Model Building and Model Checking for Biochemical Processes

Marco Antoniotti[1], Alberto Policriti[2], Nadia Ugel[1], Bud Mishra[1,3]
[1] Courant Institute of Mathematical Sciences, NYU, U.S.A.
[2] Università di Udine, ITALY
[3] Cold Spring Harbor Laboratory, U.S.A.

Abstract

A central claim of computational systems biology is that, by drawing upon mathematical approaches developed in the context of dynamical systems, kinetic analysis, computational theory and logic, it is possible to create powerful simulation, analysis and reasoning tools for working biologists to be used in deciphering existing data, devising new experiments and ultimately, understanding functional properties of genomes, proteomes, cells, organs and organisms.

In this paper we describe a novel computational tool that achieves many of the goals of this new discipline. The novelty of this system involves an automaton-based semantics of the *temporal evolution* of complex biochemical reactions starting from the representation given as a set of differential equations. More importantly, the related tools also provide ability to qualitatively reason about the systems using a propositional temporal logic that can express ordered sequence of events succinctly and unambiguously. The implementation of our mathematical and computational models in the **Simpathica** and **XSSYS** systems is described briefly. Several example applications of these systems to cellular and biochemical processes are presented: the two most prominent ones are Leibler et al.'s repressilator (an artificial synthesized oscillatory network) and Curto-Voit-Sorribas-Cascante's purine metabolism reaction model.

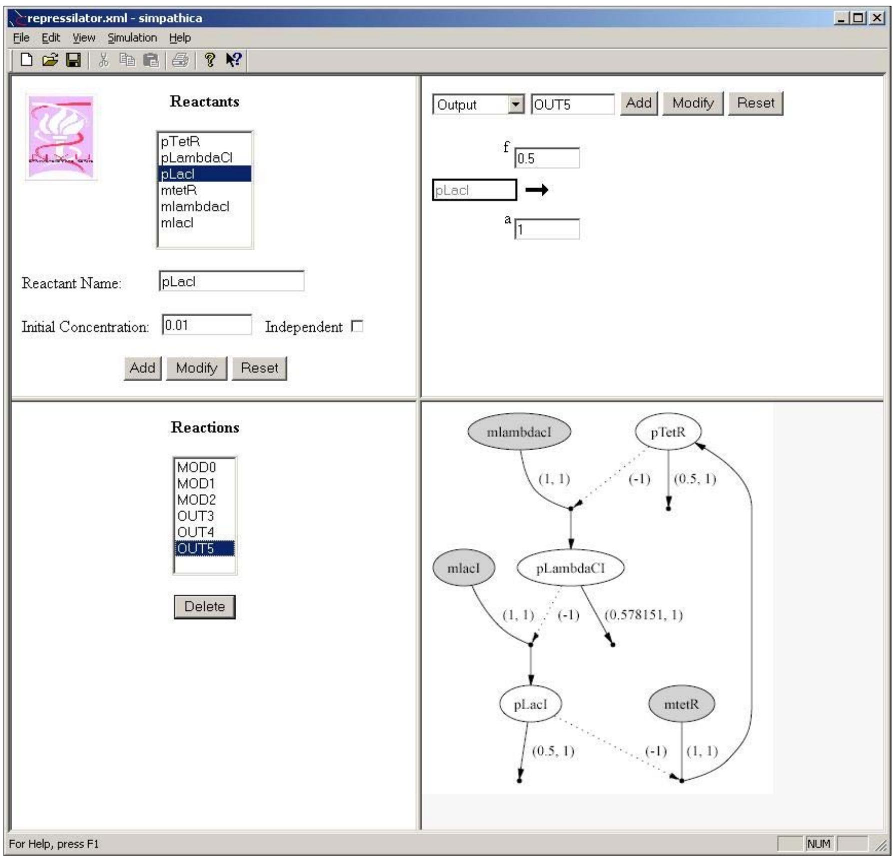
Index Entries: Biochemical Reactions, Biological Models, Cellular Processes, Model Building, Model Checking, Purine Metabolism, Repressilator and Temporal Logic.

1 Introduction

Recent advances in genomics have made it possible for the first time for a biologist to access enormous amounts of information for a number of organisms, including human, mouse, arabidopsis, fruit fly, yeast and *E. coli*. These developments are at the heart of the many renewed ambitious attempts by biologists to understand the functional roles of a group of genes using powerful computational models and high throughput microbiological protocols. The emerging fields of system biology, 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 transcriptional states of a cell (using micro-arrays) and translational states of the cell (using proteomics tools). In the process, it has become evident that the mathematical foundation of these systems needs to be explored accurately and that their computational models be implemented in software packages faithfully while exploiting the potential trade-offs among usability, accuracy, and scalability dealing with large amounts of data.

Several biological and biochemical mechanisms can be modeled with relatively simple sets of differential algebraic equations (DAE). The numerical solution to these differential equations provides a potentially powerful and effective investigative tool for biologists and biochemists. In this paper, we demonstrate the power of a novel computational tool with the ability to query massive sets of numerical data obtained from *in silico* experiments on complex biological systems. The computational tool 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 this paper we describe *XS-systems*: a new computational model extending the basic foundations provided by the "S-systems models of biochemical processes." The main innovative extension provided by the *XS-system* involves an automaton-based semantics of the *temporal evolution* of complex biochemical reactions starting from the representation as a set of differential equations. The implementation of our mathematical and computational models in the **Simpathica** and **XSSYS** systems will be described briefly (See Figures 1 and 2 for screen-shots of the two systems). However, a detailed discussion of the underlying mathematical foundations will be omitted, in order to keep the paper accessible to a wider readership. The main emphasis of the paper will remain on biological applications illustrating how we envision **Simpathica** and **XSSYS** to be used in practice. 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.


repressilator.xml - simpathica

File Edit View Simulation Help

Reactants

pTetR
pLambdaCl
pLacl
mtetR
mlambdacl
mlacl

Reactant Name: pLacl

Initial Concentration: 0.01 Independent ☐

Add Modify Reset

Output OUT5 Add Modify Reset

f 0.5

pLacl →

a 1

Reactions

MOD0
MOD1
MOD2
OUT3
OUT4
OUT5

Delete

For Help, press F1 NUM

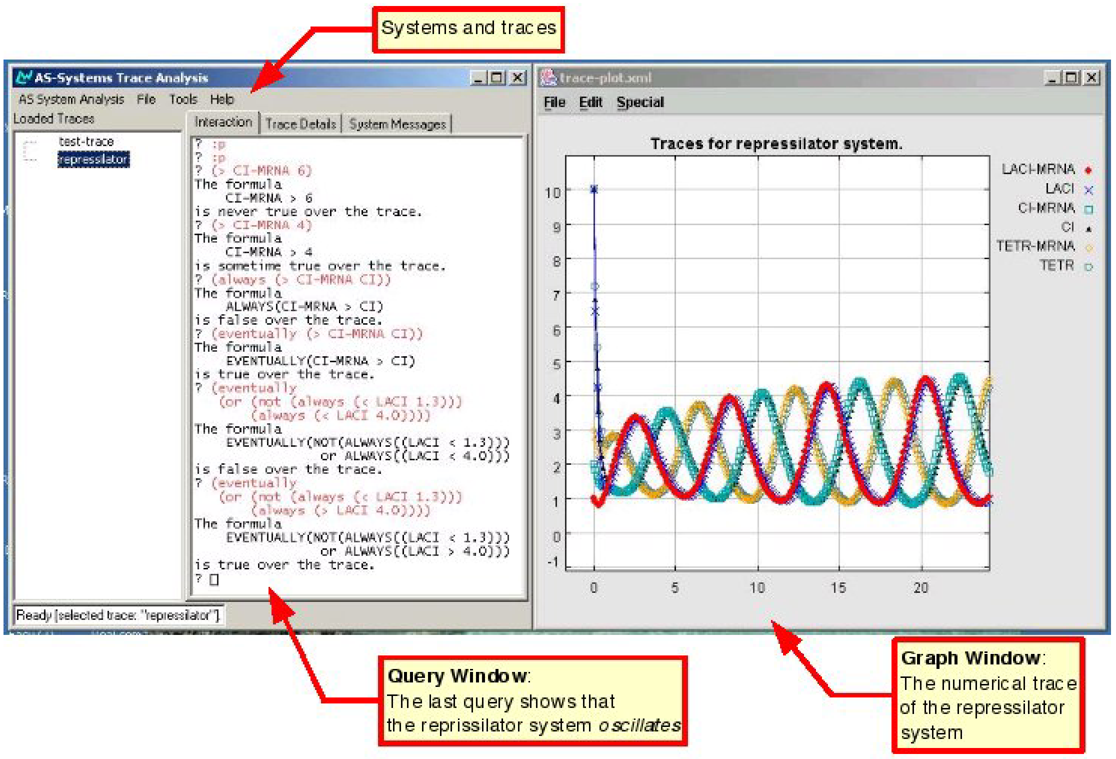
*Figure 1. The **Simpathica** Main Window. The system being analyzed is the "repressilator" circuit [2]. The upper left frame contains a list of the reactants. The upper right frame is used to insert different kinds of reactions. The lower left frame contains a list of known reactions. Finally the lower right frame contains a depiction of the reactions' network.*

The rest of the paper is organized as follows: Section 2 describes our models of biological experiments and how these models can be interpreted in terms of an automaton whose structure is determined by both numerical and analytic solutions of a set of "parameterized" differential equations. Section 2 also contains the description of a temporal logic language for expressing and verifying properties of *XS-systems* together with a prototype implementation. Section 3 contains the description of a simple system (the *repressilator* [2]) with a walk through of **Simpathica**. Section 4 contains a more complex biological example and demonstrates how the computational tools of this paper are applied. Section 5 concludes the paper.

2 Experiments and Simulation

Imagine a computational biologist about to perform simulations of complex biochemical pathways in conjunction with related experimental data collection. The researcher will often model the underlying biological and biochemical mechanisms with sets of relatively simple differential algebraic equations (DAE), each one representing a reversible chemical reaction, a degradation process, a synthesis process or a reaction modulated by an enzyme or a co-enzyme. The numerical solutions to these differential equations and the time-series "tracing" the evolutions of an RNA transcript, protein or a lipid, etc. provide the basic ingredients for data interpretation, data validation and hypothesis formation and falsification.

As the model complexity of the biological systems increases, the sets of numerical *traces* become increasingly difficult to interpret and the traditional biological reasoning process fails to scale beyond a handful of genes and relatively small and coarsely-modeled pathways. To cope with this problem, we propose a novel approach that first summarizes the numerical traces into an automaton with distinguishable biological states and a deterministic set of rules of transition from state-to-state and finally, checks the automaton model for its ability to satisfy various temporal logic statements.



```
? :p
? :p
? (> CI-MRNA 6)
The formula
    CI-MRNA > 6
is never true over the trace.
? (> CI-MRNA 4)
The formula
    CI-MRNA > 4
is sometime true over the trace.
? (always (> CI-MRNA CI))
The formula
    ALWAYS(CI-MRNA > CI)
is false over the trace.
? (eventually (> CI-MRNA CI))
The formula
    EVENTUALLY(CI-MRNA > CI)
is true over the trace.
? (eventually
   (or (not (always (< LACI 1.3)))
       (always (< LACI 4.0))))
The formula
    EVENTUALLY(NOT(ALWAYS((LACI < 1.3)))
                 or ALWAYS((LACI < 4.0)))
is false over the trace.
? (eventually
   (or (not (always (< LACI 1.3)))
       (always (> LACI 4.0))))
The formula
    EVENTUALLY(NOT(ALWAYS((LACI < 1.3)))
                 or ALWAYS((LACI > 4.0)))
is true over the trace.
? □
```

Figure 2. The main ***XSSYS*** *view. The core XSSYS interface is on the left. The left frame contains a list of the "loaded" traces (obtained from simulators such as the Simpathica/Octave engine or PLAS [4]). The right frame is used to type in various temporal logic queries and to check the results. The right window is simply a plotting application (PtPlot from UC Berkeley), which we use for visualization purposes.*

Our starting reference point is the classical S-systems, described in [4,7], and the idea that a natural completion for that approach would be an automaton *summarizing* the states along which the simulated biochemical system evolves in time. The automaton, so generated, allows the user to view, manipulate and reason about, using a well-integrated set of tools.

As an example (to be explored further), consider the case study of purine metabolism from Chapter 10 of [4]. This case study, as many others, illustrates the fact that the "right" cellular behavior is often difficult to capture with an initial abstraction model. Often it is necessary to improve the model in many successive iterations, each step involving a more accurate estimation of some model parameters; identification and elimination of some “structural” problems in the set of equations; or incorporation of some new unexpected insight obtained by closer examination of an intermediate model. Our thesis is that rapid successive refinement of a model is facilitated through the model checking algorithms and the underlying formalisms provided by temporal logic formulae are capable of identifying the *missing features* in a partial/incomplete model.

2.1 Mathematical Models, Differential Equations and Canonical Forms

Biochemical reactions can be modeled with sets of differential equations. The classical Michaelis-Menten's formulation of *reaction speed* is essentially differential equations for the rate of change of the product of an enzymatic reaction. The parameters of such equation are the constants K_m (Michaelis-Menten Constant) and V_{max} (maximum velocity of a reaction). An S-system is simply a set of ordinary differential equations where each equation appears in a particularly simple form and models the rate of change in the concentration of a product (or substrate) in terms of its synthesis from and degradation into other reactants.

Canonical Forms. A set of differential equations can be rewritten (recast) in special *canonical forms* by purely algebraic transformations and further inclusions of a set of algebraic constraint equations. S-systems are sets of ODEs in canonical form. Canonical forms have several advantages over more general forms of equations, since they can be more easily manipulated, integrated and interpreted in mathematical terms.

The extended S-system model we use – XS-systems – has a simple canonical form. An XS-system is simply a list of expressions describing the rate of change of a given quantity in a model (say the concentration of a compound), plus a set of equations describing some constraints on the relationships among some of the parameters characterizing the model. These constraints may or may not be needed for the complete model to be meaningful. Their presence depends on the system under exam. Each of the expressions describing a rate has a very simple form as well: it is simply a difference between two algebraic power-products (or monomials) one representing synthesis and the other, dissociation (i.e. it is a S-system). More formally we have the following.

Definition 1 *An* XS-system *is defined by a set of pairs of equations (a rate equation and a constraint equation)*

$$\dot{X} = \left(\alpha_i X_1^{g_{1i}} X_2^{g_{2i}} \cdots X_n^{g_{ni}}\right) - \left(\beta_i X_1^{h_{1i}} X_2^{h_{2i}} \cdots X_n^{h_{ni}}\right)$$

$$\left(a_{1j} X_1^{c_{11j}} X_2^{c_{21j}} \cdots X_n^{c_{n1j}}\right) + \left(a_{2j} X_1^{c_{12j}} X_2^{c_{22j}} \cdots X_n^{c_{n2j}}\right) + \cdots + \left(a_{mj} X_1^{c_{1mj}} X_2^{c_{2mj}} \cdots X_n^{c_{nmj}}\right) = 0$$

with index variables, i ranging from 1 to n, and j, from 1 to k. This formalism describes an XS-system with n equations and k constraints.

An XS-system can be interpreted as the representation of a set of flows of reactants within a network of reactions [4] and thus describes how to translate a graphical rendition of such reaction networks into the classical S-System. Our XS-system formulation naturally captures these steps in a computer-assisted translation, which had been traditionally carried out by a manual manipulation; see [4].

The XS-system formulation makes one more distinction between *dependent* and *independent* variables. Independent variables represent environmental conditions which influence the behavior of the system but which do not influence themselves in return. Dependent variables are all the others. Of course, to complete the description of the system it is necessary to specify all the *rate constants* (α's and β's) and the *kinetic orders* (*g*'s, *h*'s, and *c*'s) of each equation and constraint.

2.1.1 Characterizing the behavior of the system across different "states"

Although the dependent and independent variables of an S-system give a *quantitative* description of the reactants (substrates, products, enzymes, etc.) involved in the experiment, a tool for the *qualitative* analysis of the system is still missing. We have developed the theoretical framework for such a tool and provided a first implementation in the **XSSYS** system.

We note that the *data* we can count on in the development of our tool are not only *numerical* (i.e. the solution of the power-law differential equations defining the S-system) but also *logical,* appearing in the form of constraints, known to the biologist modeling an experiment, and relating values of the substances involved.

A simple and natural example of a logical property of many biological systems is the one describing the existence of a *steady-state.* Informally, a system is in a "steady state" when nothing "changes" in the system as time passes. More formally, for the purpose of our discussion, the "steady state" is reached when all the first derivatives of the functions describing dependent variables are equal to zero. The software tool can check this event by solving an algebraic problem to detect the existence of a common root (this operation may be rather involved, see [12]).

Very often the biologist not only knows that in the absence of external stimuli, such a state *must* be reached sooner or later, but also knows what are (at least) the relative values of substances involved in such a state. Another natural property involves boundedness of the reactant concentrations involved in a biological process and may need to be ascertained as a precondition to other interesting properties such as existence of a limit-cycle or steady-state behavior.

The key to manipulate "qualitatively" these notions (e.g. the notion of steady-state) lies in grouping each instant in time and the corresponding values of each variable into "*states*". In the simplest case we simply denote each instant in time (our *sampled* time) as associated to a "state". More interesting states (with deteriorating computational efficiency implications) are constructed by *grouping* several time instants according to some simple rules, e.g. a linearization rule that groups states, if their rates of change are within a user defined parameter.

Such construction yields an "*automaton,*" a common abstraction tool used in computer science and other engineering disciplines. The automaton we construct allows us to capture qualitative features of a biological system, e.g., the notion of a steady state. Note that the construction of the automaton will eventually be hidden from the user. The automaton serves as a tool to answer queries about the evolution of the system, and its construction is provided by the software tool.

Definition 2 *Given an S-system E, the S-system automaton AS describes a set of qualitative states, S, of the system together with rules of state-transitions, Δ. More formally, an AS associated with E is a 4-tuple AS = (S, Δ, S_0, F), where S $\subseteq D_1 \times \cdot\cdot\cdot \times D_{n+m}$ is a (finite or infinite) set of states, Δ $\subseteq$ S × S is the transition relation, and S_0, F $\subseteq$ S are the initial and final states, respectively.*

The key idea is that the values of the dependent and independent variables uniquely characterize the *state* of the system and that the collection of such values, together with a relation governing the possible transitions, constitutes the automaton qualitatively describing the behavior of the system.

The subsequent steps consist in providing the biologist with a *language* to impose constraints on qualitative features of the system under study. To this end, we introduce a notion that renders the temporal evolution of the system in terms of a trace:

Definition 3 *A trace of an S-system automaton AE is a (finite or infinite) sequence $s_0, s_1, \dots s_n, \dots$ such that s_0 is an initial state ($s_0 \in S_0$) and there is a transition rule allowing the automaton to enter s_{i+1} from s_i, (Δ(s_i, s_{i+1}) holds) for all $i \geq 0$.*

A trace can also be defined as:

$$trace(AE) = \langle\langle X_1(t) \dots X_{n+m}(t)\rangle \mid t \in \{t_0 + k\,\mathbf{step} : k \geq 0\}\rangle,$$

is called the trace of AE.

A *trace* of an S-system automaton is therefore a sequence of arrays of values that allows a complete description of the dynamics of the reactants within a fixed time interval $[t_0, t]$. The precision of the description is parametric in the value of the **step** variable: the smaller the **step** the higher the precision.

The following definition is used in order to "focus" the automaton on certain dependent values determined by a fixed subset of variables --- these are the variables used explicitly or implicitly in the description of a qualitative property or needed by the quantitative analysis.

Definition 4 *Given any set of variables U $\subseteq \{X_1, \dots, X_{n+m}\}$, the sequence:*

$$trace(AE|U) = \langle\langle X_i(t) \mid X_i \in U\rangle : t \in \{t_0 + k\,\mathbf{step} : k \geq 0\}\rangle,$$

is called the trace of U.

If U consists of a single variable X_i the trace is called the trace of X_i.

Notice that a single trace of the automaton results when only one “set-up” (e.g., initial conditions, a set of values for the parameters, a set of signaling events, etc.) for the system is being considered. In order to allow more than one trace, it is necessary to consider different “set-ups,” e.g., many possible values for the parameters, such as *rate constants* and *kinetic orders*. If multiple traces are available, they can be combined within a single model containing different possible evolutions of the system: a very common situation easily handled by any *branching time* temporal logic, which allows a conceptual clock to split intermittently in order to model simultaneous evolutions in many possible worlds. Of course, the question of how many of these traces are necessary to assess the validity of the model is still open, we envision our tools to be used in an iterative fashion, converging to a “good” model of the system under investigation.

Regardless of whether one single trace or many of them are combined into a unique model, the number of resulting qualitative states as defined above can be prohibitively large and may ultimately be redundant for a qualitative analysis. In order to optimize the computational efficiency of the quantitative analysis, we have implemented a *collapse* operation combining those states that are indistinguishable as the numerical values characterizing them are same or only differ by imperceptively small amount.

A distinctive feature of the collapsing operations is that they can be interleaved with the phase performing the numerical computation of the approximate solutions of the power-law differential equations. This feature guarantees that the collapse does not impose a heavy computational burden on the entire system, during the process of producing a more genuine temporal logic model (the automaton) for the S-system.

As a consequence, once this automaton has been constructed, it provides many different avenues for:

a) Performing a qualitative analysis on the temporal evolution of the S-system, and

b) Studying in *parallel* (within a single structure) multiple evolutions and experiments differing in rate constants and kinetic orders, e.g. wild-types and many mutants.

Note that the automaton proposed here are not necessarily unique and one may consider more complex automata with different semantics and amenable to different logical analysis. For instance, a *timed* automaton (with quantitative temporal information or with constraint labeling the transitions) could be produced at no additional cost during the same numerical computation. Currently, other variant automata are under active investigation and we anticipate several more novel model automata to be incorporated into our final tool. In particular we are investigating how to consider several traces at a time and how to build incrementally a Hybrid Automata without loss of information: this will address the issues of *exhaustivity* which must be considered with care in order to assess the overall validity of a model.

2.1.2 Temporal Logic (TL)

Temporal Logic (TL) [8,10] has been studied in depth in the context of systems whose behavior change in time, for instance, computer hardware, network protocols and engineering systems. We omit a detailed introduction to any or all of many specific Temporal Logics that have been introduced in the past. Instead we concentrate on the main ideas at the core of these logics in order to provide the intuition about how it can be used in the analysis of biochemical systems.

Fundamental to a temporal logic is the notion that time-dependent terms from natural language, such as “eventually” and “always,” can be given a precise meaning (semantics) in terms of the abstract behavior of a system under discourse. As an example, consider the following sentence:

The concentration of guanosin triphosphate (GTP) is equal to x.

Such a sentence is *true* only in certain circumstances. Given a biological system in equilibrium the above sentence may or may not be true at any or all instants of time. In particular, we can easily construct sentences (in a suitable natural language) that express the fact that, *given a certain set of initial conditions* the above sentence *will **eventually** hold true*. Temporal Logic precisely formalizes the meaning of the adverb *eventually* (and other such “modes”: *always*, *infinitely often* and *almost always*) and the resulting semantics lead to a precise model-checking algorithm for determining the validity of TL sentences in the context of an automaton.

This particular attribute of TL is very important as it concisely captures the notion of a logical property like “steady-state” and formalizes this notion in a simple consistent way that is directly handled by the model-checking algorithm.

Consider a system *M* and a (simulation) trace **trace***(M).* If we consider a state *s* in **trace***(M)***,** we can simply check if all the first derivatives in *s* are 0. Suppose we have a procedure that answers *yes* (or *no*) when this is the case. Let us call this predicate, **zero_derivative**. Suppose that there actually is a state *s'* in **trace***(M)* where **zero_derivative** yields *yes*. Now, by the rules of Temporal Logic the following statement would be *true*

```
Eventually(zero_derivative)
```

for each instant from the start, at least up until the instant characterized as state s'.

Now we can expand the language of Temporal Logic and introduce a new predicate “steady state” to be a synonym of the following concept: there exists an instant (a state s' in **trace***(M)*) after which **zero_derivative** will always be true. More formally,

```
steady_state(M)
```

is defined to be logically equivalent to the following:

```
Eventually(Always(zero_derivative))
```

meaning that, when we consider the simulation (or *in vivo*) trace of the system there will be a time where all the rates of change of the system's variables reach 0 and remain at that value.

Alternatively, we could be more selective and ask whether some specific variable reaches the steady state. We can determine the answer as a result of the Definition 4.

```
steady_state(M, GTP).
```

Another set of properties that we may want to express (and subsequently check) is the one involving “persistence.” In other words, properties of the form: *something is **always** true* (or *false*). For instance, we could ask whether in a given system

```
Always (GTP > k).
```

Thus, we query whether the GTP level always remains greater than k, independent of other changes occurring during the evolution of the system.

How to translate a statement in English into Temporal Logic. The previous discussion illustrates the main ideas needed to translate an English sentence involving temporal claims into a query in temporal logic. The translation from English to TL is rather straightforward. Simple conjunctions ("and"s), disjunctions ("or"s) and negations ("not"s) can be expressed directly. The corresponding prepositional logic is then augmented with temporal modes: "Always" and "Eventually."

Now, suppose we wish to determine if (1) our system reaches a steady state *and* (2) the level of GTP is less than k after a certain instant. This statement is simply expressed in TL as

```
steady_state and Eventually(Always(GTP < k)).     (a)
```

Note that the validity of the above statement is completely determined by the two constituent sub-expressions. Furthermore, the truth property of the statement requires examining the entire system trace, since **steady_state** is a "global" property, and the second conjunct has the same form. To appreciate the subtleties of TL, consider the following expression: eventually the system will be in steady state and the level of GTP will be less than k.

```
Eventually(steady_state and Always(GTP < k))     (b)
```

Given the properties of TL, the above expression (if true) will actually guarantee that when the system attains the steady state, it *also* has a GTP level less than k. This is a different statement than (a), and it shows how flexible and yet precise a TL statement can be, without sacrificing a high degree of expressive power. In [11] we discuss some of the mathematical and computational problems associated with this approach, e.g. the dependency of the analysis on the density of time points.

3 A Simple Example: the *Repressilator*

As a simple yet very interesting example, consider the *repressilator* system constructed by Elowitz and Leibler [2]. 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^{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.

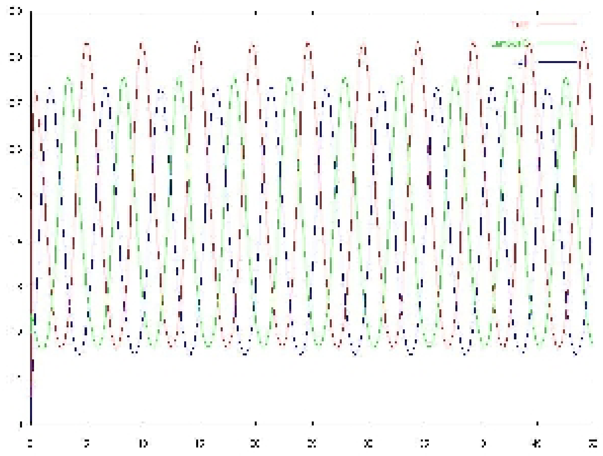


Figure 3 The simulation trace of the repressilator system.

The observed trace of the six combined variables is shown in Figure 3. The system exhibits an oscillatory steady state.

Simpathica was used to enter the description of the repressillator system and to analyze its behavior. Although this is a simple toy example for our application, it still presents a clear idea regarding how a biologist may use the Simpathica system.

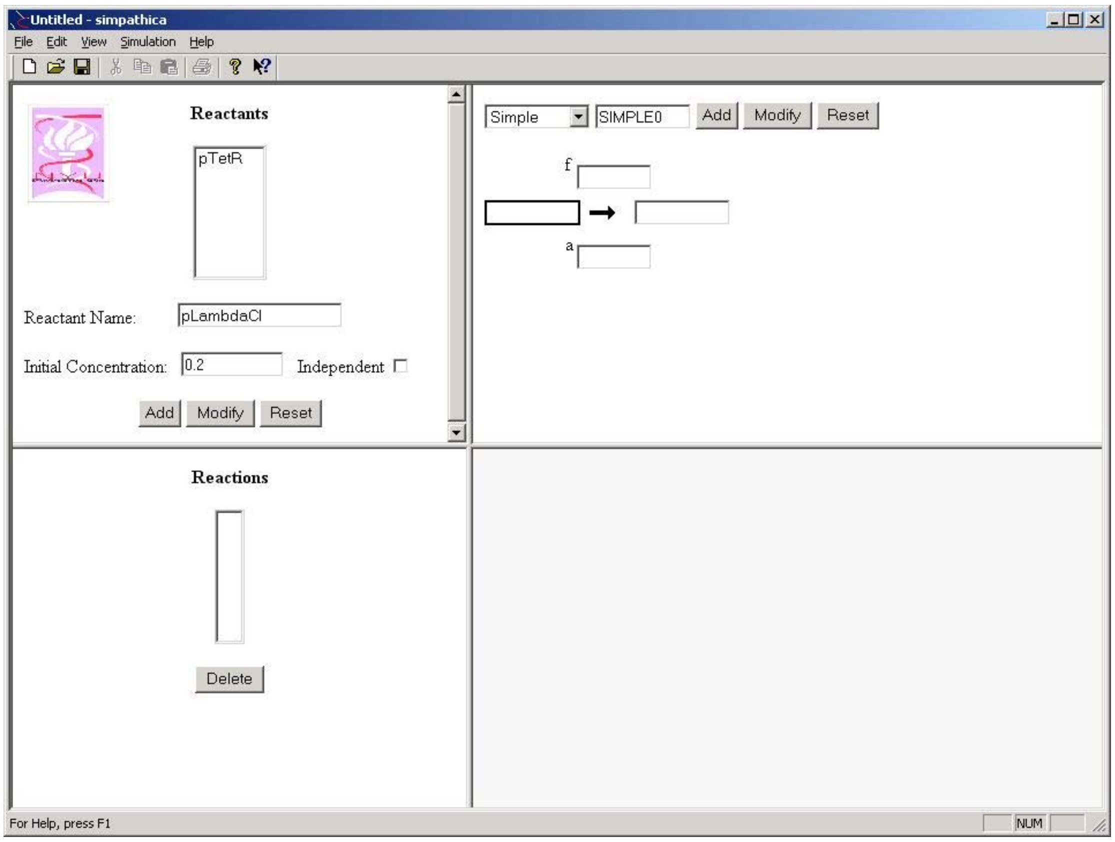


Figure 4. Entering a reactant. The name for the quantity we want to use in the simulation (in this case pLambdaCI*) is entered in the text field along with the initial concentration (upper left frame).*

Figure 4 Shows how to use SIMPATHICA to enter a reactant (in this case *pLambdaCI* – the protein *LambdaCI*). In Figure 5 a reaction is inserted in the system and the graphical depiction is immediately visible in the bottom left frame; in this case the reaction just entered is a modulated reaction: the production of *TetR* is inhibited by the amount of *LambdaCI* present. Figure 6 shows how to enter a second reaction in the system.

Once all the reactions have been entered, along with a set of initial conditions, the menu “Simulation->Run Simulation” will perform the following steps.

1.Generate the appropriate set of differential equations in canonical form. (**Simpathica** strives to generate an S-System from a given network – if it cannot, it generates a GMA-system).

2.Start the integrated Octave program (http://www.octave.org) to simulate the system.

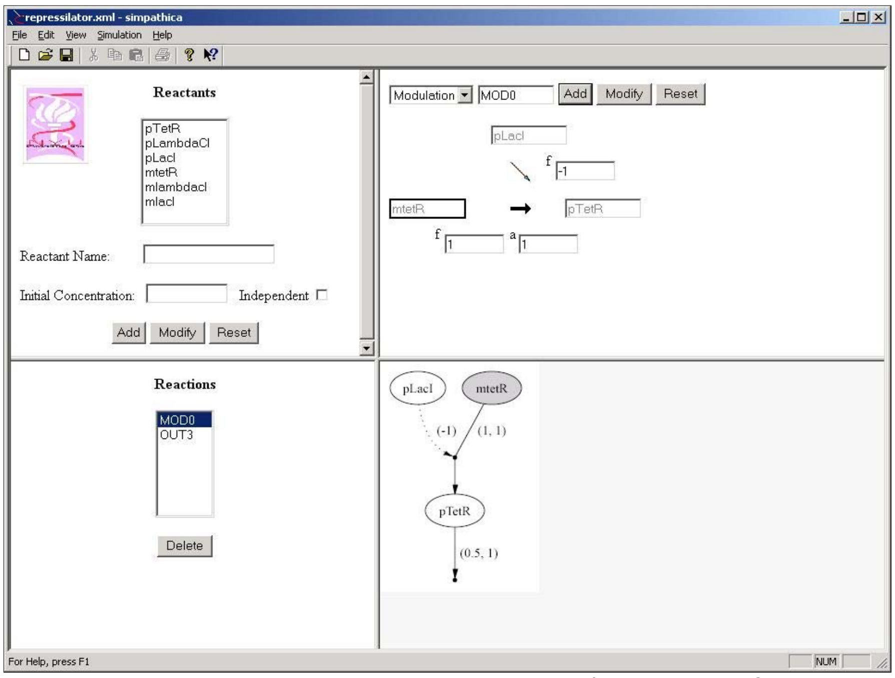


Figure 5. Entering a reaction. A modulated reaction (production of TetR*, inhibited by* LambdaCI*) is entered in the system. The rate of the reaction is* a = 1*. The inhibition effect is expressed by the exponent* f = -1*, assigning a label to the modulation arrow (the light one originating from the* pLacI *field). Note the graphic rendition in the lower right frame. The grayed oval represents the "input" of the reaction, while the dotted arrow represents the modulation effect of* pLacI*.*

Finally, we can use our system to verify that the repressilator system's variables actually "oscillate." We may formulate and test a query such as the one shown below (for variable *pLambdaCI*):

```
Eventually[not Always(pLambdaCI < 0.25)
            or Always(pLambdaCI > 0.5)].
```

The above query states that the value of the `pLambdaCI' variable oscillates[1] between the two extremes of 0.25 and 0.5.

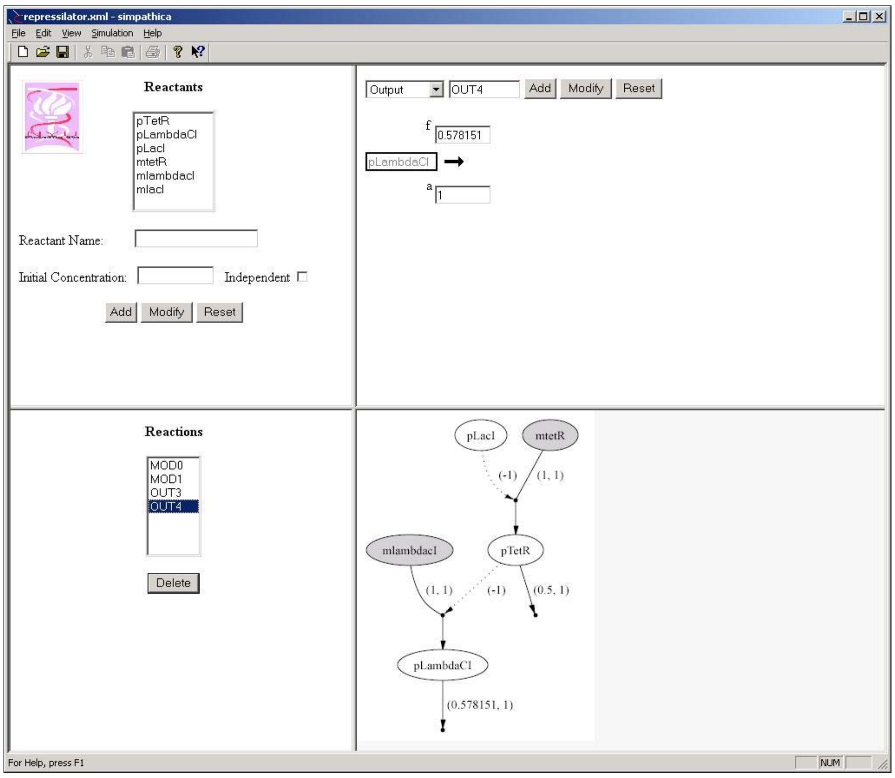
The repressilator example is interesting because it indicates how well our system scales when many variables are present in the model. In a more recent work by Guet et al. [9], the authors show how to construct plasmids containing many more kinds of promoters (e.g., five) modulating these genes, thus potentially many more gene network topologies. Asking precise

1A more precise, yet more complex, query would be

```
Eventually[(pLambdaCI < 0.25)
          and Always[(plambdaCI < 0.25)
                     implies Eventually[plambdaCI > 0.5]]
          and Always[(plambdaCI > 0.5)
                     implies Eventually[plambdaCI < 0.25]]].
```

This query seems to be common enough to warrant a "macro" like `steady_state()`.

and circumscribed queries about the behavior of such combinations is much more preferable to visually examining complex graphical renditions of the complex interacting patterns of reactant concentrations.


repressilator.xml - simpathica

Reactants: pTetR, pLambdaCl, pLacl, mtetR, mlambdacl, mlacl

Reactant Name:

Initial Concentration: Independent ☐

Output: OUT4

f 0.578151

pLambdaCl →

a 1

Reactions: MOD0, MOD1, OUT3, OUT4

Figure 6. Entering a second reaction. In this case we enter a new simpler reaction. Note how the new reaction is rendered in the bottom right frame and how it appears in the reactions list in the bottom left frame.

4 A More Complex Example: *Purine Metabolism*

Let us now revisit in detail the example of *purine metabolism* described in [4] Chapter 10 and fully analyzed in [5,6]. The pathway for purine metabolism is presented in Figure 7. A brief description of the key reactions follows, and the reader is invited to examine the more detailed summaries contained in [4,5,6] as well as the related literature referenced there.

The main metabolite in purine biosynthesis is *5-phosphoribosyl-α-1-pyrophosphate* (*PRPP*). A linear cascade of reactions converts PRPP into *inosine monophosphate* (*IMP*). 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 *hypoxanthine* (*HX*) and *xanthine* (*XA*). XA is finally oxidized into *uric acid* (*UA*). 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, *adenine* *phosphoribosyltransferase* (*APRT*) and *hypoxanthine-guanine phosphoribosyltransferase* (*HGPRT*) 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.

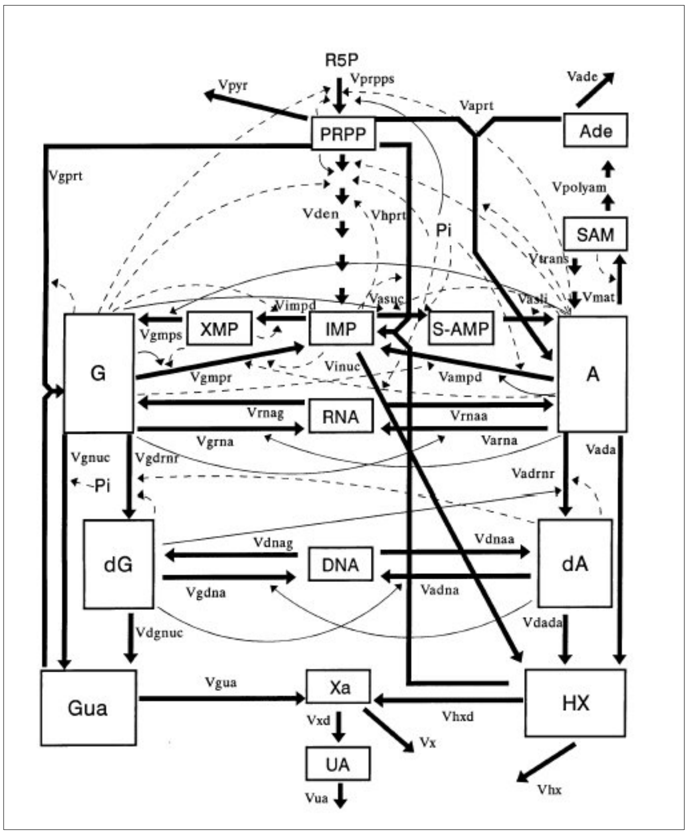


Figure 7.The metabolic scheme of purine metabolism in human. (Reprinted from [6], where a full description and further references may be found.)

In [4], 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 also 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.

4.1 Model 2

Given the purine metabolism model labeled as "model 2" in [4], and following the example closely, we start by querying the simulation trace for the reachability of a *steady state*. This property is easily formulated in our framework with the following query

```
steady_state().
```

A precise definition of the predicate steady_state in terms of other atomic predicates has been given earlier in the paper. In a similar manner we proceed to other interesting queries, as illustrated below.

1. *Variation of the initial concentration of PRPP does not change the steady state.*

   ```
   (PRPP = 10 * PRPP1) implies steady_state()
   ```

   This query will be true when evaluated against the modified simulation run (i.e. the one where the initial concentration of PRPP is 10 times the initial concentration in the first run – PRPP1). Figure 8 illustrates how XSSYS treats such a query.

2. *Persistent increase in the initial concentration of PRPP does cause unwanted changes in the steady state values of some metabolites.*

   In this case, if the increase in the level of PRPP is in the order of 70% (see [4]) then the system does reach a steady state, and we expect to see increases in the levels of IMP and of the hypoxanthine pool in a "comparable" order of magnitude. This means that

   ```
   Always (PRPP = 1.7*PRPP1) implies steady_state()
   ```

   will be true over the modified experiment trace. Note the use of "always" in this TL query to indicate the persistent increase of the PRPP value. For a contrast, consider the following statement:

   ```
   Eventually(Always (PRPP = 1.7 * PRPP1)
               implies
               steady_state()
               and Eventually(Always(IMP < 2 * IMP1))
               and Eventually(Always(hx_pool < 10*hx_pool1)))
   ```

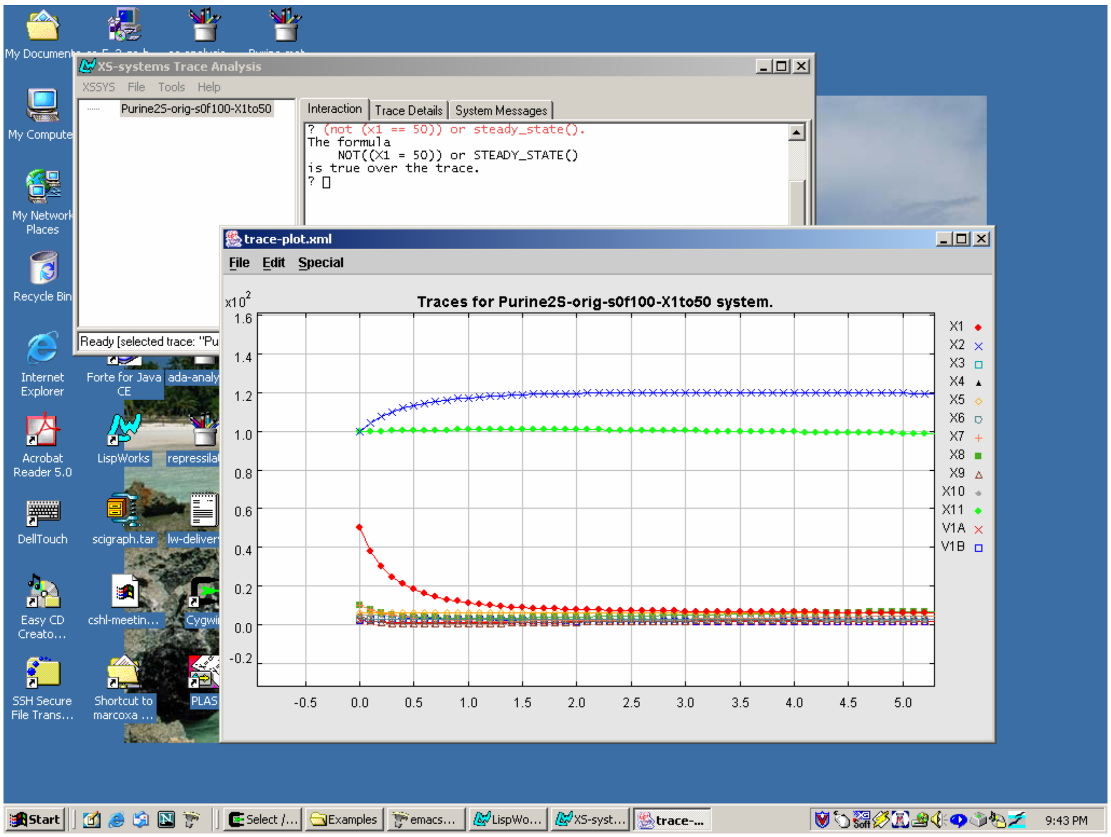
   where IMP1 and hx_pool1 are the values observed in the unmodified trace. The above statement turns out to be false over the modified experiment trace. In fact, the increase in IMP is about 6.5 fold while the hypoxanthine pool increase is about 60 fold. Figure 9 illustrates how XSSYS treats such a query.

Since the above queries turn out to be false over the modified trace, we conclude that the model "over-predicts" the increases in some of its products and that it should therefore be amended.

4.2 Modes 3, 4, and “Final”

The following sequence of models (leading to the “final” one, which is adapted from [5,6]) improves their response as compared with known clinical data. Again, we consider a simple example from [4] where we express the properties being tested with our TL language.

Temporary perturbations. We reused the model published in [4] and the data generated with PLAS software with a variation. The PLAS model reaches the steady state, and 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 concentration level also rises and HX level falls before returning to predicted steady state values.



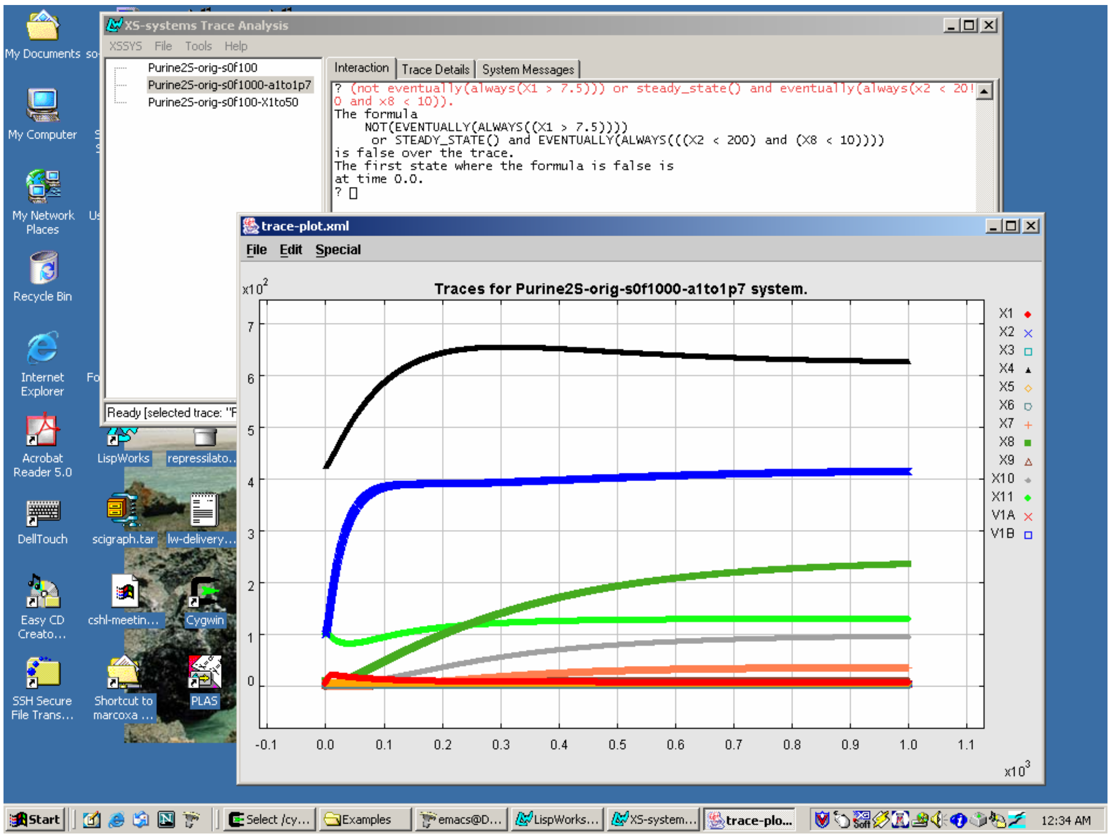
```
? (not (x1 == 50)) or steady_state().
The formula
    NOT((X1 = 50)) or STEADY_STATE()
is true over the trace.
? 
```

Figure 8. The first example discussed in the context of “Model 2”. The variable X1 is PRPP and the fact that the initial condition is changed is expressed as stated.

These results are tested with the same TL expressions we described earlier. However, we need to make a change to our model to make PRPP increase only after a certain time after the simulation has started. This change allows us to reformulate our query as follows:

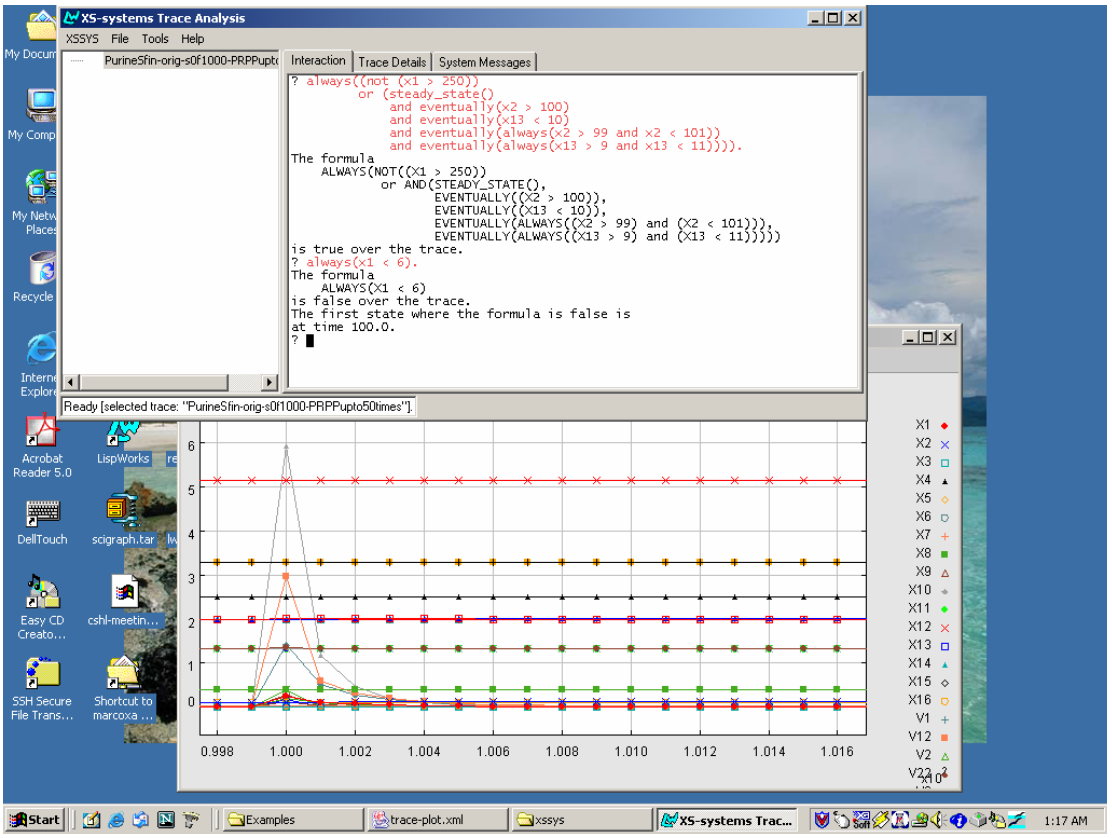
```
Always(PRPP > 50 * PRPP1
             implies
             (steady_state()
              and Eventually(IMP > IMP1)
              and Eventually(HX < HX1)
              and Eventually(Always(IMP = IMP1))
              and Eventually(Always(HX = HX1))
```

The above query may be interpreted as follows: an (instantaneous) increase in the level of PRPP will not make the system stray from the predicted steady state, even if temporary variations of IMP and HX are allowed. Figure 10 shows how XSSYS responds to the new query.



```
? (not eventually(always(X1 > 7.5))) or steady_state() and eventually(always(x2 < 20!
0 and x8 < 10)).
The formula
    NOT(EVENTUALLY(ALWAYS((X1 > 7.5))))
     or STEADY_STATE() and EVENTUALLY(ALWAYS(((X2 < 200) and (X8 < 10))))
is false over the trace.
The first state where the formula is false is
at time 0.0.
? 
```

Figure 9. In Model 2 of [4] the level of PRPP is persistently raised to 1.7 its undisturbed steady state level. The XSSYS correctly answers the query whether the levels of IMP (variable X2) and the HX pool (variable X8) are still within acceptable bounds. Notice that they are not, thus, indicating a flaw in Model 2.



```
? always((not (x1 > 250))
         or (steady_state()
             and eventually(x2 > 100)
             and eventually(x13 < 10)
             and eventually(always(x2 > 99 and x2 < 101))
             and eventually(always(x13 > 9 and x13 < 11)))).
The formula
    ALWAYS(NOT((X1 > 250))
            or AND(STEADY_STATE(),
                   EVENTUALLY((X2 > 100)),
                   EVENTUALLY((X13 < 10)),
                   EVENTUALLY(ALWAYS((X2 > 99) and (X2 < 101))),
                   EVENTUALLY(ALWAYS((X13 > 9) and (X13 < 11)))))
is true over the trace.
? always(x1 < 6).
The formula
    ALWAYS(X1 < 6)
is false over the trace.
The first state where the formula is false is
at time 100.0.
?
```

Figure 10. The in silico trace of the "final model" from [4]. We arbitrarily increased the level of PRPP (variable X1) to more than 250 at time step 100. The XSSYS system correctly answers both queries. Because of numerical fluctuations we had to ask a less stringent question about the steady state value of IMP (variable X2) and HX (variable X13).

5 Concluding Remarks:

We have released a preliminary version of our software to the DARPA's Biospice community (`http://www.biospice.org`). Several interesting questions remain to be further explored. E.g. we are working toward integrating some Time/Frequency Analysis techniques into our tools, in order to better discriminate different traces, and to incrementally construct a better model without losing information (both numerically and symbolically). Just in order to get the interested reader to appreciate the challenges posed by this new branch of computational biology we list the following questions.

Reactions Models: We have primarily focused on a simple ODE model (Differential Algebraic Equations, DAE) and narrowed this even further to a model based on S- and XS-systems. Does this imply that we are accommodating a significant deviation from reality? How can a stochastic model representing small number of molecules interacting pair-wise and randomly be incorporated?

Numerical Issues: Many of the operations we propose need to be carefully analyzed from the numerical analyst point of view, in order to take into account error propagation issues.

Hybrid Systems: Certain interactions are purely discrete and after each such interaction, the system dynamics may change. Such a hybrid model implies that the underlying automaton must be modified for each such mode. How do these enhancements modify the basic symbolic model?

Spatial Models: The cellular interactions are highly specific to their spatial locations within the cell. How can these be modeled with richer abstractions of automata, e.g., cellular-automata? How can we account for dynamics due to changes to the cell volume? The time constants associated with the diffusion may vary from location to location; how can that be modeled?

State Space: A number of interacting cells can be modeled by product automata. In addition to the classical "state-explosion problem" we also need to pay attention to the variable structure due to i) Cell division, ii) Apoptosis and iii) Differentiation.

Communication: How do we model the communication among the cells mediated by the interactions between the extra-cellular factor and external receptor pairs?

Hierarchical Models: Finally, as we delve into more and more complex cellular processes, a clear understanding can only be obtained through modularized hierarchical models. What are the ideal hierarchical models? How do we model a population of cells with related statistics?

Acknowledgements:

We thank Mike Wigler, Misha Gromov, Ale Carbone, Tom Anantharaman, Vivek Mittal, Rob Lucito, Jack Schwartz, Roger Brockett, Sanjoy Mitter, Shankar Sastry, Sri Kumar, Mita Desai, David Harel, Amir Pnueli, Jane Hubbard, Charlie Cantor, Yuri Lazebnick and Harel Weinstein for offering many exciting ideas, useful contributions and ways to think about genomes, proteomes, pathways and cellular processes. Many of our close colleagues from NYU Bioinformatics group, Cold Spring Harbor Laboratory and Mt. Sinai School of Medicine have directly and indirectly contributed to this effort: Toto Paxia, Raoul Daruwala, Joey Zhou, Archi Rudra, Naomi Silver, Frank Park, Chris Wiggins, Violet Chang, Elizabeth Thomas, Ken Chang and Joe West. To all of them, we are grateful. The work reported in this paper was supported by grants from NSF's Qubic program, DARPA's Biospice program, HHMI biomedical support research grant, the US department of Energy, the US air force, National Institutes of Health and New York State Office of Science, Technology & Academic Research.

References

[1] U.S. Bhalla and R. Iyengar (1999). Emergent Properties of Networks of Biological Signaling Pathways, *Science* **238**, 381-387.

[2] M. Elowitz and S. Leibler (2000). A synthetic oscillatory network of transcriptional regulators, *Nature* **403**, 335-338.

[3] J. Keener, and J. Sneyd (1998). *Mathematical Physiology*, Springer-Verlag, New York.

[4] E. O. Voit (2000). *Computational Analysis of Biochemical Systems*, Cambridge.

[5] R. Curto, E. O. Voit, A. Sorribas and M. Cascante (1998). Mathematical models of purine metabolism in man, *Mathematical Biosciences* **151**, 1-49.

[6] R. Curto, E. O. Voit, A. Sorribas and M. Cascante (1998). Analysis of abnormalities in purine metabolism leading to gout and to neurological dysfunctions in man, *Biochemical Journal* **329**, 477-487.

[7] E. O. Voit (editor) (1991), *Canonical Nonlinear Modeling*, Van Nostrand Reinhold.

[8] E. A. Emerson (1990). Temporal and modal logic, in J van Leeuwen, ed., *Handbook of theoretical computer science*, Vol. B, chapter 16, pp. 997-1072. Elsevier, Amsterdam.

[9] C. C. Guet, M. B. Elowitz, W. Hsing and S. Leibler (2002). Combinatorial Synthesis of Genetic Networks, *Science* **286**, 1466-1470.

[10] B. Mishra and E.M. Clarke (1985). Hierarchical Verification of Asynchronous Circuits Using Temporal Logic, *Theoretical Computer Science* **38**, 269-291.

[11] M. Antoniotti, A. Policriti, N. Ugel and B. Mishra (2002). XS-systems: eXtented S-systems and Algebraic Differential Automata for Modeling Cellular Behavior, *Proceedings of the International Conference of High Performance Computing*, *HiPC'02*, Bangalore, India.

[12] M. A. Savageau (1993), Finding Multiple Roots of Nonlinear Algebraic Equations Using S-System Methodology, *Appl. Math. And Comput.* 55, 187-199.