purine metabolism is presented in Figure 4. A brief description of the key reactions follows, and the reader is invited to examine the more detailed summaries contained in the literature referenced in [38, 15, 16].

The main metabolite in purine biosynthesis is PRPP (*5-phosphoribosyl-α-1-pyrophosphate*). A linear cascade of reactions converts PRPP into IMP (*inosine monophosphate*). IMP is the central branch point of the purine metabolism pathway. IMP is transformed into AMP and GMP. Guanosine, adenosine and their derivatives are recycled (unless used elsewhere) into HX (*hypoxanthine*) and XA (*xanthine*). XA is finally oxidized into UA (*uric acid*). In addition to these processes, there appear to be two "salvage" pathways that serve to maintain IMP level and thus of *adenosine* and *guanosine* levels as well. In these pathways, APRT (*adenine phosphoribosyltransferase*) and HGPRT (*hypoxanthine-guanine phosphoribosyltransferase*) combine with PRPP to form ribonucleotides.

The consequences of a malfunctioning purine metabolism pathway are severe and can lead to death. The entire pathway is quite complex and contains several feedback loops, cross-activations and reversible reactions, and thus an ideal candidate for reasoning with the computational tools we have developed.

In [38], a sequence of models for purine metabolism is presented alongside an analysis of how to identify discrepancies with physically observed data, and how to amend the current model in order to explain these discrepancies.

We show how to formulate queries over the simulation traces to express various desirable properties (or absence of undesirable ones) that the model should possess. Should any of these queries "fail", the model will be marked for further examination, experimentation and correction.

Consider the "Final" model for purine metabolism presented in [38]. The in silico experiment shows that when an initial level of PRPP is increased by 50-fold, the steady state concentration is quickly absorbed by the system. The level of PRPP returns rather quickly to the expected steady state values. IMP

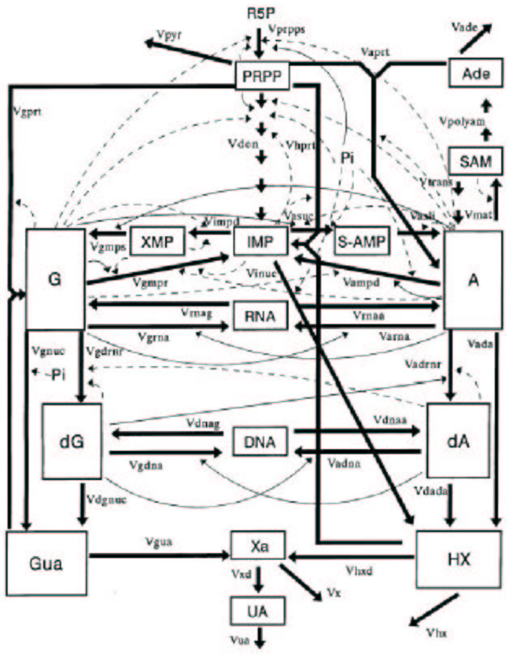


Fig. 4. The metabolic scheme of purine metabolism in human. (Reprinted from [15], where a full description and further references can be found.)

concentration level also rises and HX level falls before returning to predicted steady state values. To prove that the "Final" model in [38] correctly shows this behavior we proceed in the following way. First we simulate the system in normal conditions, with the initial values given in [38], using Sympathica. In this way we obtain the concentrations PRPP1, IMP1, HX1, ... of the reactants in the steady state. In particular, we have that PRPP1= 4.98, IMP1= 100.18, and HX1= 10.11. Then we ask Sympathica to simulate the system under the following conditions:

– initial concentration of PRPP equal to 50∗PRPP1;
– initial concentrations of all other reactants equal to the concentrations in the steady state;
– steps of one second;
– final time 5000 seconds.

Hence, we use Simpathica to formulate the following query:

$$\begin{array}{l}\textsc{steady\_state}()\ \textsc{and}\\ \textsc{Eventually}(\mathrm{IMP} > \mathrm{IMP1})\ \textsc{and}\ \textsc{Eventually}(\mathrm{HX} < \mathrm{HX1})\ \textsc{and}\\ \textsc{Eventually}(\textsc{Always}(\mathrm{IMP} = \mathrm{IMP1}))\ \textsc{and}\\ \textsc{Eventually}(\textsc{Always}(\mathrm{HX} = \mathrm{HX1}))\end{array}$$

In particular the trace we obtain, with respect to PRPP, IMP and HX, is of the form:

$$\langle\ \langle 249, 100.18, 10.11\rangle, \langle 8.95, 129.35, 2.13\rangle, \ldots, \langle 4.98, 100.18, 10.11\rangle\ \rangle$$

Applying the collapsing with $\vec{\delta} = \langle 1, 1, 1\rangle$ we obtain an (hybrid) automaton with 7 states which correctly satisfies the formula. In this case we obtain the correct answer both using the standard and the hybrid automaton.

Let us now concentrate our attention on HX and consider only the part of the previous query relative to this reactant, i.e.

$$\textsc{Eventually}(\mathrm{HX} < \mathrm{HX1})\ \textsc{and}\ \textsc{Eventually}(\textsc{Always}(\mathrm{HX} = \mathrm{HX1})).$$

Clearly, this formula is true in the trace. In fact the trace, with respect to HX, has the following form

$$\langle 10.11, 2.13, 4.98, \ldots, 10.11, \ldots, 10.11\rangle$$

By applying the projection operation we conclude that the formula is false, since we obtain a loop between the first and the last state. Instead, by using bisimulation we correctly demonstrate that the formula is true.

7.2 Quorum Sensing in *Vibrio fischeri*

In this section we present an extension of our framework which allows a system to be described by more than one S-system. The aim is to be able to capture and reason about more complicated systems classically modeled by hybrid automata. The extension has not yet been implemented in our tool set, since it requires an automata composition operation which needs to be further investigated.

Hybrid automata are natural formal models for mixed discrete-continuous systems. Typical examples are systems showing different continuous behaviors according to specific discrete values of some of the involved variables. Each state of a hybrid automaton usually models one continuous behavior (through the set of differential equations specified in the flow condition), and each state transition models the triggering mechanism (through the jump condition) allowing the changing of the continuous model.

A good example of mixed discrete-continuous biological system is the quorum sensing process in *Vibrio fischeri* (see [26, 32, 37]). Cell-density dependent gene expression in prokaryotes is a process where a single cell is able to sense when a *quorum* (i.e., a minimum population unit) of bacteria is achieved and correspondingly exibits a certain behavior. This type of cell-to-cell signaling is called *quorum sensing*, and the bioluminescence phenomenon in *Vibrio fischeri* is an example of this kind of process.

Vibrio fischeri is a marine bacterium that can be found both as a free living organism and as a symbiont of some marine fish and squid. As a free living organism, it exists at low density and is non-luminescent while, as a symbiont, it lives at high densities and is luminiscent. The accumulation of an activator molecule or *autoinducer* is responsible for triggering the production of light. The bacteria are able to sense the cell density by detecting not only the presence but also the concentration of the autoinducer. Hence, a natural way to model such different behavior of cells is to use a hybrid automaton where each state (mode) represents a specific behavior of the cell and the switching from one state to another is ruled by the degree of concentration of the autoinducer.

Before introducing a mathematical model for *Vibrio fischeri*, we describe the details of the luminescence phenomenon, which is controlled by the transcription of the *lux* genes. Figure 5 shows the *lux* region, which is organized in two trascriptional units (operons):

– the O_L operon contains the *luxR* gene which encodes the protein LuxR, a transcriptional activator of the system;
– the O_R operon contains the seven genes *lucICDABEG*. The transcription of *luxI* produces the protein LuxI required for the endogenous production of the autoinducer A_i. The genes *luxA* and *luxB* code for the luciferase subunits. The genes *luxC*, *luxD* and *luxE* code for proteins of the fatty acid reductase, needed as aldehyde substrate for luciferase. The gene *luxG* encodes a flavin reductase. Along with LuxR and LuxI, the cAMP receptor protein (CRP) plays an important role in controlling luminescence.

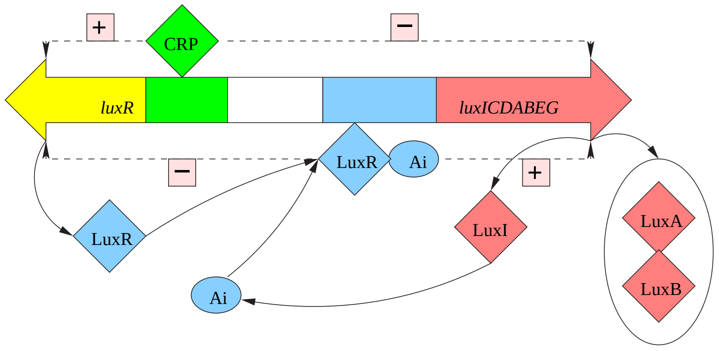


Fig. 5. The *lux* region of *Vibrio fischeri*.

The biochemical network of reactions in the cell is shown in Figure 6 and works as follows: the autoinducer A_i binds to protein LuxR to form a complex C_O

which binds to the *lux box*. The *lux box* is between the two transcriptional units and contains a binding site for CRP. The transcription from the *luxR* promoter is activated by the binding of CRP to its binding site, and the transcription of the *luxICDABEG* by the binding of C_O to the *lux box*. However, growth in the levels of C_O and cAMP/CRP inhibit *luxR* and *luxICDABEG* transcription, respectively.

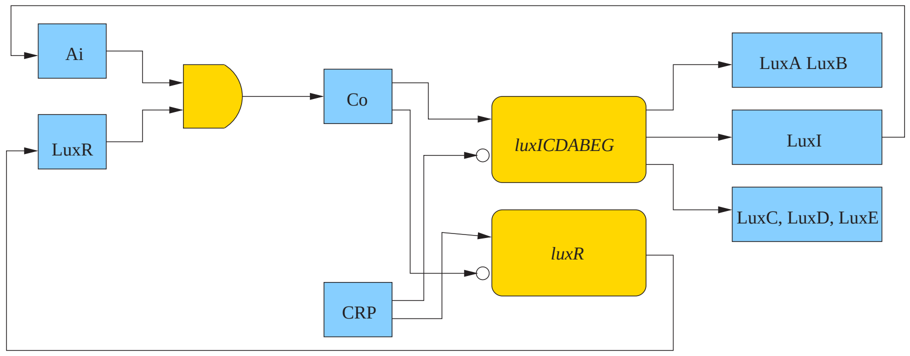


Fig. 6. The biochemical network of quorum sensing in *Vibrio fischeri*.

A mathematical model of the quorum sensing in *Vibrio fischeri* has been proposed by Alur et. al. in [1]. The model is an hybrid automaton composed of three different states (i.e., three systems of differential equations) corresponding to the modes *OFF*, *POS* and *NEG*. The switching from one mode to another is ruled by the degree of concentration of the autoinducer A_i. More precisely, the mode *OFF* corresponds to very low concentration of A_i (i.e., $A_i < A_{i\_}$) within the bacterium and no luminescence; the mode *POS* (positive growth) corresponds to increasing concentration of A_i (i.e., $A_{i\_} <= A_i <= A_{i_+}$) and luminescence; the mode *NEG* (negative growth) corresponds to high concentration of A_i (i.e., $A_i > A_{i_+}$). For a complete description of the model we refer to [1]. The three systems of differential equations associated with each mode of the Alur's model are not S-systems. However, it is possible to translate such systems into three equivalent S-systems equipped with linear constraints.

At the moment a biologist can use Simpathica to simulate and query one S-system per time. When more than one S-system is involved, he/she could define, by looking at the behavior of each system, which are the conditions under which the system reflect a real behavior, and which are the triggering conditions for combining the various systems. The idea is then to automatically compose the systems on the basis of the defined conditions. Essentially, the composition operation should first glue together the states of the different automata and then eliminate the states that, according with the specified conditions, do not reflect a real biological behavior.

Coming back to the *Vibrio fischeri* example, by simulating the obtained S-systems separately, it is possible to build the three corresponding hybrid automata, which could be then combined with respect to the degree of concentration of the A_i autoinducer to obtain the final hybrid model. Figure 7 illustrates how the three automata should be combined. Clearly the depicted automata do not really reflect the real system behavior.

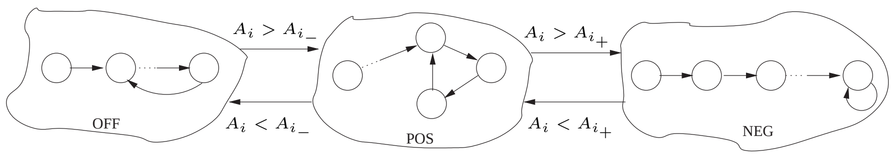


Fig. 7. Strcture of the *Vibrio fischeri* final model.

8 Conclusions

In this paper we have described how hybrid automata can be used to model and analyze set of traces representing the behavior of a biological system. Automata give a qualitative view of a set of traces by abstracting from the time instants, and thus allowing a compact representation in which the properties of the system can be easily investigated. The use of hybrid automata, instead of standard ones, simplifies the construction of a qualitative, but complete, model of a biological system. In fact, powerful techniques such as (bisimulation-)projections and collapsing can be "safely" applied to hybrid automata in order to reduce the number of states. In particular, while the bisimulation based projection we present could be applied also to standard automata, the "good" behavior of the collapsing operation with respect to the verification of temporal formulae strongly depends on the information which is stored in each state of hybrid automata. Notice that, although we have presented a construction of hybrid automata from standard S-systems, it is not difficult to modify our framework in order to deal with more complicated systems, e.g., systems whose differential equations change during the evolution of the system itself.

In the future, we intend to extend our tool set in two directions: (1) integrate temporal model checking tools with time-frequency analysis tools capable of identifying distinct "modes" of the system, and (2) incorporate a *learning* scheme in our approach to keep track of a parametrized family of automata in order to identify the kinetic parameters of the system.

References

1. R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G. J. Pappas, H. Rubin, and J. Schug. Hybrid Modeling and Simulation of Biomolecular Networks. In *Hybrid*

Systems: Computation and Control, volume 2034 of *LNCS*, pages 19–32. Springer-Verlag, 2001.

2. R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Richel, editors, *Hybrid Systems*, LNCS, pages 209–229. Springer-Verlag, 1992.
3. M. Antoniotti and A. Göllü. SHIFT and SMART-AHS: A Language for Hybrid Systems Engineering, Modeling, and Simulation. In *Conference on Domain Specific Languages*, Santa Barbara, CA, U.S.A., October 1997. USENIX.
4. M. Antoniotti, F. C. Park, A. Policriti, N. Ugel, and B. Mishra. Foundations of a Query and Simulation System for the Modeling of Biochemical and Biological Processes. In *Proc. of the Pacific Symposium of Biocomputing (PSB'03)*, 2003.
5. M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. XS-systems: extended S-systems and algebraic differential automata for modeling cellular behaviour. In *Proc. of Int. Conference on High Performance Computing (HiPC'02)*, 2002.
6. M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. Model Building and Model Checking for Biological Processes. *Cell Biochemistry and Biophysics*, 2003. To appear.
7. U. S. Bhalla. Data Base of Quatitative Cellular Signaling (DOQCS). Web site at `http://doqcs.ncbs.res.in/`, 2001.
8. R. W. Brockett. Dynamical Systems and their Associated Automata. In *Systems and Networks: Mathematical Theory and Applications*, volume 77. Akademie-Verlag, 1994.
9. M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing Finite Kripke Structures in Propositional Temporal Logic. *Theoretical Computer Science*, 59:115–131, 1988.
10. N. Chabrier and F. Fages. Symbolic Model Checking of Biochemical Networks. In C. Priami, editor, *Computational Methods in Systems Biology (CMSB'03)*, volume 2602 of *LNCS*, pages 149–162. Springer-Verlag, 2003.
11. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An Opensource Tool for Symbolic Model Checking. In E. Brinksma and K. G. Larsen, editors, *Int. Conf. on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 359–364. Springer-Verlag, 2003.
12. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.
13. M. Curti, P. Degano, and C. T. Baldari. Casual pi-calculus for Biochemical Modelling. In C. Priami, editor, *Computational Methods in Systems Biology (CMSB'03)*, volume 2602 of *LNCS*, pages 21–33. Springer-Verlag, 2003.
14. M. Curti, P. Degano, C. Priami, and C. T. Baldari. Casual π-calculus for Biochemical Modelling. DIT 02, University of Trento, 2002.
15. R. Curto, E. O. Voit, A. Sorribas, and M. Cascante. Analysis of Abnormalities in Purine Metabolism leading to Gout and to Neurological Dysfunctions in Man. *Biochemical Journal*, 329:477–487, 1998.
16. R. Curto, E. O. Voit, A. Sorribas, and M. Cascante. Mathematical Models of Purine Metabolism in Man. *Mathematical Biosciences*, 151:1–49, 1998.
17. V. Danos and C. Laneve. Graphs for Core Molecular Biology. In C. Priami, editor, *Computational Methods in Systems Biology (CMSB'03)*, volume 2602 of *LNCS*, pages 34–46. Springer-Verlag, 2003.
18. H. de Jong. Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. DIT 4032, Inria, 2000.

19. G. Delzanno and A. Podelski. DMC User Guide. 2000.
20. A. Dovier, C. Piazza, and A. Policriti. A Fast Bisimulation Algorithm. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. of Int. Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 79–90. Springer-Verlag, 2001.
21. M. Elowitz and S. Leibler. A Synthetic Oscillatory Network of Transcriptional Regulators. *Nature*, 403:335–338, 2000.
22. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. MIT Press, 1990.
23. T. A. Henzinger. The Theory of Hybrid Automata. In *Proc. of IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Press, 1996.
24. T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HYTECH: A Model Checker for Hybrid Systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.
25. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.
26. S. James, P. Nilson, J. James, S. Kjellenberg, and T. Fagerstrom. Bioluminescence Control in the Marine Bacterium Vibrio Fischeri: An analysis of the dynamic lux regualtion. *J Mol Biol*, 296(4):1127–1137, 2000.
27. P. D. Karp, M. Riley, S. Paley, and A. Pellegrini-Toole. The MetaCyc Database. *Nucleic Acid Research*, 30(1):59, 2002.
28. P. D. Karp, M. Riley, M. Saier, and S. Paley A. Pellegrini-Toole. The EcoCyc Database. *Nucleic Acids Research*, 30(1):56, 2002.
29. KEGG Database. `http://www.genome.ad.jp/kegg/`.
30. H. Kitano. Systems Biology: an Overview. *Science*, 295:1662–1664, March 2002.
31. D. Kozen. Results on the Propositional mu-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.
32. H. H. McAdams and A. Arkin. Simulation of Prokaryotic Genetic Circuits. *An. Rev. Biophis. Biomol. Struct.*, 27:199–224, 1998.
33. O. Müller and T. Stauner. Modelling and Verification using Linear Hybrid Automata. *Mathematical and Computer Modelling of Dynamical Systems*, 6(1):71–89, 2000.
34. R. Paige, R. E. Tarjan, and R. Bonic. A Linear Time Solution to the Single Function Coarsest Partition Problem. *Theoretical Computer Science*, 40:67–84, 1985.
35. PathDB Database. `http://www.ncgr.org/pathdb/`.
36. A. Regev, W. Silverman, and E. Shapiro. Representation and Simulation of Biochemical Processes using the π-calculus Process Algebra. In *Proc. of the Pacific Symposium of Biocomputing (PSB'01)*, pages 459–470, 2003.
37. D. M. Sitnikov, J. B. Schineller, and T. O. Baldwin. Transcriptional Regulation of Bioluminescence Genes from Vibrio Fischeri. *Mol. Microbiol.*, 17(5):801–812, 1995.
38. E. O. Voit. *Computational Analysis of Biochemical Systems. A Pratical Guide for Biochemists and Molecular Biologists.* Cambridge University Press, 2000.
39. WIT Database. `http://wit.mcs.anl.gov/WIT2/`.