Representing Genetic Networks as Labeled Hybrid Petri Nets for State Space Exploration and Markov Chain Analysis

Curtis Madsen
School of Computing
University of Utah
50 S. Central Campus Dr. Rm. 3190
Salt Lake City, UT 84112

June 5, 2009

Abstract

This paper presents the bachelor's thesis of Curtis Kendall Madsen which can be broken down into the following three goals. The first goal of this project is to develop a way to convert genetic networks into logical models. Once this is done, finding the state graph of these nets and performing *Markov chain analysis* on them can provide researchers with insight into the reachability of the states in the original network. Therefore, the second goal of this project is to develop an automated tool that can perform state space exploration of a logical model, and the third goal is to implement a Markov chain analyzer for the stage graph.

For the logical representation of genetic networks, the conversion method uses *labeled hybrid Petri nets* (LHPNs) because they are designed for modeling logic while still allowing for important information required by Markov chain analysis such as transition rates to be included in the model. This conversion method is automated and is integrated into the `iBioSim` program allowing users to transform a *Genetic Circuit Model* (GCM) into an LHPN with just a click of a button. Also, iBioSim now includes the LHPN file type in its file tree so users can view and edit LHPNs once conversion is complete. In addition, a method for performing state space exploration on an LHPN allows the user to view the state graph using Graphviz's Dotty tool. Finally, a Markov chain analyzer that uses an iterative method to calculate a probability distribution over the states in the state graph makes it possible for users to determine the probability that any given state in the system will be reached. Since LHPNs are also used to represent asynchronous circuit designs in `ATACS`, users can use these methods to view the state graph and perform Markovian analysis on these designs as well.

Contents

1 Introduction

Over the past few decades, synthetic biology has become a great interest to biologists and engineers alike [6, 8]. Synthetic biology combines the research of biology and engineering with the ultimate goal of being able to design and build useful biological systems. In order to accomplish this goal, ways to model and analyze biological networks are crucial. Although many different techniques have been proposed and implemented, there is still a great need for better ways to model and analyze networks. A method for converting a system to a logical model could be very useful because it would allow biologists to leverage the abilities of existing logical analyzers. These logical models could also be used to determine which states of the biological system are more probable to be reached than others.

The *labeled hybrid Petri net* (LHPN) model is one such logical model that could be used to represent a biological system. Petri nets are often used by asynchronous circuit designers to model the logic of their circuits. Also, a state graph can easily be constructed from an LHPN to help determine which states of a system are reachable. However, this process can be quite tedious especially if the state space of the system is large. It would therefore be beneficial to both synthetic biologists and asynchronous circuit designers if there were some way to automatically generate a state graph from an LHPN.

Once designers have a state graph, they need to be able to analyze it. One useful piece of information is the probability of reaching any state in the state graph. One way to accomplish this goal is to use *Markov chain analysis*. A *Markov chain* is a special kind of *stochastic process* where the future evolution of the system depends only on the current state of the system and not on its past history. Since most state graphs already have this property, they are prime candidates for Markov chain analysis.

This paper presents this project which is to first develop a tool that automatically converts a *Genetic Circuit Model* (GCM) into an LHPN. The next step is to perform state exploration on the LHPN to produce a state graph representation of the original model. Once the tool finishes state exploration, it can output a file containing the state graph that can be viewed with Graphviz's Dotty tool. The final part of this project is to implement a Markov chain analysis engine that performs Markovian analysis on the state graph using the transition rates from the LHPN. This analysis produces a probability distribution over the states in the graph that can be useful in determining which states of the model are more likely to be reached than others.

Section 2 presents a biology background and gives a detailed definiton of Markov chains that will be helpful to someone with a computer science background in understanding this paper. Section 3 describes the related work that has been done in modeling and analyzing genetic networks. Section 4 discusses the conversion of a genetic circuit into a logical model. Section 5 describes how to find the state space of the

logical model and the data structure used in the state space exploration algorithm. Section 6 presents the Markov chain analysis algorithm. Section 7 discusses the benefits of integrating this project into `iBioSim` and `ATACS`. Section 8 proposes future extentions to this project. Section 9 concludes this report.

2 Background

In order to fully understand this paper, some information on biology and Markov chains is required.

2.1 Biology

All living organisms are composed of one or more cells. They are the basic building blocks of life. In order for the organism to function, each cell must be able to obtain nutrients from its environment, convert these nutrients to energy, send signals to other cells, respond to signals from others cells and its environment, and reproduce. In order to accomplish these tasks, each cell has a genetic regulatory system of DNAs, RNAs, proteins, and other small molecules that interact with each other in chemical reactions to regulate gene expression and keep the cell functioning properly.

The following is a list of terms used throughout this paper:

Species A molecule in a biochemical network.

Reaction A transformation of species.

Activator A modifier which activates a reaction.

Repressor A modifier which represses a reaction.

Concentration The number of molecules over the volume of the cell.

2.2 Markov Chains

A Markov chain is defined as a discrete-time stochastic process that exhibits the Markov property [4]. A stochastic process, as defined by Stewart, is "a family of random variables $\{X(t)|t \in T\}$ defined on a given probability space and indexed by the parameter t, where t varies over some index set (parameter space) T" [20]. For most stochastic processes, the index set T is representative of time and each random variable $X(t)$, also referred to as a state, signifies an observation at time t. Stochastic processes are characterized by uncertainty of future states. For example, flipping a coin is a stochastic process because there is a chance that

the coin will come up heads or tails, and we won't know which it will come up for sure until we actually flip it. Stochastic processes can have the property of being either discrete-parameter or continuous-parameter. The difference between these two has to do with the index set T. If the elements of T take discrete values (i.e. $T = \{0, 1, 2, 3, ...\}$), then the stochastic process is said to be discrete-parameter. Otherwise, if the elements of T take continuous values (i.e. $T = \{t|0 < t < 10\}$), then the stochastic process is considered to be continuous-parameter.

If the stochastic process exhibits the Markov property, then it is called a Markov process. Having the Markov property means that "the future evolution of the system depends only on the current state of the system and not on its past history" [20]. In other words, systems exhibiting the Markov property are memory-less in that the sequence of transitions leading up to a given state will not affect the transition from that state to the next state. This property is formally stated in equation (1).

$$\texttt{Prob}\{X_{n+1} = x_{n+1}|X_0 = x_0, X_1 = x_1, ..., X_n = x_n\} = \texttt{Prob}\{X_{n+1} = x_{n+1}|X_n = x_n\} \tag{1}$$

Our coin flipping example also exhibits the Markov property because as long as it is a fair coin, it will still have a 50 percent chance of coming up heads no matter how many times the coin has come up heads in the past. This is a very powerful property of a Markov process because it allows us to predict all the future states of the system with only knowledge of the current state.

If the Markov process has a state space that is discrete, then it is called a Markov chain. Having a discrete state space just means that the states in the Markov chain are countable and can be indexed by a countable set such as the natural numbers. This property allows Markov chains to be represented in a presentable way. "Markov chains are often described by a directed graph, where the edges are labeled by the probabilities of going from one state to the other states" [4]. These probabilities of going from one state to another are referred to as transition probabilities and are denoted by p_{ij} where $p_{ij} = \texttt{Prob}\{X_{n+1} = j|X_n = i\}$ [13]. The transition probabilities in a Markov chain can be used to build a transition matrix which can then be used to find the stationary distribution of the chain using either direct or iterative methods.

State graphs can be represented as directed graphs with edges labeled with the transition rates from one state to another. This means that Markov chain analysis can easily be applied to them to calculate a probability distribution over their states.

3 Related Work

The modeling and analysis of biological networks has been of great interest to researchers in synthetic biology. Many different models and methods have been proposed to represent and analyze genetic systems [14]. Probably the most prevalent modeling language for representing genetic networks is the *Systems Biology Markup Language* (SBML) [3]. SBML allows users to describe the species of a network and how those species interact with each other through reactions. In addition, SBML allows users to add special functions to their model such as constraints, events, and rules.

SBML models can be transformed into a group of *ordinary differential equations* (ODE) through the use of the law of mass action [24, 5]. This ODE model can then be analyzed and simulated through the use of well known methods of analysis [19, 21]. The problems with ODEs are that they assume that the concentrations of species are continuous quantities, that these concentrations are very large amounts, and that reactions occur deterministically. Most genetic networks have small, discrete amounts of species and reactions that occur sporadically.

Because of the problems found with ODE simulation, researchers have turned to stochastic simulation. *Gillespie's stochastic simulation algorithm* (SSA) is one such method of analysis [11]. The SSA first sets up the initial state of the system by initializing time and the concentration of each molecule. Then, it loops through evaluating the propensity functions of each molecule, drawing random numbers to determine when the next reaction fires, firing the next reaction, and checking to see if the simulation time has been reached. Through this algorithm, SSA effectively deals with the small, discrete amounts of species and the randomness of reactions. SSA also improves efficiency of simulation by stepping over useless time steps (time steps where no reactions occur).

SSA has also been improved further by *Gibson and Bruck's Next Reaction Method* [10], which eliminates the need to update the propensity functions and the need to generate random numbers and next reaction times during every iteration of the algorithm, and *Tau Leaping* [12, 7], which allows many fast reactions to fire each time step. Reaction-based abstraction has also been applied to SSA to help deal with really large models. Some of these abstractions include irrelevant node elimination, enzymatic approximations, operator site reduction, and dimerization reduction [16]. These methods abstract away constant species and fast uninteresting reactions creating a smaller model that is easily simulated.

Even though a lot can be learned about a genetic network by using stochastic analysis, researchers are often interested in the states that a network can be in. This analysis method does not explicitly give the probability of different states of the system occurring. Often a researcher has to perform many different

simulations and then compare the data and record which states the network ended up in by hand. This can be very computationally intensive because simulations can take a lot of time and resources to generate. For this reason, methods to convert biological networks into logical models have been developed. One of these methods consists of a researcher analyzing a network for various components such as feedback loops and constructing a qualitative logical model by hand [22, 23]. Although this method ultimately produces a desired logical model, it can be quite time consuming and error prone. Thus, automated techniques have been developed that take an SBML model and abstract the model into a logical SBML model [15]. These logical SBML models usually consist of boolean variables representing different levels of species concentrations and reactions containing equations representing the relationships among species.

With these logical models, logical analysis such as Markov chain analysis can be applied to the network. Researchers can then determine which states the system is most likely to be in without running hundreds, thousands, or millions of stochastic simulations. However, the one downside of using these logical models is that they are generated from SBML which can be very tedious to use when building a model from scratch. This is because SBML is used to model networks on a low level and therefore requires a lot of detail.

To address this problem with SBML, researchers have developed higher level models such as the *Genetic Circuit Model* (GCM) [18, 17]. In this model, a user can specify the species in a network along with how they influence each other through either activation or repression. The user also has the ability to specify some parameters to fine tune the network. From this, automated methods can generate an SBML model that can be simulated to see how the network behaves. However, directly converting a GCM to a logical model would be more efficient because it would save researchers the time and effort it takes to convert from the high level description of GCM to the low level description of SBML and back to the high level description of a logical model.

4 Conversion of a Genetic Circuit Into a Logical Model

When converting a biological network into a logical model, there are a few goals to keep in mind. First, it is crucial to preserve the behavior of the original network. This is important because otherwise the conversion would not be correct and the model it produces would not provide users with any useful information about the original model. Next, the conversion process needs to have some way of breaking the concentrations of species up into different levels. This will greatly improve state space exploration because it will allow the algorithm to only generate states for each level instead of a state for every possible value that each species

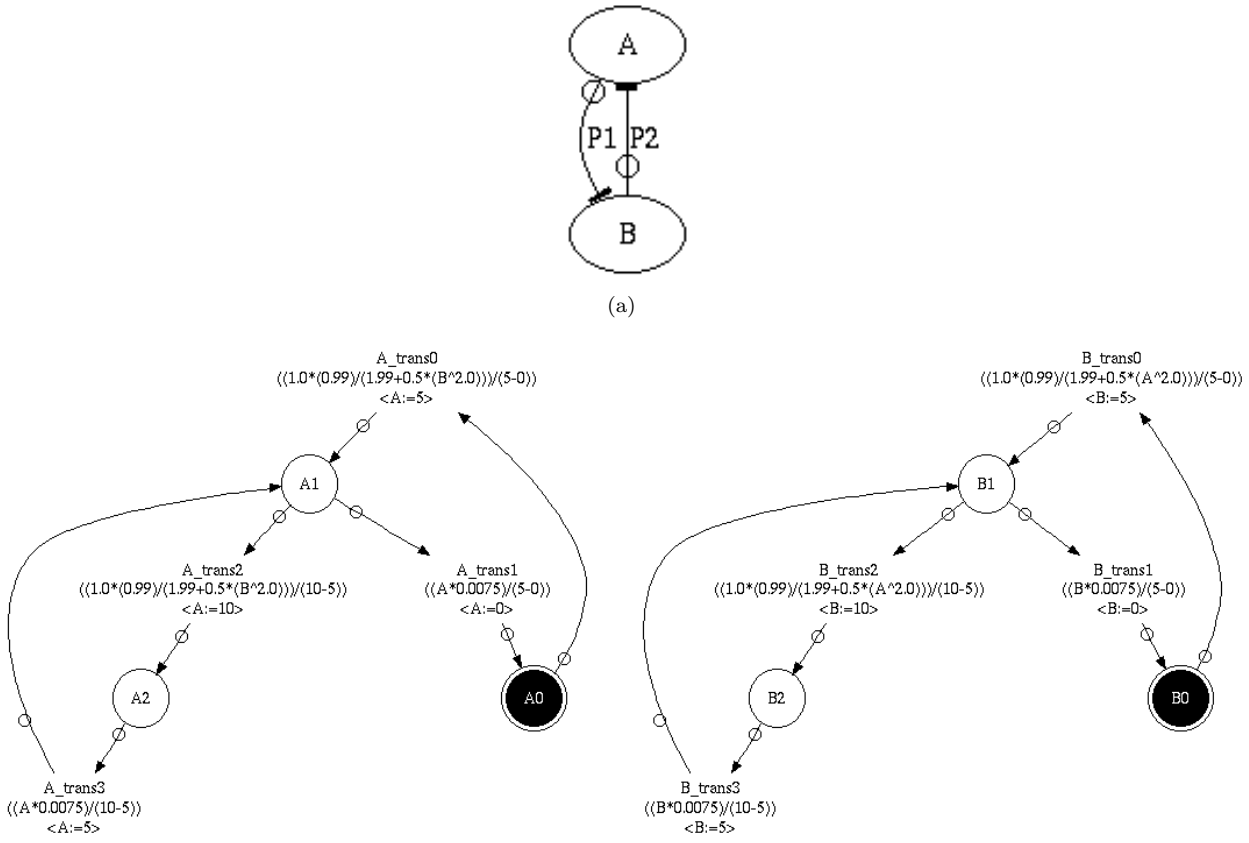

Figure 1: (a) A simple GCM where A represses B and B represses A. (b) The LHPN that is generated from the GCM in (a). This LHPN was generated by selecting levels at 0, 5, and 10 for A and 0, 5, and 10 for B. The subnet on the left represents that concentration of species A because each of its transitions makes assignments to the variable representing A, and the subnet on the right represents the concentration of species B because each of its transitions makes assignments to the variable representing B. Initially the places representing the concentrations of A and B being equal to zero are marked. Finally, as shown in the diagram, each transition has a transition rate equation that is dependant on the repression influence of the other species. Each of these are viewed using Graphviz's Dotty program.

can take.

With these considerations in mind, there are a couple of options for a logical model to consider. The first option is the SBML model. It has the benefit of allowing transition rates to be stored as equations within a reaction's kinetic law. It also allows for the creation of a species for each concentration level for each species in the GCM. However, the reactions in SBML need to be defined by piecewise equations that depend on the concentrations of the other species in the system in order to create a logical model that behaves the same as the original model.

Given this complication, the other model to consider is the LHPN model. LHPNs offer the benefit of the

creation of transitions that could be used to store the transition rates and make assignments to variables. Also, finding the state space for an LHPN is quite easy because it just involves keeping track of tokens and firing transitions. For these reasons, the best choice is the LHPN model for state space exploration and Markov chain analysis.

The GCM data structure is composed of several parts that need to be considered when converting the model into an LHPN. The most important part is the species because the concentrations of each species determines the state of the system. The conversion process begins by first creating a new variable in the LHPN for each species in the GCM. Then, the user is asked to provide thresholds for each species and a place representing each of these thresholds is created. These places are then linked together with transitions to and from the place with the closest greater than threshold and the place with the closest less than threshold. These new transitions also get assigned variable assignments that are consistent with the places they connect to. After this, the LHPN has everything we need to do state space exploration and Markov chain analysis expect for transition rates. We then use the global parameters and the local parameters from the influences and species of the GCM to fill in these rates with an equation that takes into account the activation and repression influences of other species. After this final step, the final LHPN model behaves the same as the original GCM model. Figure 1 shows an example of a simple GCM and its corresponding LHPN.

5 State Space Exploration

There are two mains goals in state space exploration. The first goal is to make the state graph viewable. This would be really handy for users who just want to look at the state graph to make sure that their system is doing the right thing. It could also be useful if a user wants to perform optimizations or analysis on the system by hand by allowing them to print off the state graph and draw on it directly. The other goal of state graph exploration is to come up with a data structure for storing the state graph. This data structure needs to be usable by future tools so they can analyze the system. It is also desirable to choose a data structure that is as efficient as possible when is comes to the amount of space it will use in the worst case.

The first option for a state graph data structure is to just have a directed graph of nodes and edges. Each node could then contain all of the information about a state and the edges would be the transitions from one state to another. This data structure has the benefit that it is simple to implement and understand because it is how people naturally think of and draw state graphs. However, the downside of using this data structure is that it would be very difficult to retrieve a state out of the graph given its state encoding

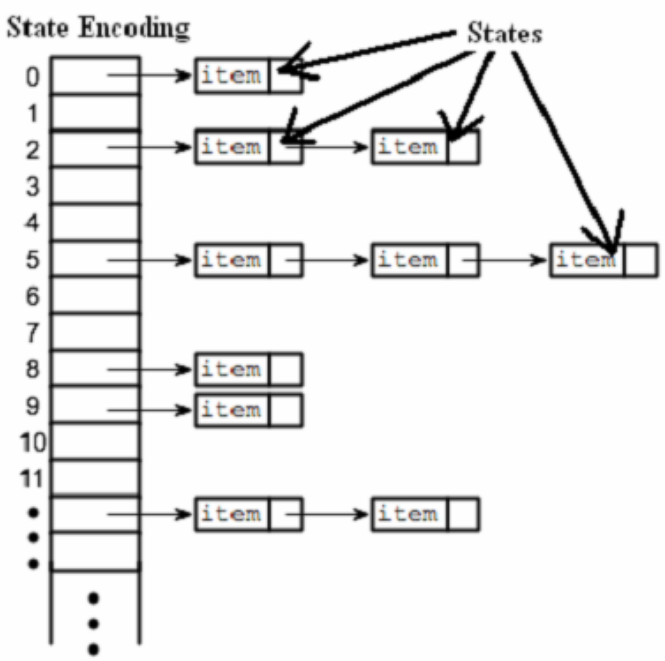

Figure 2: This figure shows a graphical representation of the bucketed hash map data structure used to store a state graph. The array labeled "State Encoding" is an array where each element is a state encoding that is assigned to a certain index by a hashing function. Each state encoding points to a linked list of all of the states that have that same state encoding.

because this operation would require that a search algorithm walks through each state in the state graph in search of the given state.

Another option for the state graph data structure is to use a bucketed hash map like the one shown in figure 2. In this data structure, the state encodings would be mapped to a linked list of all the states that have this same encoding. Each state would then also have its own pointers to its next and its previous states so that the state graph could be traversed. This solves the problem of searching for a state given its state encoding because the hash map would allow for quick look-up of a state. One concern of using this data structure is collisions. However, this should not be a problem because of the linked lists of states. If two or more state encodings hash to the same bucket in the hash map, then the states will be joined in a linked list instead of overwriting each other. Since this option solves the problem that the first data structure had and is fairly efficient when it comes to searching for a given state, it is the data structure that the state space exploration algorithm uses.

The next decision to be made with state space exploration is how to view the state graph. One option is to write a graphical viewer program from scratch, but this can be very time consuming and there already

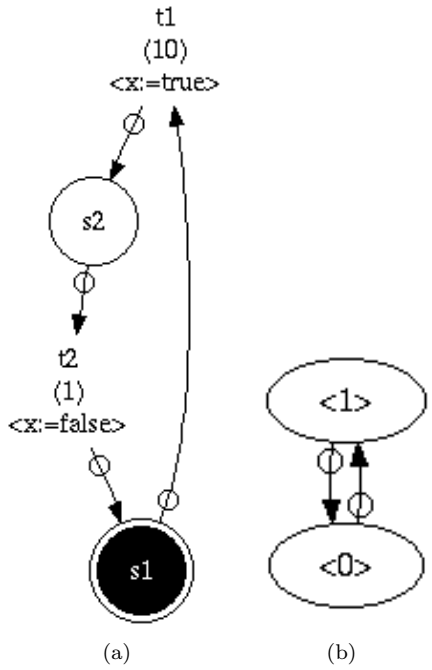

Figure 3: (a) A simple Petri net with two transitions and two states. Initially state $s1$ is marked and x is false. Transition $t1$ assigns x to true and has a transition rate of 10 and transition $t2$ assigns x to false and has a transition rate of 1. (b) The state graph that is generated from the net in (a) after state space exploration is complete. Each of these is viewed using Graphviz's Dotty program.

exists ways to view the state graph such as using Graphviz's Dotty program [9]. This program offers a simple way to view a state graph once it is formatted in a dot file. Figure 3 shows an example of a simple Petri net and its corresponding state graph viewed with Dotty. The ease of generating a dot file is a good motivating factor for using Dotty. A dot file for a state graph is composed of one line for each state and one line for each transition. Because it is so easy to use, the state space exploration algorithm uses Dotty to view state graphs.

6 Markov Chain Analysis

The goal of Markovian analysis is to be able to take a state graph and its corresponding LHPN and compute a probability distribution over that states in the state graph. There are several iterative and direct methods to choose from for calculating this distribution. Just like in the case where there is a choice of data structures for the state graph, it is desirable to pick an algorithm that is efficient yet simple to understand and implement.

One option for implementing Markov chain analysis is to use the direct method of Gaussian elimination. The general idea of this method is to reduce the transition matrix by using a pivot that moves along the

diagonal of the transition matrix while elementary row operations are applied to the transition matrix to reduce the upper right hand corner of the matrix and zero out the lower left hand corner [20]. The problem with using this and other direct methods is that a transition matrix needs to be constructed and maintained. If the algorithm is not careful on how this is done, then this process can be very costly both in processing time and in memory. Iterative methods can also be costly when it comes to processing time, but they wouldn't necessarily require a transition matrix to be constructed because the probability of each state can be stored right in the state graph data structure and be updated as we walk through the state graph.

Algorithm 1 Markov Chain Analysis Iterative Algorithm

1: color the state graph with shortest distances from initial state
2: calculate the `period` of the state graph
3: set the probability of the initial state to 1
4: set the probability of all other states to 0
5: initialize `step` to 0
6: **repeat**
7: **for** each `state` in the state graph **do**
8: `step` = `step` + 1
9: **if** `color`%`period` = `step`%`period` **then**
10: $\texttt{state.prob} = \sum_{\text{prev states}} \left(\texttt{prev.prob} \times \frac{\texttt{prev.rate}}{\sum \texttt{prev.rate}} \right)$
11: **end if**
12: **end for**
13: **until** convergence
14: normalize the probabilities of each state to sum to 1

Since direct methods would be costly when it comes to time and memory while iterative methods would only be costly as far as time is concerned, the algorithm implements an iterative method called the power method. The general idea behind this method is to initialize a vector to a random initial distribution where all of the entries sum to one. This vector is then continually multiplied by the transition matrix until the values in the vector converge [20]. In the case of the state graph, the initial state's probability can be set to one while all other states' probabilities can be set to zero. The probability of each state can then be updated by summing up the probability of its previous states multiplied by the transition probability from each of these states. This process is repeated until convergence and then the probability of each state is normalized so that they all sum to one. This algorithm's pseudocode is given in Algorithm 1.

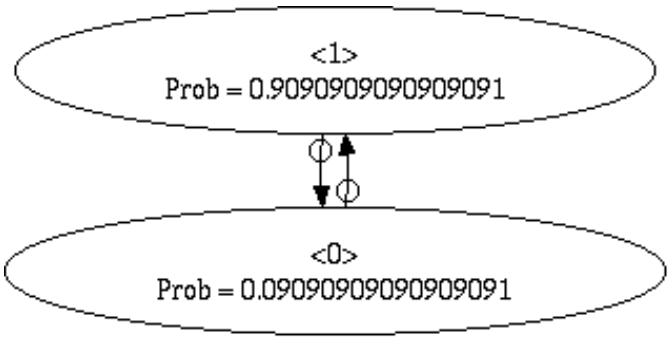

Figure 4: This figure shows the state graph of the Petri net from figure 3 after Markov chain analysis has been applied. This shows that the net is most likely going to spend about 10 percent of its time in state $s1$ and 90 percent of its time in state $s2$.

7 Integration Into iBioSim and ATACS

Once the implementation process is complete, the final step of this project is to integrate it into `iBioSim` [2] and `ATACS` [1]. First, a save as option is added to GCMs that allows users to convert them to LHPNs and the lpn file type is added to the `iBioSim` file tree. `iBioSim` can then take advantage of the LHPN editor that already exists in `ATACS` which allows LHPNs to be edited in a GUI and viewed with Dotty. Next, a right-click option is added to lpn files that allows users to view the state graph and perform Markov chain analysis. By clicking the view state graph option, the program will call the state space exploration algorithm to automatically create a state graph from the lpn file and output a dot file containing the state graph. It will then call Dotty on the dot file allowing the user to view the state graph. The perform Markov chain analysis button will do the same thing except the program will make an extra call to the Markovian analysis algorithm to perform Markov chain analysis before the state graph is outputted to a dot file. This will cause the dot file to include probabilities for each state which will show up when the state graph was viewed. An example of selecting the perform Markov chain analysis option on the Petri net in figure 3 is shown in figure 4.

8 Future Work

Although this tool can perform state space exploration and Markov chain analysis, there are a lot of extensions that can be made to it to improve its usefulness and usability. First of all, it can sometimes be difficult to choose where to break the concentrations of species up into different levels. If these levels are chosen poorly, then the state space exploration and Markov chain analysis will get bogged down because there will be too

many states or they won't be able to tell us anything interesting about the network because there are too few states. One improvement to this tool could be to automate some way of suggesting good thresholds for the species' concentrations. In order to accomplish this goal, an analysis method must be developed that analyzes a network and determines at which concentrations interesting things happen in the network.

The next improvement deals with state space exploration. The state space exploration algorithm currently only deals with boolean variables when computing a state encoding for each state. This needs to be extended to include discrete variables in an LHPN. Making this change would require that there be some other way of representing the encoding of each state such as using the marking or combining all of the variables together in a string that is delimited by special characters; however, it would allow the state graph explorer to generate a better state graph for the nets produced by the GCM to LHPN function.

Many of the variables in an LHPN can be assigned a range of values instead of a single value. Currently, the Markov chain analyzer does not take these ranges into account, but it could be extended to deal with them. Although is is unclear exactly how to deal with the ranges, one idea could be to analyze the state graph using the lower bounds and then re-analyzing using the upper bounds and then displaying a range of probabilities for each state. The one issue with dealing with ranges is that the probabilities of all of the states needs to sum to 1, so picking a probability out of the range from each state may end up leading to a distribution that isn't consistent. This issue needs to be researched more to determine if there is a good way of handling it.

9 Conclusion

In conclusion, the logical representation of a genetic network combined with a Markov chain analysis engine will help researchers greatly because it will allow them to obtain the probability of each state of the system occurring. Instead of running a lot of simulations and doing this analysis by hand, these researchers can spend their time on more important things.

This project allows users to convert a GCM to an LHPN and effectively analyze that LHPN model to determine which states their system can get into and what the probability of being in each state is. Because it is integrated into `iBioSim` and `ATACS`, users can easily use this tool by selecting to convert a GCM to an LHPN, to perform state space exploration, or to perform Markov chain analysis and viewing the output with Dotty. The information gained from Markov chain analysis can help researchers better understand the network they are analyzing and maybe one day allow them to construct their own biological networks. Also,

with the LHPN that is derived from a GCM, researchers can use virtually any analyzer that accepts LHPNs to analyze their model or can use the logical analysis engine to analyze other LHPNs.

References

[1] ATACS. `http://www.async.ece.utah.edu/ATACS/`.

[2] iBioSim. `http://www.async.ece.utah.edu/iBioSim/`.

[3] Systems Biology Workbench Development Group. `http://www.sbw-sbml.org/`.

[4] Markov chain - properties of markov chains, markov chains with a finite state space, scientific applications, markov parody generators. *Cambridge Encyclopedia*, 49, `http://encyclopedia.stateuniversity.com/pages/14446/Markov-chain.html`.

[5] H. Abrash. Studies concerning affinity. *Journal of Chemical Education*, 63:1044–1047, 1986. English translation of Waage and Guldberg's 1864 paper.

[6] A. Arkin. Setting the standard in synthetic biology. *Nature Biotechnology*, 26:771–774, 2008.

[7] Y. Cao, D. T. Gillespie, and L. R. Petzold. Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys.*, 124, 2006.

[8] D. Endy. Foundations for engineering biology. *Nature*, 438:449–453, 2005.

[9] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications. *Software - Practice and Experience*, 00(S1):1–5, 1999.

[10] M.A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem.*, A 104:1876–1889, 2000.

[11] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[12] D. T. Gillespie and L. R. Petzold. Tau leaping. *J. Chem. Phys.*, 119:8229–8234, 2003.

[13] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability: Second Revised Edition.* American Mathematical Society, 1997.

[14] H. De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *J. Comp. Biol.*, 9(1):67–103, 2002.

[15] H. Kuwahara and C. Myers. Production-passage time approximation: A new approximation method to accelerate the simulation process of enzymatic reactions. *Journal of Computational Biology*, 15(7):779–792, 2008.

[16] Hiroyuki Kuwahara. *Model Abstraction and Temporal Behavior Analysis of Genetic Regulatory Networks.* PhD thesis, University of Utah, 2007.

[17] N. Nguyen. *The Design of a Genetic Muller C-Element.* PhD thesis, University of Utah, 2008.

[18] N. Nguyen, H. Kuwahara, C. Myers, and J. Keener. Design and analysis genetic circuits. In *The 13th IEEE International Symposium on Asynchronous Circuits and Systems*, March 2007.

[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The art of Scientic Computing, 2nd ed.* Cambridge University Press, 1992.

[20] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Priceton University Press, 41 William Street, Princeton, NJ, 08540, 1994.

[21] S. H. Strogatz. *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry And Engineering*. Westview Press, 1994.

[22] D. Thieffry and R. Thomas. Dynamical behaviour of biological networks: Ii. immunity control in bacteriophage lamabda. *Bull. Math. Biol.*, 57(2):277–297, 1995.

[23] R. Thomas. Regulatory metworks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology*, 153:1–23, 1991.

[24] P. Waage and C. M. Guldberg. Studies concerning affinity. *Forhandlinger: Videnskabs - Selskabet i Christinia*, 35, 1864.