Systems Biology and Automata

B. Mishra[1,2] and A. Policriti[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] Dept. of Mathematics and Computer Science, University of Udine, Udine, Italy
mishra@cs.nyu.edu, policrit@dimi.uniud.it

1 Modeling in Biology

Sydney Brenner wrote, “In late 1962, Francis Crick and I began a long series of conversations about the next steps to be taken in our research. ... I had come to believe that most of molecular biology had become inevitable and that, as I put it in a draft paper, ‘we must move on to other problems of biology which are new, mysterious and exciting. Broadly speaking, the fields which we should now enter are development and the nervous system.’ ... In a letter dated 5 June 1963 (see below), I wrote to Max [Perutz] and explained my views to him. ‘The great difficulty about these fields [development and the nervous system] is that the nature of the problem has not yet been clearly defined, and hence the right experimental approach is not known. There is a lot of talk about control mechanisms, and very little more than that.... It seems to me that, both in development and in the nervous system, one of the serious problems is our inability to define *unitary steps* of any given process. ... It is possible that the repressor/operator theory of Jacob and Monod will be the central clue, but there is not very much to suggest that this is so, at least in its simple form.’ ” [ref: Sydney Brenner, September 1987. From The Nematode Caenorhabditis elegans, WB Wood and the community of C elegans researchers, eds., Cold Spring Harbor Laboratory Press, Cold Spring Harbor, 1988.]

We continue to struggle with this problem of formulating a “unitary step” that precisely defines how a complex biological system makes a transition from one “state” or one “control mode” to another as well as the conditions under which such transitions are enabled. This is because we recognize that automata (either discrete or hybrid, terms defined subsequently in this paper), based on the formulation of these unitary steps, can elegantly model practically all biological control mechanisms, allow us to reason about such mechanisms in a modal logic systems with modes constructed over a next-time operator and can become the foundational framework for the emerging field of systems biology. In this paper, we will also see how these models can lead to more rigorous algorithmic analysis of large amounts of biological data, produced as (numerical) *traces* of *in vivo*, *in vitro* and *in silico* experiments—currently a central activity for many biologists and biochemists. Since modeling biological systems requires a careful consideration of both qualitative and quantitative aspects, our automata-based tools can effectively assist the working biologists to make predictions, generate falsifiable hypotheses and design well-focused experiments—activities in which the *time* dimension cannot be left out of consideration.

Thus, ultimately, the aim of our work is to elucidate the rôle played by *automata* in modeling biological systems and to investigate the potential of such tools when combined with more "classical" approaches used in the past to devise models and experiments in biology. Our discussion here is based primarily on our experience with a novel system that we introduced recently (called, XS-systems) and used it to implement algorithms and software tools (Simpathica). These conceptual tools have been integrated with prototype implementations, and are currently undergoing many interesting and growing sets of enhancements and optimizations—developments that can only be hinted at in this paper.

Both the theoretical as well the practical aspects of our approach have focused our attention on the very notion of automata and directed our research—evolving naturally—towards the study and use of *hybrid* automata: a family of tools originally created to capture both continuous and discrete behavior in a unique formalism and with combined computational techniques.

1.1 The mathematical tradition

If one considers the history of the applications of mathematics to biology, a rather natural observation consists in noticing that it has mostly centered around the design of conceptual tools for the mathematical interpretation of *biological experimental data.* The mathematicians involved in assisting biologists have primarily tackled the problem of designing—occasionally very complex and sophisticated—formal tools that hope to capture the nature of the biological phenomena. As examples, one may name several approaches based on systems of differential equations, statistical models, numerical problems and solutions, developed in a large variety of subfields of mathematics—all leading to important and deep cross-fertilization with some aspects of biology: genomics, bioinformatics and functional genomics.

By their very nature, if the above-mentioned formal tools, on one hand, turn out to be useful when correctly designed and specified, or on the other hand explain some intrinsic limits to their usefulness in terms of modalities of interaction with the user, then their impact on biology can be considered to be significant. Nonetheless, once the mathematical models are defined, the interaction with the biologists is in general rather rigid. Even in the case of models with some flexibility (e.g., some unknown parameters of the biological processes kept in their symbolic forms), this rigidity shifts to the inherent difficulty in determining the right values of parameters. In our opinion, this difficulty is mostly due to the fact that the *control* part of the model (defining its interaction with the external world) is not considered *ab initio.*

The field of software verification naturally suggests a different approach that has already addressed similar problems. The starting point in this field is the problem of modeling a *program* implementing a given system, to be verified in order to check if specific properties are provably true (e.g. the system is "bug"-free for some class of "bugs").

1.2 The logical tradition

While the mathematical tradition can be seen as oriented toward an *observational* approach to the design of tools, the more nascent approaches developed in the verification arena are almost entirely focused on the control mechanisms and aim to model the (relevant aspects of the) *internal logic* of a system. If one wishes to model and specify mechanisms by describing, "how things work," the natural solutions and techniques must be based on various classes of automata and rely on their history, deeply rooted in our logical tradition and formal tools. In our opinion it is not merely by chance that such a tradition is interleaved with the study of languages (natural and formal) and that resulting tools have become pervasive and actively studied in the fields of specification and verification.

As a natural byproduct of this historical intertwining of automata with languages, the possibility of querying the model (the automata) becomes available. But the level of expressivity of the query language sets the stage for many computational problems and defines certain inherent computational limits. On the positive side, however, are exactly the definition and implementation of interpreters for such languages that guarantee the interaction needed to verify the control mechanisms and to (re)design (e.g., "debug") the parts of the system under study.

All in all, while the use of automata does give an additional dimension to the purely observational approach, it also faces many problems in the process of combining different kinds of knowledge under the same formal structure. As described in the introduction, this problem naturally calls into play the concepts, formalisms and algorithms, already developed in the context of hybrid automata.

2 Automata and Biology

Automata are potentially the most natural tools for pondering about biological phenomena and there are many other different ways in which they enter the picture in biological research.

Perhaps, the most direct link, attempting to connect biology (genome as a language) and automata-theoretic study of language, dates back to the work by Searls ([14]) in which the central theme has been to view biological macromolecules, represented as string of symbols, and any collection of such macromolecules in terms of formal languages.

Another classical and important use of automata in biology is represented by the work on Hidden Markov Models (see [7]) which also represents a family of biological patterns (in DNA or proteins) in terms of a Markov model with certain state-transition and emission probabilities. These and other families of applications provide interesting instances of the observations (made earlier) on how difficult it is to determine the value of parameters involved in mathematically sophisticated models (see [12]).

In a manner, much similar to ours, Alur et al. [1] have been investigating the use of automata in modeling biologically complex systems, while trying to capture the *control* mechanisms implemented by the biochemical interactions. In particular, [1] demonstrates the necessity of modeling and analyzing both continuous and discrete behaviors, showing the suitability of a tool (hybrid automata) that we will also consider in our approach. In a sense, biological systems are more of a network than a single dynamical systems to be modeled monolithically.

2.1 Our experience

Our approach was motivated by the desire to design a tool, capable of using both data suitable for "classical" mathematical modeling activities as well as, whenever available, information on the control mechanisms underlying the system. In other words, our tools were designed to combine the two somewhat disparate traditions in a single system, *deducing* an automata structure from (1) experimental data as well as (2) trajectories derived from mathematical models.

Many choices we made can be further scrutinized and, possibly, modified in the future, but the overall idea can be illustrated through the following notions, central to our implementations.

S-systems The basic mathematical tool we used is represented by the so-called *S-systems* which are basically the ones presented in [15] 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 $\boldsymbol{X}$ to denote the vector $\langle X_1, \ldots, X_n, X_{n+1}, \ldots, X_{n+m} \rangle$ of variables and $\boldsymbol{d}$ ($\boldsymbol{a}$, $\boldsymbol{b}$,...) 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 $\boldsymbol{X} \upharpoonright U$ to denote the vector of variables of U, while $\boldsymbol{d} \upharpoonright U$ 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 ($\boldsymbol{X}(t_0)$), a final time (t_f), and a step (s).

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

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

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

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 developed can be straightforwardly adapted to variable time steps. Simulations of the behavior of an S-system can be automatically obtained by using the tool PLAS (see [15]). In fact, PLAS takes in as 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.

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 3 (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*

- *$V = \{\boldsymbol{v} = \langle v_1, \ldots, v_{n+m} \rangle \mid \exists\ tr \in Tr : \boldsymbol{v}$ is in $tr\} \subseteq D_1 \times \ldots \times D_{n+m}$ is the set of states;*

– $\Delta = \{(\boldsymbol{v}, \boldsymbol{w}) \mid \exists\ tr \in Tr : \boldsymbol{v}, \boldsymbol{w}$ are consecutive in $tr\}$ *is the transition relation;*
– $I = \{\boldsymbol{v} \mid \exists\ tr \in Tr : \boldsymbol{v}$ is initial in $tr\} \subseteq V$ *is the set of initial states;*
– $F = \{\boldsymbol{v} \mid \exists\ tr \in Tr : \boldsymbol{v}$ is final in $tr\} \subseteq V$ *is the set of final states.*

Automata can be equipped with labels on nodes and/or edges (see [11]). 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 [6]). 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.

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 [8]) (an English version of CTL) with a specialized set of predicate variables whose aim is to ease the formulation of queries on numerical quantities.

Example 1. A given automaton can satisfy 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 might not satisfy 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.

In 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 4 (Projection). *Let S be an S-system and U be a subset of the set of variables of S. Given a trace $tr = \langle \boldsymbol{a}_0, \ldots, \boldsymbol{a}_j \rangle$ of S the* projection over U *of tr is the sequence $tr \upharpoonright U = \langle \boldsymbol{a}_0 \upharpoonright U, \ldots, \boldsymbol{a}_j \upharpoonright U \rangle$. 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 approximation, which in our context means that the formulae satisfied by the automaton $\mathcal{A}(S, Tr \upharpoonright U)$ are not the same satisfied by the set of traces $Tr \upharpoonright U$.

In order to avoid the approximations resulting by the use of collapsing and projection, and in order to obtain a more powerful and flexible framework, the use of hybrid automata together with a reformulation of projection and collapsing is naturally introduced (see [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 5 (Hybrid automata). *A* hybrid automaton $H = (Z, V, \Delta, I, F, init, inv, flow, jump)$ *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* $e \in \Delta$ *is labeled by* $jump(e)$*, which is a constraint whose free variables are in* $Z \cup Z'$*.*

The usefulness of hybrid automata has been widely proved in the area of verification (see, e.g., [13]). In order to exploit the expressive power of hybrid automata their properties have been deeply studied (see [9]), 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 [10]) developed at Berkeley University, and Charon (see [1]) developed at the University of Pennsylvania.

In our context and in view of our previous observations, notice that S-system automata retain only quantitative information maintained as the values of the variables in instants corresponding to steps. The values at instants between two steps are lost. This situation becomes particularly dangerous when we apply a reduction operation such as collapsing. We circumvented 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 $\boldsymbol{X} = \langle X_1, \ldots, X_{n+m} \rangle$ and $\boldsymbol{v} = \langle v_1, \ldots, v_{n+m} \rangle$ we use the notation $\boldsymbol{X} = \boldsymbol{v}$ to denote the conjunction $X_1 = v_1 \wedge \ldots \wedge X_{n+m} = v_{n+m}$. The notation $\boldsymbol{v} \leq \boldsymbol{X} < \boldsymbol{w}$ has a similar meaning, while $\dot{\boldsymbol{X}} = (\boldsymbol{w} - \boldsymbol{v})/s$ 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 6 (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 $\boldsymbol{v} \in V$ let $init(\boldsymbol{v}) = \boldsymbol{X} = \boldsymbol{v}$;*
- *for each $\boldsymbol{v} \in V$ such that $(\boldsymbol{v}, \boldsymbol{w}) \in \Delta$ let*[1] *$inv(\boldsymbol{v}) = \boldsymbol{v} \leq \boldsymbol{X} < \boldsymbol{w}$;*
- *for each $\boldsymbol{v} \in V$ such that $(\boldsymbol{v}, \boldsymbol{w}) \in \Delta$ let $flow(\boldsymbol{v}) = \dot{\boldsymbol{X}} = (\boldsymbol{w} - \boldsymbol{v})/s$;*
- *for each $(\boldsymbol{v}, \boldsymbol{w}) \in \Delta$ let $jump((\boldsymbol{v}, \boldsymbol{w})) = \boldsymbol{X} = \boldsymbol{X}' = \boldsymbol{w}$.*

Notice from the above definition that being in a state $\boldsymbol{v}$ does not necessarily mean that the values of the variables are exactly $\boldsymbol{v}$: they can in fact assume values between $\boldsymbol{v}$ and $\boldsymbol{w}$. In particular, they grow linearly in this interval and when they reach $\boldsymbol{w}$ 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 [9]). 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 $\mathcal{A}(S, Tr)$.

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. This process assumes an additional relevance when we apply any collapsing technique to reduce the number of states.

3 Conclusion

We are left with the following questions: Have we arrived at the "right" automata definitions that naturally capture the biologist's intuitions? Are the "unitary steps" that emerge from these definitions at the right level from a biologist's viewpoint? Is it too detailed to obscure the central principles of biological control mechanisms? Is it too coarse, missing the effects of principles, central to biology.

As time enters the picture, further ingredients should probably be added: different scales and levels of *granularity*, a certain modularity and freedom in

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

the choice of the tools to be used for the quantitative analysis, the availability of a powerful environment to facilitate the interface with database data or to have automatic check on the formalisms employed.

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 *Lecture Notes in Computer Science*, pages 19–32, 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*, Lecture Notes in Computer Science, pages 209–229, 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, B. Mishra, C. Piazza, A. Policriti, and M. Simeoni. Modelling cellular behavior with hybrid automata: Bisimulation and collapsing. In C. Priami, editor, *International workshop on Computational Methods in Systems Biology, (CMSB'03)*, volume 2602 of *LNCS*, pages 57–74, Rovereto (ITALY), February 2003. Springer Verlag.
5. M. Antoniotti, B. Mishra, A. Policriti, and N. Ugel. Xs-systems: extended s-systems and algebraic differential automata for modeling cellular behavior. In V.K. Prasanna S. Sahni and U. Shukla, editors, *Proceedings of the International Conference on High Performance Computing, HiPC 2002*, volume 2552 of *LNCS*, pages 431–442, Bangalore (INDIA), December 2002. Springer Verlag.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* 1999.
7. R. Durbin, S.Eddy, A. Krough, and G. Mitchison. *Biological Sequence Analysis.* CUP, 1998.
8. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072, 1990.
9. 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.
10. 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.
11. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.
12. S. Kim, H. Li, E. R. Dougherty, N Cao, Y. Chen, M. Bittner, and E. B. Suh. Can Markov Chain Models Mimic Biological Regulation? *Journal of Biological Systems*, 10(4):337–357, 2002.
13. O. Müller and T. Stauner. Modelling and Verification using Linear Hybrid Automata. *Mathematical and Computer Modelling of Dynamical Systems*, 6(1):71–89, 2000.
14. David B. Searls. The computational linguistics of biological sequences. In Larry Hunter, editor, *Artificial Intelligence and Molecular Biology*, chapter 2, pages 47–120. AAAI Press, 1993.
15. E. O. Voit. *Computational Analysis of Biochemical Systems. A Pratical Guide for Biochemists and Molecular Biologists.* Cambridge University Press, 2000.