Dynamic Behavior of Cell Signaling Networks – Model Design and Analysis Automation

Natasa Miskov-Zivanov¹,², Diana Marculescu², and James R. Faeder¹
¹Department of Computational and Systems Biology, School of Medicine, University of Pittsburgh
²Department of Electrical and Computer Engineering, Carnegie Mellon University

E-mail: ¹{nam66,faeder}@pitt.edu, ²dianam@cmu.edu

ABSTRACT
Recent work has presented logical models and showed the benefits of applying logical approaches to studying the dynamics of biological networks. In this work, we develop a methodology for automating the design of such models by utilizing methods and algorithms from the field of electronic design automation. We anticipate that automated discrete model development will greatly improve the efficiency of qualitative analysis of biological networks.

Categories and Subject Descriptors
B.6.3 [Logic design]: Design Aids; I.6 Simulation and modeling; J.3 Life and medical sciences.

General Terms

Keywords
Biological networks, Logical modeling, Design automation techniques, Hardware description language, Hardware-based emulation, FPGA.

1. INTRODUCTION
Anticipation of a biological system's response to a drug or a pathogen, or prediction of tumor initiation and its temporal and spatial progression is of great interest in medicine. As cells constantly sense their environment and respond to it (Figure 1), predicting how a cell will respond to a particular stimulus is a grand challenge in cell biology.

Therefore, it is becoming increasingly important to develop models of these complex biological systems, and more specifically, models of regulatory networks that control crucial biological processes. Different computational approaches have been proposed for modeling and studying biological systems. However, the complexity of these models increases rapidly with the size of the network.

Modeling biological networks using Boolean variables and logical operators has gained increasing attention in recent years [1-6]. This approach allows for more coarse-grained yet accurate studies of the dynamics of cell signaling network and its behavior (state transitions) from initial state to steady state (or steady cycle). However, the lack of automated tools for constructing these models limits their application to large-scale networks. Even for smaller networks, the current manual approach to model development is time-consuming, because it requires a trial and error process to work out the regulatory logic (e.g., necessary vs. sufficient, AND vs. OR, etc.). We propose an automated approach to construct biological circuit models that relies on tools from the well-established field of electronic design automation (EDA).

Once the model is developed using our methodology, it is straightforward to use the model for emulating biological network behavior into Field Programmable Gate Array (FPGA) platforms. FPGAs are ideally suited to implement highly parallel architectures and they allow for efficient and accurate analysis of complex signaling and regulatory networks. The hardware-based emulation framework that has been described in [7, 8] enables orders of magnitude speedup when compared to classic simulation in silicon.

The contributions of presented work include the following:
• Allows for the development of complex logical models in which network elements may have multiple discrete values and which would otherwise be difficult (if not impossible) to develop manually;
• Allows for efficient simulation of logical models using a hardware-based emulation approach, with orders of magnitude speedup when compared to existing software-based approaches.

The rest of this paper is organized as follows. In Section 2, we review related work on developing and analyzing models of cell signaling networks. In Section 3, we present preliminaries of developing logical models for cell signaling networks. Our methodology for designing models is described in Section 4 and approaches for studying models are described in Section 5. We conclude our work with Section 6.

2. RELATED WORK
A number of methods and approaches have been used for modeling complex networks of biochemical interactions. The methods range from master equations and the Monte-Carlo method [9], ordinary differential equations (ODE) [10, 11], reaction rule-based models [12, 13], all the way to Boolean networks [1, 5, 14, 15].

Differential equations, which capture the underlying reaction kinetics in terms of rates and concentrations, are perhaps the most commonly used approach to modeling biochemical pathways and networks. Often, the information about reaction rates and parameter values is hard to determine from the existing knowledge about the system and needs to be estimated or guessed. Rule-based modeling formalisms and associated

Figure 1. The cell as a "black-box."
Models of biological networks are usually developed manually, through literature overview, discussions with experts and by studying experimental data. There exist a number of tools to write or draw these models using formalisms mentioned above [12, 14, 17]. Rule-based modeling represents a step towards automating development of models of reaction networks. Protein regulatory networks can be assembled using 'reverse engineering' methods such as Bayesian network analysis [18] or inferred systems of differential equations. In [19], multiple linear regressions were used to develop a first order differential equation model of a gene and protein regulatory network. However, the number of reactions and biochemical species implied by rules can become enormously large, making it difficult to construct, simulate and analyze conventional models derived from rules.

An effort to automate the development of Boolean models of regulatory networks is presented in [4]. The authors in [4] present the methodology in which a tool, called CellNetOptimizer, is used to assemble Boolean models from protein signaling networks and raw data, using data management application, DataRail [20]. All possible Boolean models that are compatible with the signaling network are collected, and then applying a fitness test to each model reduces this set. The main drawback is that the complexity of this approach increases rapidly with the size of the network and the number of connections in the network.

3. LOGICAL MODELING OF BIOLOGICAL NETWORKS

In Figure 3(top left), we present an interaction map of a small system with four elements (e.g., proteins). Each element has an associated discrete variable. One can consider these variables representing different aspects of protein activity and function, for example: inactive or absent (0 value) vs. active or present with different levels of activity or concentration (e.g., low, medium, high can be represented using values 1, 2, 3, etc.). As it can be seen in Figure 3(top left), interactions between elements are represented using two types of arrows, pointed error (positive regulation, activation, $\rightarrow$) and blunt arrow (negative regulation, inhibition, $\leftarrow$).

A model also includes a set of logic rules that more precisely define interactions, including information about whether a regulator is necessary or sufficient. In Figure 3(top left) we show rules for $E_2$, $E_3$, and $E_4$. $E_1$ does not have incoming arrows and thus its value cannot change. The "*" sign in rules represents the next-state value for a variable. It can be seen from these rules that $E_1$ and $E_4$ are both necessary for activation of $E_2$, while $E_4$ is activated only when $E_1$ is present and $E_3$ is not.

Logistical models are more abstract than reaction network models, in which reaction rates and molecule counts are represented precisely. At the same time, logical models allow for studying the dynamics of a larger system, capturing external ligand and receptor interactions, internal cell signaling, and gene transcriptions inside nucleus, all within a single model. Finally, logical models can capture system behavior (state transitions) from the initial state until reaching a steady state or limit cycle.

We show in Figure 3 state transition graphs (STGs) for the example system when two different update schemes are used, synchronous and asynchronous. Synchronous update scheme assumes that all variables are updated simultaneously using update rules. State transitions are deterministic and for each state there is exactly one possible next state. State transitions resulting from synchronous update scheme applied on the example system are shown in Figure 3(top right). States in the figure represent values of vector $(E_1, E_2, E_3, E_4)$. Asynchronous update scheme assumes that, in each update round, all variables (that is, those variables that are not fixed and have rules associated with them) are updated according to their logic rules, one by one, and the order of variable updates is randomly chosen in each round. The STGs at the bottom of Figure 3 represent state transitions that can occur in the example system when an asynchronous update scheme is used.

It is also possible to combine synchronous and asynchronous update schemes by grouping rules and ordering the groups, such that groups of rules are updated in order of their rank, while a random asynchronous scheme is applied to all rules within the same rank [14]. This combined update scheme is well-suited to biological signaling networks because it is able to take into account known differences in the timescale of different processes and interactions while averaging over timing differences that are not known.

4. MODEL DESIGN METHODOLOGY

In Figure 2, we show the main steps of our approach for designing and studying discrete models of cell signaling.
networks. In the following, we detail each step of our model design methodology using an example model of cell signaling network shown in Figure 4. As it can be seen from the figure, the interaction map of the network (Figure 4(left)) includes several layers of signaling:

- Receptor layer: ligands (L1, L2) bind to receptors (R1, R2) outside of the cell to initiate cell signaling;
- Cytoplasm layer: internal cell molecules (E1-E12) propagate signaling through events such as phosphorylation and binding, eventually activating transcription factors (TF1, TF2, TF3);
- Nucleus layer: transcription factors move to nucleus where they bind genes and initiate gene transcription (G1, G2).

4.1 Selection of model elements and their influence sets

The first step in model design is the identification of elements and their regulators (influence sets) through extensive literature survey and consultation with experts. Since the list of important components and interactions is rapidly evolving, this is often an iterative process. Moreover, as we outline in Figure 2, the overall model design process is expected to be iterative, with model studies providing new insights into the system that in turn allow for developing a better model.

4.2 Number of element discrete values

For each element of the network to be modeled, the number of discrete values representing different levels of element activity must be defined. Our modeling methodology is not limited to binary variables, but also allows for including multiple discrete values.

It is often the case in cell signaling that different concentration of ligand, or its affinity to binding a receptor result in different cell responses. Therefore, in our example model in Figure 4, we assume that receptor R1 can have three different levels of activity, 0, 1, and 2. In order to model this within Boolean framework, we encode variable R1 using two Boolean variables, high (R1_HI) and low (R1_LO):

\[ R1 = 0 \text{ (R1 HI=0, R1 LO=0)} \]
\[ R1 = 1 \text{ (R1 HI=0, R1 LO=1)} \]

The fourth case (R1_HI=1, R1_LO=1) can be treated as a "don't care." Rules for updating variables of the model are shown on the right of the interaction map in Figure 4.

After initially assigning a single Boolean variable to all elements (except receptor R1) in the example model, we observed oscillations in the negative loop between elements E9 and E11. In Figure 5(a), we present simulation results for two scenarios: R1 = 1 and R1 = 2. The curves in Figure 5(a) show how element E11 changes from initial state until the final simulated round, averaged across 1000 simulation runs using the asynchronous update scheme. Figure 5(a)(bottom) outlines several individual simulation trajectories for element E11, emphasizing the effect of random asynchronous update scheme.

The oscillations observed in E11 for R1=2 scenario occur because E11 is assumed to be necessary for E6 activation, which then activates E9, which in turn activates E10. Once E10 is activated, it inhibits E11, preventing further activation of E6 and activation of E9. Oscillations can occur in these networks as a result of negative feedback loop existing within the network. However, oscillations can also be an artifact of modeling. In case of logical modeling, oscillations can occur in negative feedback loops with odd number of inhibitions within the loop, like in the loop E6→E7→E8→E9→E10→E1.

In order to avoid oscillations in the logical model, one can apply several approaches:

1. Control negative feedback loop with an element that is outside of the loop;
2. Model elements of the loop with multiple (more than two) levels of activity;
3. Change rules from necessary to sufficient (AND to OR rules) or vice versa.

The approach number 2 can be used to model damped oscillations. We will discuss the choice of update rules (approach 3) in the following sub-section. Here we discuss in more detail approach 1. We modeled E3, which is outside of the loop, with three levels of activity (0, 1, and 2), such that the control of the loop is different between the two scenarios. When E3 is at the lower level (E3=1, scenario R1 = 1), the inhibition of E11 by E10 can overcome its activation by E3. When E3 is at the higher level (E3=2, scenario R1 = 2), activation is stronger, and E11 is active as long as E3=2. The effect of this change in levels of E3 on average value of E11 is shown in Figure 5(b)(top). The new rule that is used for E11 and that captures different levels of E3 is highlighted in Figure 5(b)(top). As can be seen from this figure, results for scenario R1=1 remain similar to the case where E3 has only two levels of activity, while scenario R1=2 does not lead to oscillations anymore.

4.3 Update rule derivation

Once model elements, their influence sets, and levels of activity are defined, the next step is to derive element update rules. We created a framework to automate rule derivation from influence tables.

Influence tables are used to present the value of an element as a result of different combinations of its activators. Influence tables can be derived either from existing knowledge, or in an automated fashion from experimental time series data [19]. In the example models we are presenting here, we created the influence tables manually, from existing literature and via discussion with experts. We are currently working on developing a more formal approach for deriving these tables. In our framework, we write these tables to an input file and then use
logic synthesis tools and algorithms to automate derivation of logic rules. Depending on the complexity of these tables, we use K-maps, Quine-McCluskey or Espresso [21, 22].

Since element E11 from the model in Figure 4 has only two levels of activity (0 and 1), and is regulated by the three-level E3 (after model modifications described above), and by the two-level E10, we show in Figure 5(c) how this is translated into Boolean variables. The bottom two influence tables in Figure 5(c) are same as K-maps. Although this example is simple from the perspective of logic circuits, it is important to make sure that we accurately implement biological relationships within logic rules. Furthermore, the framework that we created can handle significantly larger examples than the ones shown in Figure 5. When combinations of variables are such that they do not represent realistic biological events, we implement those cases in influence tables as "don't cares." As shown in Figure 5(c), a symbol "x" in the influence tables is included in cases where both E3_LO and E3_HI are equal 1, due to the fact that this combination represents a case that is not defined by modeling E3 with three levels of activity. However, from the logic synthesis perspective, these cases can help minimize logic functions.

Figure 5(d) presents the simplest examples of influence tables, where an element has two regulators and all three of them can have only two levels of activity, 0 and 1. Element E9 and its two regulators, E1 and E8, are highlighted in Figure 5(d), together with examples of deriving an update rule for E9 from its influence table. Experiments can suggest several different regulatory relationships: (i) both elements E1 and E8 are necessary for activation of E9 (top table), (ii) one of the regulators is necessary and sufficient for activation, the other one, when present, can increase the level of element E9 (middle table) or, (iii) either E1 or E8 is sufficient for activation of E9 (bottom table). Conditions (i) and (iii) can be accurately captured with logic rules. However, condition (ii) requires including multiple levels of activity when defining E9. As can be seen from the middle table in Figure 5(d), the logical rule can only capture activation of E9 by regulator E8 (necessary and sufficient regulator), but not regulation by E1. It can be seen from the examples described so far that including multiple levels of activity (more than two) provides much more flexibility in modeling element relationships and system characteristics, while still being simpler to simulate than, for example, ODEs.

The examples in Figure 6 present influence tables in which both effect and its regulators have three levels of activity (0, 1, and 2). Figure 6(a) shows derivation of a rule for signaling of two receptors, R1 and R2, which can both be activated by binding two different types of ligands (L1_1 or L1_2 for R1 and L2_1 or L2_2 for R2). In our models of biological signaling networks we often have elements with larger number of regulators, where both effect and its regulators have at least three levels of activity. In such cases, our framework uses a more efficient method, Espresso [22], to obtain logic rules. In Figure 6(b), we show an example of modeling gene regulation with two transcription factors and one inhibitor. The table shown in Figure 6(b) is slightly more complex than tables in previous examples. Thus, creating rules for the whole cell signaling network from a large number of such tables becomes impractical, and emphasizes the importance of rule design automation. Furthermore, in examples in Figure 6(a)(b), we also include statements that use integer values and capture the same relationships as influence tables. Such statements can be used in hardware description languages as an alternative to developing influence tables, something we are currently working on.

Figure 6(c) presents another example of gene regulation with three transcription factors and one inhibitor. As can be seen, increasing number of regulators by only one (going from example in Figure 6(b) to Figure 6(c)), the influence table size increases significantly. We also show in this example that, as is often the case in modeling biological systems, for some table entries there may not be an existing experimental observation to support entered values. In such cases, model design automation allows for developing alternative rules and testing which of the rules help us recapitulate best the behavior of the system. In the following section, we describe the framework that we have developed to further automate model analysis, and especially to speed up the analysis of models that have alternative rules.
We have applied our model design methodology to two cell signaling networks. One network includes circuitry that controls differentiation of T cells (lymphocytes) into two different phenotypes [6]. Understanding control of T cell differentiation is critical for developing new treatments in cancer and autoimmunity. The model that we have developed for T cell differentiation has 39 elements, where several elements are modeled with three levels of activity, which leads to 45 variables overall.

The second model that we are currently working on is a model of effects of malaria infection on mosquito cells [23]. This model has 41 elements, most of which are modeled with more than two levels of activity. Many elements also have more than two regulators, making the model increasingly complex. The tool that we have developed for automating logic rule development allows us to create rules for this model, which would be otherwise time consuming and error prone if done manually.

5. MODEL ANALYSIS AUTOMATION

In this section, we describe methods for analyzing developed models. The methods presented herein emphasize again the potential that EDA techniques bring to systems biology.

4.4 Application of model design methodology

We have applied our model design methodology to two cell signaling networks. One network includes circuitry that controls differentiation of T cells (lymphocytes) into two different phenotypes [6]. Understanding control of T cell differentiation is critical for developing new treatments in cancer and autoimmunity. The model that we have developed for T cell differentiation has 39 elements, where several elements are modeled with three levels of activity, which leads to 45 variables overall.

The second model that we are currently working on is a model of effects of malaria infection on mosquito cells [23]. This model has 41 elements, most of which are modeled with more than two levels of activity. Many elements also have more than two regulators, making the model increasingly complex. The tool that we have developed for automating logic rule development allows us to create rules for this model, which would be otherwise time consuming and error prone if done manually.

5.1 Hardware design framework

In order to speedup model analysis one can use a hardware emulation approach, which can start and control multiple network simulations in parallel, as well as compare final results obtained from parallel simulations. The design of such a framework usually starts with defining larger blocks. The next step in hardware design is to define modules within each block and describe them in a hardware description language (e.g., Verilog). One issue that arises when considering the qualitative studies of biological networks in hardware is defining the time of the execution of individual interactions. FPGAs are suitable for implementing biological network models because of their concurrent processing nature. To support this, we intend to develop modeling techniques that can facilitate correct relative timing when emulating signaling networks. One way to approach the problem of implementing asynchronous network updates using synchronous hardware (that uses a clock signal) is to design a system with an embedded pseudo-random number generator, as described in [7, 8].

Details about the Verilog implementation of the hardware emulation framework and the design uploading onto an FPGA are described in [7, 8]. After all blocks of the circuit are designed in Verilog, one can use simulation to analyze the design and to view signal waveforms. As an example, ModelSim [24] can be used to provide a waveform view of simulated signals, which can be very useful for analyzing element trajectories from initial to steady state.
5.2 Simulations via multicore parallel implementation

Another approach to speedup simulation of biological networks is to use multi-core platforms that have gained a lot of attention recently due to their potential to decrease the computation runtime. We implemented a logical simulator in C under OpenCL and executed it on an 8-way multi-core platform running Ubuntu Linux on a virtual machine where only four of the eight cores are used. Similar to the BooleanNet simulations in Python, the simulation takes as input the number of nodes, the name of each node, its initial state and a list of activators and inhibitors for each node. The objective of the simulator is to allow the network to progress until a steady state is reached. Each core of the multi-core platform runs at a 2GHz clock.

In our preliminary analysis of several models, FPGA implementation is several orders of magnitude faster than Python (BooleanNet [14]) implementation, and still an order of magnitude faster than the multi-core implementation (Table I). Given that these times reflect the emulation of a single instance of the model on the FPGA and that the FPGA platform can simultaneously run several copies (up to six, as shown in [7, 8]), it is expected that the speed of hardware emulation when compared to the parallel software simulations can be 2-3 orders of magnitude larger and 4-5 orders of magnitude larger than that of sequential, Python-based simulation in BooleanNet.

6. CONCLUSION

In this work, we have presented a methodology for automating the design and analysis of logical models of biological networks. This methodology allows for efficient design of discrete (not only Boolean) models, which have been proven beneficial in studying dynamics of biological networks.

Our future work will include development of tools to further automate the procedure of creating influence tables, as well as development of synthesis tools that translate a network description written in Verilog into logic rules, which can be studied using hardware emulation framework. This will allow for even more efficient construction and analysis of large-scale biological network models from which new insights can be gained.

7. ACKNOWLEDGMENTS

This work was supported by NIH Grants UL1-RR024153 and 1R01AI080799, and NSF Expeditions in Computing Grant 0926181. The authors would like to thank Solomon Sia (Electrical and Computer Engineering, Carnegie Mellon University), for his help in developing the tool that is part of the logic rule development methodology. We would also like to thank Penelope Morel (Immunology, University of Pittsburgh), Yoram Vodovotz (Surgery, University of Pittsburgh), and Shirley Luckhart (Medical Microbiology and Immunology, University of California-Davis) for beneficial discussions related to model development.

8. REFERENCES


Table I. Hardware emulation (FPGA) speedup vs. simulations in Python (BooleanNet) or simulations in C under OpenCL on an 8-way multi-core platform (Parallel).

<table>
<thead>
<tr>
<th>Model</th>
<th># nodes</th>
<th># rounds</th>
<th>BooleanNet</th>
<th>Parallel</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-LGL [1]</td>
<td>61</td>
<td>15</td>
<td>60s</td>
<td>0.14</td>
<td>0.019s</td>
</tr>
<tr>
<td>Th diff [2]</td>
<td>23</td>
<td>25</td>
<td>65s</td>
<td>0.16</td>
<td>0.022s</td>
</tr>
<tr>
<td>T cell diff [6]</td>
<td>54</td>
<td>16</td>
<td>45s</td>
<td>-</td>
<td>0.016s</td>
</tr>
</tbody>
</table>


