Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

EXECUTABLE BIOLOGY

Jasmin Fisher and Thomas A. Henzinger

School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL)
Lausanne, CH-1015, Switzerland

ABSTRACT

Computational modeling of biological systems is becoming increasingly common as scientists attempt to understand biological phenomena in their full complexity. Here we distinguish between two types of biological models —mathematical and computational— according to their different representations of biological phenomena and their diverse potential. We call the approach of constructing computational models of biological systems *Executable Biology*, as it focuses on the design of executable computer algorithms that mimic biological phenomena. We give an overview of the main modeling efforts in this direction, and discuss some of the new challenges that executable biology poses for computer science and biology. We argue that for executable biology to reach its full potential as a mainstream biological technique, formal and algorithmic approaches must be integrated into biological research, driving biology towards a more precise engineering discipline.

1 INTRODUCTION

Over the last decade, it has become apparent that biological research has reached a point where the accumulated data exceeds the human capacity to analyze it. This vast amount of information generated by DNA microarrays, genome sequencers, and other large-scale technologies, requires computer power to store, search, and integrate into a coherent whole. Systems biology, a new science combining biology, chemistry, physics, mathematics, electrical engineering, and computer science (to name a few), aims to integrate the data concerning individual genes and proteins into a complete picture; and to investigate the behavior and relationships of various elements in a particular biological system in order to understand how the system functions.

At the core of systems biology lies the construction of models describing biological systems. Over the years, biologists have been using diagrammatic models in order to summarize a mechanistic understanding of a set of observations. Despite the many benefits gained from using these models, as well as their useful simplicity, advances in systems biology have prompted biologists to harness computers in order to build and analyze ever larger models. The long-term vision is that large-scale models would revolutionize biological comprehension and eventually lead to drug discovery and the design of new therapies.

We distinguish between two types of such models. The first type includes models that use computer power in order to analyze mathematical relationships between quantities. The second is a new type of models, which resemble computer programs, thus creating a new field that we call *executable biology.* We explain the differences between these two types of models, explore some recent executable biological models, and suggest, from the executable biology point of view, some main challenges for computer science and biology in order for executable biology to realize its full potential.

2 MATHEMATICAL VERSUS COMPUTATIONAL MODELS

With the advent of ever more powerful and less expensive computers, computational modeling has become central to all science and engineering disciplines. The models fall into two categories.

- In the first category, computers are used to simulate *mathematical* models, and solve them for certain unknowns. A classical example are models based on differential equations, which are useful to capture many situations in the natural sciences and in engineering. These models were often developed before computation became feasible on a grand scale, but they are now profiting immensely from our increasing computational abilities.
- The second kind of model —still less known— is more directly *computational*, rather than mathematical, in that it presents a recipe —an algorithm— for an abstract execution engine to mimic a design or a natural phenomenon. Such

models are ideally suited to capture complicated causal chains of events. They have been used recently to model biochemical processes (Priami et al. 2001, Regev et al. 2001, Errampalli et al. 2004, Cardelli 2005) and the behavior and development of certain parts of biological systems (Kam et al. 2001, Kam et al. 2003, Efroni et al. 2003, Fisher et al. 2005, Fisher et al. 2006, Sadot et al. 2006).

Let us now draw a clearer distinction between mathematical and computational models. The two kinds of models differ in the languages in which they are specified: mathematical models are specified in mathematics (typically by equations); computational models are specified by computer programs (by code; often very high-level code written in a modeling language). Consequently, mathematical and computational models yield different kinds of insights.

Mathematical models can be simulated, and possibly solved. The basic entity of a mathematical model is the *transfer function*, which relates different numerical quantities to each other. A transfer function may be specified, for example, by a differential equation that relates an input and an output quantity. Complex mathematical models are constructed through the composition of transfer functions, yielding a network of interdependent quantities. If the constraints for individual transfer functions are relatively simple (e.g., linear differential equations), then mathematical models are amenable to mathematical *analysis*. In more complicated cases, they are still amenable to computational *simulation*.

Computational models can be executed. By contrast, the basic entity of computational models is the *state machine*, which relates different qualitative configurations ("states") to each other. A state machine may be specified, for example, by simple computer programs that define how, given certain events, one state is transformed into another. Complex computational models are constructed through the composition of state machines, yielding a "reactive system". The components of such a system represent biological entities —e.g., cells— which react to events in neighboring components by state transformations.

Such computational models are highly nonlinear and nondeterministic, and therefore generally not amenable to mathematical analysis. However, while for the simulation of a mathematical model an algorithm must be devised, a computational model prescribes the steps taken by an abstract machine, and is therefore inherently and immediately *executable*. Since the primary semantics of reactive models is computational, we speak of "execution" instead of simulation, and thus of *executable biology*. As computers are extremely efficient in executing instructions —much more so than in solving or simulating mathematical equations— the execution of very large computational models is possible.

Quantitative versus qualitative modeling of biology. In biology, mathematical models exist for many quantitative relationships between variables, such as molecule concentrations. Such models, however, are difficult to obtain if the number of interdependent variables grows, and if the relationships depend on qualitative events, such as a concentration reaching a threshold value. If precise quantitative relationships are unknown, if they involve many different variables, and if they change over time, depending on certain events, then computational models offer a natural and effective alternative. Indeed, since these premises are often satisfied, biology presents an almost perfect playground for computational models; much more so than, say, physics, where mathematical models dominate. As computational models are *qualitative*, they do not presuppose a precision that is not present in the experimental data; as they are *nondeterministic*, they allow many possible outcomes of a chain of events, and like nature, do not necessarily produce predetermined results.

Computational models can be analyzed by model checking. Computational models cannot only be executed, but they can also be used for testing and comparing hypotheses. Suppose that we have collected experimental data. A computational model represents a hypothesis about the mechanism that results in the data. An execution of the model can be used to check if a possible outcome of the mechanism conforms to the data. Due to nondeterminism, each repeated execution may yield a different possible outcome. Therefore it is impossible to check by executing the model if *all* possible outcomes conform to the data. This, however, can be done by a technique called *model checking* (Clarke et al. 1999). Model checking systematically analyzes all of the infinitely many possible outcomes of a computational model without executing them one by one.

If model checking tells us (1) that all possible outcomes of the computational model agree with the experimental data, and (2) that all experimental outcomes can be reproduced by the model, then the model represents a mechanism that explains the experimental data. If (2) is violated, then the hypothesis that the computational model captures a mechanism for explaining the data is found to be wrong. In this case, either the model must be enriched as to produce the additional outcomes that are present in the data, or completely revised. If (1) is violated, then the situation is more interesting. In this case, the mechanistic hypothesis represented by the model may be wrong, and one may attempt to restrict the model as to not produce outcomes that are not supported by the data. Alternatively, the experimental data may be incomplete and not exhibit some possible observations that would show up if more data were collected. Thus, in case (1), model checking can offer suggestions for additional, targeted experiments that would either confirm or invalidate the mechanistic hypothesis represented by the computational model.

Computational models can be compared by equivalence checking. A second technique, called *equivalence checking* (Milner 1971, Clarke et al. 1999), can be used to explain differences in the possible outcomes between two computational models, which represent two different mechanistic hypotheses. Suppose that we want to check which of two mechanisms better explains a given set of data points. We can use equivalence checking to find an outcome that is possible in one model, but not in the other. This distinguishing outcome can be used to suggest new experiments, whose results will rule out one of the two hypothetical mechanisms.

While model checking is a "filter" that selects from a set of possible mechanistic explanations the one that best explains the available data, equivalence checking is an "amplifier" that highlights the differences in the possible outcomes produced by a set of different mechanisms. Both model checking and equivalence checking are techniques that were developed for validating properties of computer systems (hardware and software). Yet viewed as algorithms for analyzing computational models, they find natural applications in validating mechanisms used to explain biological phenomena.

3 MODELS OF EXECUTABLE BIOLOGY

A large number of recent and ongoing efforts are putting the executable biology framework into practice. We now briefly review a few of these efforts.

Executable models for biological mechanisms. The resemblance between biological systems and reactive systems (Kam et al. 2001, Harel 2002) suggests the use of methods and tools designed for the construction and analysis of computational reactive systems to model biological systems. The first effort that followed this path was a modest model of T-cell activation (Kam et al. 2001). Using the visual language of Statecharts (Harel 1987), this model describes the various stages in which a T-cell can be over its life span; and the transitions between these different stages. The initial T-cell model was followed by a more extensive animated model of T-cell differentiation in the thymus (Efroni et al. 2003). In order to visualize this model, the method of *reactive animation* (Efroni et al. 2003, Efroni et al. 2005), in which a reactive system drives the display of an animation software, was developed. These studies were followed by ongoing efforts to model *Caenorhabditis elegans* development (Kam et al. 2003, Fisher et al. 2005, Fisher et al. 2006, Sadot et al. 2006), using both Statecharts and a more recent visual language called Live Sequence Charts (Damm and Harel 2001).

Executable models can be used to formulate biological mechanistic models, as well as to test their consistency with the experimental observations on which they are based. The consistency of such executable models can be tested by means of formal verification (i.e., model checking) (Fisher et al. 2004, Fisher et al. 2006). As part of the ongoing effort to model *C. elegans* vulval development, we have created a formal dynamic model of vulval fate specification based on the proposed mechanistic model of Sternberg and Horvitz (1989). This work (Fisher et al. 2005) has demonstrated that state-based mechanistic models are particularly well suited for capturing the level of understanding obtained using the tools and approaches common in the field of developmental genetics, and that creating such executable biological models is indeed beneficial. More recent work (Fisher et al. 2006) reports on a dynamic computational model of the more sophisticated understanding of vulval fate specification that we have today. There, we use model checking to test the consistency of the current conceptual model for vulval precursor cell fate specification with an extensive set of observed behaviors and experimental perturbations of the vulval system. The analysis of this model has predicted new genetic interactions that may further elucidate the mechanisms underlying precise pattern formation during animal development.

Process calculi for executing molecular processes. A different approach stresses the importance of concurrency and interaction between molecules as the main tools that drive the execution of biological processes. This approach uses process calculi, languages that have been developed to model networks of communicating processes (Milner 1999). A process is associated with a substance, and the interaction between substances is modeled as communication between the processes.

Initial work along this line suggested to use the *pi-calculus* (Milner 1999) as a modeling language for molecular interactions (Regev et al. 2001). These studies included the modeling of the RTK-MAPK signal transduction pathway and the construction of the biospy simulation environment. This work was later extended to use the stochastic pi-calculus (Priami 1995) in order to model a gene regulatory positive feedback loop (Priami et al. 2001). Many other studies have followed this direction, including experiments with the *ambient* *calculus* (Regev et al. 2004) and the *brane calculus* (Cardelli 2004), as well as applications of this methodology to modeling transcription factor activation and the glycolysis pathway (Curti et al. 2004), and RKIP inhibition of ERK (and its analysis using model checking) (Calder et al. 2005). A recent review by Cardelli (2005) discusses the process calculus approach in depth.

Hybrid models combining mathematical and executable models. Hybrid systems combine in one framework variables ranging over discrete and continuous domains (Henzinger 1996). The discrete variables are controlled by discrete transition changes that may depend on the values of continuous variables. The changes in continuous variables are according to differential equations (preferably linear) that depend on discrete control states (i.e., the combined value of all discrete variables). Hybrid systems aim to bridge the gap between mathematical models and executable models by combining the two. The discrete part of such models is the executable control mechanism that drives hybrid systems. However, hybrid systems require specialized software for the simulation and analysis of the continuous part. A major part of the work on hybrid systems is focused on the construction of algorithms that perform the required analysis. We now briefly describe a few studies on modeling and simulating biological systems using hybrid models.

In a series of studies, Ghosh and Tomlin have constructed a hybrid model of the Notch-Delta decision (Ghosh and Tomlin 2001, Ghosh et al. 2003). Their work distinguishes between the control structure regulating the production of the Notch and Delta proteins (represented by discrete control variables), and the part that follows the levels of these proteins (using continuous values and differential equations). Their model reproduces the Delta-Notch decision: cells that express Delta are surrounded by cells that express Notch. In addition, a detailed analysis of a two-cell model shows that the model's behavior matches a classical model using nonlinear differential equations (Ghosh and Tomlin 2001). In a separate analysis of the same model, Ghosh and Tomlin partition the initial states of the hybrid system according to the resulting Delta-Notch pattern (Ghosh et al. 2003).

Other applications of hybrid systems to model biological phenomena can be found for example in the work of Alur et al. (2001), which uses a language called Charon to execute hybrid models of a repressilator network; and in other work like (de Jong et al. 2003, Antoniotti et al. 2004, Hu et al. 2004, Lincoln and Tiwari 2004).

Additional approaches related to executable biology, which we do not discuss here due to lack of space, include Petri nets (e.g., Barjis and Barjis (1999), Dill et al. (2005)), and boolean networks (e.g., Shmulevitch et al. (2002), Li et al. (2004)).

4 CHALLENGES FOR COMPUTER SCIENCE

The use of computational models in biology poses new challenges for computer science. We now present four such technical challenges. Since these models were originally developed, within computer science, to model hardware and software systems, it is not surprising that certain adjustments are in order when modeling biological systems.

Identifying the appropriate model of concurrency. Complex reactive systems are built from simple ones by an operation called *parallel composition*, which puts several computational models together and executes all of them. The composition is called "parallel" because the individual models, so-called *components*, represent activities that go on concurrently, rather than one after the other. However, there are several different interpretations of what "concurrently" means in detail. At one extreme is the *synchronous* interpretation, where execution proceeds in a sequence of steps, and each component contributes one quantum of computation to each step. This interpretation is appropriate for many hardware systems, where the components represent individual gates of a circuit.

At the other extreme is the *asynchronous* interpretation, where each computation step represents the contribution of a single component, but different steps may represent different components. Implicit to the asynchronous interpretation is a scheduler, which chooses for each step the component that will contribute to that step. The scheduler is usually assumed to be "fair", meaning that it cannot neglect to choose any component forever. This interpretation is appropriate for software, where the components represent individual threads of a multi-threaded program. It is important to note that because of the choices made by the scheduler, the asynchronous model allows for many possible outcomes when executing a system, even if each individual component behaves deterministically.

For biological models, where the components may represent individual cells, neither the synchronous nor the asynchronous interpretation of concurrency seem adequate. This is because biological entities, such as cells, often proceed "roughly" in lock-step, but not completely so: some reactions may be a bit faster here, and a bit slower there; a bit faster now, and a bit slower later. But neither do biological reactions proceed at completely independent rates, as the asynchronous model would have it. The most appropriate way of modeling concurrency in biological systems seems to be a form of *bounded asynchrony* (Fisher et al. 2006), where the scheduler is constrained to be "bounded fair", meaning that it cannot neglect to choose any component for too long.

The choice of concurrency model (synchronous or asynchronous) has far-reaching implications on algorithms for model checking and equivalence checking. Any new concurrency model, such as bounded asynchrony, presents therefore the challenge of designing efficient methods for the analysis of the model.

Identifying the appropriate level of abstraction. A significant advantage of computational models is that different models can be used to describe the same system at different levels of detail, and that the various levels can be related formally. For example, the same software may be described in a high-level programming language or in low-level machine code. The high-level language is more suitable for a programmer to read and write; the low-level code is more suitable for a machine to execute; and there exist algorithms (so-called *compilers*) to automatically translate one into the other.

There are several natural levels of abstraction for describing biological systems using computational models. For example, the individual components may represent molecules, or at a less detailed level, they may represent cells. If mathematical models are considered as well, there are even more possibilities. For instance, variables may represent concentration levels of molecules, which represents a level between the modeling of individual molecules and the modeling of cells as atomic entities.

There are several challenges in this area. First, we need to identify the levels of abstraction that are most useful for carrying out manipulations and analyses of biological models, both by humans and by machines. In other words, we need to identify the equivalent of high-level programming languages and of low-level machine languages for the modeling of biological systems. Second, we need to formally relate the different levels of abstraction. At a minimum, this means that certain properties that are established on one level (such as the possibility of outcomes) must be preserved at the other level. Without such a theory of property preservation we cannot establish the equivalence of models at different levels. More ambitiously, we would like to have compilers that translate models at one level into equivalent models at the other level.

Identifying the appropriate roles for probability and time. Computational models are naturally nondeterministic, in the sense that repeated executions of a model may give different results. This is useful for modeling systems whose behavior cannot be predicted with mathematical precision, e.g., because slight differences in reaction times, molecule concentrations, environment conditions, etc. may cause significantly different outcomes. Nondeterminism, however, does not quantify the likelihood of any specific outcome; it simply specifies the set of possible results of a dynamic process. In order to quantify the probability of every possible outcome, stochastic models are needed. Computational models can be extended to Markov decision processes in order to represent stochastic behavior. However, model checking algorithms for probabilistic systems are considerably less efficient, and no satisfiable solutions are known for abstracting probabilistic systems. New results in these directions would considerably strengthen the appeal of computational models for biology (Rutten et al. 2004).

A second, related issue concerns the modeling of time. Many biological processes can be modeled as continuous-time Markov chains (Priami et al. 2001, Calder et al. 2005). The use of continuous time, like the use of probabilities, leads to appealingly natural models, but pays a price in terms of scalability of available analysis methods, and offers a degree of accuracy that is often not supported by the available data. We believe that the key lies in the development of nondeterministic discrete-time abstractions, like bounded asynchrony, which are able to capture and explain the phenomena of interest. These abstractions need to be related back to the underlying continuous, stochastic processes, and ideally, are automatically synthesized from them. It should be noted that discrete abstractions, while sacrificing numerical precision (a precision that may not be justifiable by the data), introduce complexity in the form of memory (i.e., states) into the system. They provide a qualitative characterization of quantitative models, and in the desirable case, they robustly reproduce the possible outcomes of a biological process (Alur et al. 2000).

Identifying useful building blocks. The design of large computational models is greatly aided by identifying a small set of basic building blocks that, possibly instantiated with different parameters, can be composed to build the complex systems of interest. Important examples of the building-block approach can be found in hardware design. Large circuits are designed from a very small number of different types of gates. At a higher level of abstraction, computer architectures are designed from standard components such as registers and arithmetic units. Such standard components can be collected in component libraries, so that designers can draw on them when needed. For the effective use of computational models in biology, we need such libraries of standard components, at different levels of abstraction (e.g., molecules, reactions and inhibitions, pathways, cells). In order to make these building blocks accessible to biologists, we further need a system description language that can be learned easily by experts in biology (rather than experts in computer science). Such a language should build on visual notations that are already familiar to biologists.

5 CHALLENGES FOR LIFE SCIENCES

Executable biology poses new challenges also for the life sciences. Three key challenges emerge as particularly relevant: the development of new methodologies to test experimentally dynamic scenarios proposed by executable models; the development of new techniques to collect quantitative data through laboratory experimentation; and most importantly, the shift of biology to an engineering science, where students are educated to use formal approaches to biology.

Developing experimental methodologies to test system dynamics. Dynamic models can represent phenomena of importance to biology which static diagrammatic models cannot represent, such as time and concurrency. By adding time-dynamic aspects to biological models, the influence of time on system behavior becomes an important component to validate experimentally. We expect that the increasing usefulness of dynamic models in biology will also lead to an increasing need to validate dynamic scenarios. An example of such dynamic scenarios was described in a recent executable model representing aspects of cell fate specification (Fisher et al. 2005), where we have suggested the existence of a race between two signaling pathways, which determines cell fates during *C. elegans* vulval development.

Unfortunately, in practice, time is experimentally daunting. Techniques that enable direct measurement of pathway activities or protein levels, at a single-cell resolution, are not yet a common practice. The challenge therefore is two-fold: first, to develop techniques that allow such measurements; and second, to allow these measurements to be taken continuously from the same element, providing a dynamic view of the studied processes.

Developing new techniques to collect quantitative data. A major effort in recent years is the construction of quantitative mathematical models that explain biological phenomena. A variety of biological systems have been modeled in this way, including the networks controlling bacterial chemotaxis (Bray et al. 1998, Alon et al. 1999), development patterning in *Drosophila* (Burstein 1995, Marnellos and Mjolsness 1998), and infection of *E. coli* by lambda phage (McAdams and Shapiro 1995). From an executable biology point of view, we are interested in such models for two reasons. First, quantitative parts can be incorporated into computational executable models to establish hybrid models (Henzinger 1996), adding another dimension of accuracy to computational models. Second, computational models often represent high-level abstractions of detailed mathematical models. Hence we would like to use mathematical models to infer abstract computational models directly from data, without the laborious manual process of hand-coding computational models.

Mathematical models by their very nature handle quantities and require large amounts of detailed data (i.e., concentrations of molecules, individual binding constants, reaction rates, and probability distributions). Such data is difficult to obtain, particularly in relevant biological contexts (in vivo, in single cells rather than whole populations, etc.) (Gilman and Arkin 2002). Experimental data, such as fluorescence levels of tagged proteins or immunoblots, is usually limited to unit-less ratios of expression levels that are only proportional to the actual protein concentrations. Not having direct measures can significantly hinder or complicate finding parameter values (Brown and Sethna 2003). In many studies (see Kaern et al. (2005) and Sprinzak and Elowitz (2005) for reviews), the linear relation between the concentrations of fluorescent proteins and their measured fluorescence intensities has been used to measure protein levels in living cells, though only in relative terms and not in absolute numbers. Hence, improvements in the existing experimental methodologies to enable direct quantitative measurements are essential. One of the recent efforts to follow this path is the development of a technique that converts observed fluorescence intensities into numbers of molecules (Rosenfeld et al. 2006).

Biology as an engineering science. Modeling of biological systems is intended to facilitate our understanding of biological phenomena in their full complexity. In order for executable biology to realize its potential as a mainstream biological research technique, the method should be used extensively by biologists. Thus, we have to suggest forms to represent models and data that are natural, formal, and can become standard. Stressing user-friendliness, flexibility, visuality, and accessibility are therefore a critical necessity. Indeed, one of the challenges facing computer scientists today is the necessity to build software tools that are more accessible to biologists. In particular, we hope that such tools will enable the integration of executable biology into everyday biological methodology, bringing to a much wider audience the advantages we discussed above.

At the same time, a major challenge for biologists is to apply more formal approaches in biology, and develop a formal and standardized representation of biological knowledge and data (Gilman and Arkin 2002, Lazebnik 2002). In order to use executable biology tools and build executable models, biologists will need to adapt a more formal and algorithmic view of systems. Even with the most user-friendly tools, the construction of executable models is still similar to the engineering of hardware and software systems, which require the user to think about state machines and recipes of computation. These notions, to which computer scientists are introduced at an early stage, are crucial for the successful application of these techniques on a large scale by biologists. In accordance with the suggestions to formalize biology, part of the formal education of future biologists should include engineering disciplines that teach an algorithmic and formal way of thinking. Notions such as state transitions, concurrency, and abstraction should become familiar to every biologist.

6 CONCLUDING REMARKS

Modeling offers great advantages in integrating and evaluating information, providing strong predictions, and focusing experimental directions. The long-term vision is that system-level models would revolutionize biological comprehension and eventually lead to drug discovery and the design of new therapies. Although the way to achieve this ambitious goal is still long, it holds the promise of changing the face of biology and medicine. Moreover, the magnitude of the systems to be handled will challenge computer science to develop tools and techniques that handle ever more complex systems. It is our strong belief that abstraction and high-level reasoning will play a large role in the realization of this goal, both in biology and computer science.

Executable biology is a pioneering and powerful approach in this direction. We believe that much research is still needed in order to create user-friendly formalisms to be handled by biologists, and that these efforts require the tight collaboration between biologists and computer scientists. The inherent differences between mathematical and executable models and the difficulty to obtain precise data make both styles indispensable. We therefore expect that executable biology will take its place in the mainstream of biological research.

REFERENCES

Alon, U., M.G. Surette, N. Barkai, and S. Leibler. 1999. Robustness in bacterial chemotaxis. *Nature* 397:168–71.

Alur, R., C. Belta, F. Ivancic, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug. 2001. Hybrid modeling and simulation of biomolecular networks. In *HSCC*, LNCS 2034:19–32. Springer.

Alur, R., T.A. Henzinger, G. Lafferriere, and G.J. Pappas. 2000. Discrete abstractions of hybrid systems. *Proc. IEEE* 88:971–984.

Antoniotti, M., B. Mishra, C. Piazza, A. Policriti, and M. Simeoni. 2004. Taming the complexity of biochemical models through bisimulation and collapsing: Theory and practice. *Theor. Comp. Sci.* 325:45–67.

Barjis, J., and I. Barjis. 1999. Formalization of the protein production by means of petri nets. In *Proc. IIS*, 4–9. IEEE.

Bray, D., M.D. Levin, and C.J. Morton-Firth. 1998. Receptor clustering as a cellular mechanism to control sensitivity. *Nature* 393:85–88.

Brown, K.S., and J.P. Sethna. 2003. Statistical mechanical approaches to models with many poorly known parameters. *Phys. Rev. E.* 68.

Burstein, Z. 1995. A network of model of developmental gene hierarchy. *J. Theor. Bio.* 174:1–11.

Calder, M., V. Vyshemirsky, D. Gilbert, and R. Orton. 2005. Analysis of signaling pathways using the PRISM model checker. In *CMSB*, 179–190.

Cardelli, L. 2004. Brane calculi. In *CMSB*, LNCS 3082:257–278. Springer.

Cardelli, L. 2005. Abstract machines of systems biology. *Trans. Comp. Sys. Bio.* 3:145–168.

Clarke, E.M., O. Grumberg, and D. Peled. 1999. *Model Checking.* MIT Press.

Curti, M., P. Degano, C. Priami, and C. Baldari. 2004. Modeling biochemical pathways through enhanced pi-calculus. *Theor. Comp. Sci.* 325: 111–140.

Damm, W., and D. Harel. 2001. LSCs: Breathing life into message sequence charts. *Form. Meth. Sys. Des.* 19:45–80.

de Jong, H., J.-L. Gouzé, C. Hernandez, M. Page, T. Sari, and J. Geiselmann. 2003. Hybrid modeling and simulation of genetic regulatory networks: A qualitative approach. In *HSCC*, LNCS 2623:267–282. Springer.

Dill, D., M. Knapp, P. Gage, C. Talcott, K. Laderoute, and P. Lincoln. 2005. The pathalyzer: A tool for analysis of signal transduction pathways. In *Recomb WSB.*

Efroni, S., D. Harel, and I.R. Cohen. 2003. Toward rigorous comprehension of biological complexity: Modeling, execution, and visualization of thymic T-cell maturation. *Genome Res.* 13:2485–97.

Efroni, S., D. Harel, and I.R. Cohen. 2005. Reactive animation: Realistic modeling of complex dynamic systems. *IEEE Computer* 38:38–47.

Errampalli, D.D., C. Priami, and P. Quaglia. 2004. A formal language for computational systems biology. *Omics* 8:370–80.

Fisher, J., D. Harel, E.J. Hubbard, N. Piterman, M.J. Stern, and N. Swerdlin. 2004. Combining state-based and scenario-based approaches in modeling biological systems. In *CMSB*, LNCS 3082:236–241. Springer.

Fisher, J., N. Piterman, A. Hajnal, and T.A. Henzinger. 2006. Predictive modeling of *C. elegans* vulval development. Submitted.

Fisher, J., N. Piterman, and T.A. Henzinger. 2006. Bounded asynchrony: A notion of concurrency tailored for modeling cell-cell interactions. Submitted.

Fisher, J., N. Piterman, E.J. Hubbard, M.J. Stern, and D. Harel. 2005. Computational insights into *Caenorhab-*

ditis elegans vulval development. *Proc. Natl. Acad. Sci. USA* 102:1951–6.

Ghosh, R., A. Tiwari, and C. Tomlin. 2003. Automated symbolic reachability analysis with application to delta-notch signaling automata. In *HSCC*, LNCS 2623:233–248. Springer.

Ghosh, R., and C. Tomlin. 2001. Lateral inhibition through delta-notch signaling: A piecewise affine hybrid model. In *HSCC*, LNCS 2034:232–246. Springer.

Gilman, A., and A.P. Arkin. 2002. Gentic code: Representations and dynamical models of genetic components and networks. *Ann. Rev. Genomics Hum. Genet.* 3:341–69.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Sci. Comp. Prog.* 8:231–274.

Harel, D. 2002. A grand challenge for computing: Towards full reactive modeling of a multi-cellular animal. *Bull. EATCS* 81:226–235.

Henzinger, T.A. 1996. The theory of hybrid automata. In *LICS*, 278–292. IEEE.

Hu, J., W.C. Wu, and S. Sastry. 2004. Modeling subtilin production in bacillus subtilis using stochastic hybrid systems. In *HSCC*, LNCS 2993:417–431. Springer.

Kaern, M., T.C. Elston, W.J. Blake, and J.J. Collins. 2005. Stochasticity in gene expression: From theories to phenotypes. *Nat. Rev. Genet.* 6:451–64.

Kam, N., D. Harel, and I.R. Cohen. 2001. The immune system as a reactive system: Modeling T-cell activation with Statecharts. In *VLFM*, 15–22. IEEE.

Kam, N., D. Harel, H. Kugler, R. Marelly, A. Pnueli, E.J.A. Hubbard, and M.J. Stern. 2003. Formal modeling of *C. elegans* development: A scenario-based approach. In *CMSB*, LNCS 2602:4–20. Springer.

Lazebnik, Y. 2002. Can a biologist fix a radio? *Cancer Cell* 2: 179–82.

Li, F., T. Long, Y. Lu, Q. Ouyand, and C. Tang. 2004. The yeast cell-cycle network is robustly designed. *Proc. Natl. Acad. Sci. USA* 101:4781–6.

Lincoln, P., and A. Tiwari. 2004. Symbolic systems biology: Hybrid modeling and analysis of biological networks. In *HSCC*, LNCS 2993:417–431. Springer.

Marnellos, G., and E. Mjolsness. 1998. A gene network approach to modeling early neurogenesis in *Drosophila*. In *Pac. Symp. Biocomp.*, 30–41.

McAdams, H.H., and L. Shapiro. 1995. Circuit simulation of genetic networks. *Science* 269:650–6.

Milner, R. 1971. An algebraic definition of simulation between programs. In *IJCAI*, 481–489. British Computer Society.

Milner, R. 1999. *Communicating and Mobile Systems: The pi-Calculus.* Cambridge University Press.

Priami, C. 1995. The stochastic pi-calculus. *Comp. J.* 38:578–589.

Priami, C., A. Regev, E.Y. Shapiro, and W. Silverman. 2001. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Info. Proc. Lett.* 80:25–31.

Regev, A., E.M. Panina, W. Silverman, L. Cardelli, and E.Y. Shapiro. 2004. Bioambients: An abstraction for biological compartments. *Theor. Comp. Sci.* 325:141–167.

Regev, A., W. Silverman, and E.Y. Shapiro. 2001. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pac. Symp. Biocomp.*, 459–70.

Rosenfeld, N., T.J. Perkins, U. Alon, M.B. Elowitz, and P.S. Swain. 2006. A fluctuation method to quantify *in vivo* fluorescence data. *Biophys. J.* 91: 645–54.

Rutten, J., M. Kwiatkowska, G. Norman, and D. Parker. 2004. *Mathematical techniques for analyzing concurrent and probabilistic systems*, Volume 23 of *CRM Monograph Series*. American Mathematical Society.

Sadot, A., J. Fisher, D. Barak, Y. Admanit, M.J. Stern, E.J.A. Hubbard, and D. Harel. 2006. Towards verified biological models. Submitted.

Shmulevitch, I., R. Dougherty, S. Kim, and W. Zhang. 2002. Probabilistic boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics* 18: 261–74.

Sprinzak, D., and M.B. Elowitz. 2005. Reconstruction of genetic circuits. *Nature* 438:443–8.

Sternberg, P.W., and H.R. Horvitz. 1989. The combined action of two intercellular signaling pathways specifies three cell fates during vulval induction in *C. elegans*. *Cell* 58: 679–93.

ACKNOWLEDGMENTS

We apologize to colleagues whose work was not reviewed due to lack of space. Our research is supported in part by the Swiss National Science Foundation under grant 205321-111840. We thank Nir Piterman for valuable comments.

AUTHOR BIOGRAPHIES

Jasmin FISHER is a postdoc in the School of Computer and Communication Sciences at EPFL in Lausanne, Switzerland. She holds a Ph.D. degree in Neuroimmunology from the the Weizmann Institute of Science and was previously a postdoctoral fellow in the department of Computer Science also at the Weizmann Institute. Her e-mail address is `<jasmin.fisher@epfl.ch>`, and her web page is `<mtc.epfl.ch/~fisher>`.

Thomas A. HENZINGER is a Professor of Computer and Communication Sciences at EPFL in Lausanne, Switzerland, and Adjunct Professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He is a member of Academia Europaea and of the German National Academy of Sciences (Leopoldina), and a Fellow of the IEEE. His e-mail address is `<tah@epfl.ch>`, and his web page is `<mtc.epfl.ch/~tah>`.