A Biological Simulator using a Stochastic Approach for Synthetic Biology

by

Daniel D. Kim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005

Author . . . .
Department of Electrical Engineering and Computer Science
January 28, 2005

Certified by . . . .
Thomas F. Knight Jr.
Senior Research Scientist
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Biological Simulator using a Stochastic Approach for Synthetic Biology

by

Daniel D. Kim

Submitted to the Department of Electrical Engineering and Computer Science
on January 28, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Synthetic Biology is a new engineering discipline created by the development of genetic engineering technology. Part of a new engineering discipline is to create new tools to build an integrated engineering environment. In this thesis, I designed and implemented a biological system simulator that will enable synthetic biologists to simulate their systems before they put time into building actual physical cells. Improvements to the current simulators in use include a design that enables extensions in functionality, external input signals, and a GUI that allows user interaction. The significance of the simulation results was tested by comparing them to actual live cellular experiments. The results showed that the new simulator can successfully simulate the trends of a simple synthetic cell.

Thesis Supervisor: Thomas F. Knight Jr.
Title: Senior Research Scientist

Acknowledgments

I would first like to thank my thesis advisor, Professor Thomas F. Knight Jr. for piquing my interest in Synthetic Biology, giving me the opportunity to pursue what I fancied, and being kind and understanding amidst my multiple disappearances.

I also thank Jonathan Goler for the time he spent helping me pick my topic and eventually leading me to Sri Kosuri. Without Jonathan, I am sure I would still be twiddling my thumbs wondering what to do. I also owe Sri a debt of gratitude for giving up so much of his time and effort trying to get a clueless student going.

I survived this school with the camaraderie and encouragement of my friends and roomates. Thanks to them for keeping me sane throughout my schooling. I am afraid I have not given them nearly as much as they have given me.

Finally, I am deeply grateful to my family for being the anchor in my wandering life. Without them I would undoubtedly be lost at sea.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

If one were to say that the end of 20th century was defined by computers, one might also say that the 21st century will be defined by advances in biology. Many of the techniques that allowed computer technology to advance have been used to create tools that allow biology to be studied in new ways, such as PCR machines, DNA sequencing, and data mining. As engineering and biology disciplines merge, it has created a renaissance in biological research, as indicated by the increase in grants given for research related to the life sciences. MIT's new Computational and Systems Biology degree is further evidence that a new discipline between engineering and biology is here to stay and cannot be ignored.

The discipline combining engineering and biology is extremely broad. Some research groups focus on analyzing signaling systems in cells as one would analyze analog signals and systems in electrical devices while other groups use data mining techniques to discover motifs and patterns in the current plethora of DNA sequences available for analysis. Still, others desire to engineer synthetic cells in the hopes of creating cells that behave according to our desires. Other research efforts, including this thesis, try to simulate biological systems in order to aid the understanding of complex behavior. All these efforts are done by those who one day hope to decipher the systems that evolution has designed over millions of years.

1.2 Motivation

The area of biological simulation can be very broad. Simulation efforts range from protein folding, to protein to protein interaction, to cellular systems, to multi-cellular systems. This thesis focuses on simulation at the cellular, system level. I believe that successful simulation of biological systems would be an incredible resource in the attempt to understand these complex systems. Some of the advantages of an effective systems simulator are as follows:

1. Predict the behavior of cellular systems without observing actual live cells.

2. Confirm the behavior of real cellular systems that have been observed.

3. Isolate and analyze subsystems that don't exist alone in nature.

4. Engineer cellular systems to behave a certain way.

Simulations may be the wrong way to approach the problem of understanding cellular systems. They may be unable to account for extremely small but salient interactions or the results may be useless or too complex to understand. Even still, the only way to find out how effective a simulator is is to build one and see how effective a simple simulator is in simulating real systems.

1.3 Implementation

The most important part in building a simulator is designing the algorithm. Fortunately, there has been plenty of previous research on simulation algorithms. Plenty of published papers wrestle with the correctness and efficiency of various algorithms that simulate chemical reactions. The purpose of this thesis is not to devise new algorithms, but rather to implement existing algorithms in a biological context in the best way possible, and later to determine its effectiveness.

1.4 Preview

The rest of this thesis deals with the background, details, and results of creating a biological simulator.

In Chapter 2, the background for this thesis is covered. First, the context and purpose of the thesis is explained. Next, the general concept of stochastic simulation is covered along with a description of related work. Finally, the challenges of creating a simulator are talked about followed by a list goals of this thesis attempts to acheieve.

In Chapter 3, the design of the simulator is presented in detail, including various design decisions and tradeoffs. Further, details of implementation are provided along with examples of its use.

Chapter 4 explains how a simple synthetic cell was modeled using the simulator. Both the details of the synthetic cell and the result of modeling are covered.

Chapter 5 provides a discussion on the results of modeling a cell using this simulator. In addition, the successes and shortcomings of the design and implementation of this thesis are also discussed.

Finally, Chapter 6 outlines several areas in which future work might be directed while Chapter 7 summarizes the contributions of this thesis.

Chapter 2

Background and Goals

This chapter provides background on the building blocks that support and inspire my thesis. The Synthetic Biology group along with BioJade gives context and purpose to this thesis while Stochastic Kinetics provides the foundation for a efficient and correct algorithm. Related work such as Tabasco gives the inspiration and help to implement this thesis. Finally, the chapter ends with the challenges and goals associated with this thesis.

2.1 Synthetic Biology

Synthetic Biology at the Computer Science and Artificial Intelligence Laboratory (CSAIL) explores research in microbial engineering. This project combines research in both biology and computation, and is a collaboration of groups at CSAIL, the MIT Department of Biology, and the Biological Engineering Division. The Synthetic Biology project is an attempt to create a new engineering discipline that modifies simple living cells in as a way of computing and interfacing more effectively to the real world. The approach to Synthetic Biology has 4 major components [11]:

1. Specify and populate a set of standard biological parts (genes) that have well-defined performance characteristics that can be used and re-used to build biological systems.

2. Develop and incorporate design methods and tools into an integrated engineering environment.

3. Reverse engineer and re-design pre-existing biological parts and devices in order to expand the set of functions that we can access and program.

4. Reverse engineer and re-design a simple natural bacterium.

Given these goals, Synthetic Biology is an ideal context in which to build programs that simulate cellular systems.

2.2 BioJade

The Synthetic Biology group at CSAIL has been working on the second component, developing an integrated engineering environment, by creating an online repository of standard biological parts and by developing software tools that interface with the repository to enable system design. Recently written by Jonathan Goler, a program called BioJade provides such an integrated engineering environment [6]. BioJade allows system design to be done at a logic gate level and then translates the logic into biological parts in order to create cellular logic gates based on the online repository.

BioJade utilizes a Distributed, FLexible and User eXtensible protocol for simulations (D-FLUX). D-FLUX is a plug-in type architecture that can wrap third-party simulators. More importantly, D-FLUX can translate cell models designed in BioJade into any format required by a simulator. By adhering to the D-FLUX protocol, any simulation can be integrated into the engineering environment.

2.3 Stochastic Kinetics

Stochastic algorithms were developed in the late 1970's in an effort to improve the accuracy in modeling smaller systems. This section gives a general overview on stochastic processes and how they can be used to model biological systems.

2.3.1 Describing Chemical Processes

Chemical reactions are typically described in the form:

$$xX_1 + yX_2 \rightarrow zX_3 + ... \tag{2.1}$$

where x molecules of substance X_1 reacts with y molecules of substance X_2 to produce z molecules of substance X_3, and so forth. Most complex systems can be broken up into elementary reaction of the form above.

A predictive, *deterministic* model of a system is typically created by solving a set of differential equations where each equation represents the time evolution of the concentration of a reaction species (Eq. 2.2). The reactions are usually assumed to occur fast in comparison to the time scale of interest, and are often considered to have reached equilibrium as in Equation 2.3, mostly because non-equilibrium assumptions are often too complex or time consuming to solve. Such deterministic models require certain key assumptions including: a sufficiently large number of molecules and no fluctuations or correlations in concentration values. Unfortunately, such assumptions do not hold in very small systems such as cells.

$$\begin{aligned} \frac{d[X_1]}{dt} &= f_1([X_1],[X_2],[X_3]...) \\ \frac{d[X_2]}{dt} &= f_2([X_1],[X_2],[X_3]...) \end{aligned} \tag{2.2}$$

$$\begin{aligned} 0 &= f_1([X_1],[X_2],[X_3]...) \\ 0 &= f_2([X_1],[X_2],[X_3]...) \end{aligned} \tag{2.3}$$

2.3.2 Markov Processes

In an attempt to improve the accuracy for modeling smaller systems, stochastic algorithms were developed [10]. Instead of considering species concentrations as continuous functions of time, the stochastic approach interprets molecular dynamics as a jump Markov process with discrete states [5]. When modeling a system, discrete states hold the number of each type of molecule while a jump from one state to another represents the occurrence of a reaction which changes the number of molecules in the next state. Markov processes jump from one state to another using a set of probability density functions (PDF). These PDF's are created for each reaction and determine the likelihood that a state jumps from one state to another.

For example, consider the following set of reactions:

$$A + B \longrightarrow^{k_1} C$$

$$B + C \longrightarrow^{k_2} D$$

$$D + E \longrightarrow^{k_1} E + B \tag{2.4}$$

The different molecules are represented by A, B, C, D, and E while the propensities of the reactions are given by k_1, k_2, and k_3. The propensity of a reaction determines how likely a reaction is to occur. More specifically, the probability that a reaction μ will occur in the next small time dt is $a_\mu dt + o(dt)$, where $o(dt)$ represents terms that are negligible for small dt. a_μ is independent of dt, but depends on the reaction μ, the number of molecules of each kind, and variables that may change with time such as temperature and volume. k_μ is another reaction constant that is a function of variable such as volume, temperature, electrolyte concentration, etc. If the molecules are grouped together (the number of molecules of X expressed as #X), the relationship between a_μ and k_μ can be described for the reaction involving molecules A and B as follows:

$$a_1 = k_1 \times \#A \times \#B \times dt + o(dt) \tag{2.5}$$

It is important to realize that both a_μ and k_μ are *not* the traditional deterministic rate constants used when describing macroscopic systems. Instead, these rate constants are mesoscopic rate constants which are not identical to, but may be related to, macroscopic rate constants. For instance, macroscopic rate constants do not depend on volume but the concentration of molecules do, while mesoscopic rate constants do depend on volume but the concentration of molecules do not. The differences may not seem significant, but in systems sensitive to reaction rates, such as biological ones, these slight differences may become amplified during simulation [3].

Using stochastic formulation, the system is then analytically modeled by a *master equation*, a single differential equation that gives the probability of finding the system in its current state and time. However, when modeling complex systems, the *master equation* become virtually intractable [4]. An alternative to solving stochastic formulation was presented by D. T. Gillespie using Monte Carlo techniques. Gillespie's algorithm has shown to be promising in delivering accurate, yet computationally feasible modeling [4, 3]. It is therefore a derivative of Gillespie's algorithm that is used for this simulator. More information on the Gillespie algorithm and how it was implemented can be found in Section 3.2.

2.4 Related Work

There are many programs that offer different kinds of simulations. BioJade currently uses D-FLUX to plug into Tabasco and Stochastrator. The Tabasco simulator is the most relevant simulator to this thesis because it is a main source of reference and inspiration. So much in fact, that the simulator created in this thesis will hereon be referred to as T2.

2.4.1 Tabasco

Tabasco is a simulator written by Sri Kosuri and Jason Kelly originally for the purpose of simulating T7 bactiophage DNA entering bacterium [7]. Tabasco is written in Java and is based on the modified Gillespie algorithm presented by Gibson et. al., called *Next Reaction Method* (Sec. 3.2.3) [3]. Although Tabasco can model some synthetically designed cells, it is highly tailored for researching bacteriophage T7, which allows for simplifications in the simulation. In addition, Kosuris version of Tabasco lacks the some functionality that need to be modeled in more complex systems in generic bacterium.

2.4.2 Stochastrator

Stochastrator is another simulator supported by D-FLUX [6]. Written by Eric Lyons in C++, it is also uses the Gibson-Modified-Gillespie algorithm that Tabasco uses. Like Tabasco, it deals with systems at the single molecule level. However, it does not provide much interaction and does not model inputs to the system easily.

2.4.3 BioComp

A major project that is underway and headed by Darpa, BioComp is a huge undertaking that attempts to model intracellular processes by accounting for every possible known functionality [1]. Such an approach seemed from yielding useful results at the onset of this thesis. However, BioComp recently released its first version of an open-source application, BioSPICE, that models dynamic cellular network functions. Due to the release occurring near the end of this thesis, it is unknown what algorithms are used nor the effectiveness of the program. BioSPICE maintains a comprehensive environment that integrates a suite of analytical, simulation, and visualization services. A worthwhile extension to this thesis would be to see if any of the implementation or results would be of any value to the BioSPICE endeavor.

2.4.4 Dizzy

Dizzy is a chemical kinetics simulator authored by the Bolour group at the Institute of Systems Biology [2]. Dizzy can model systems both deterministically with an ODE solver or stochastically with various implementations of Gillespie's algorithm. The various implementations include the *Direct Method*, *Next Reaction Method*, and τ*-Leaping Method* (Sec. 3.2).

2.5 Challenges

The greatest challenge in simulating extremely complex systems is determining which features are important and which features can be black boxed. Events in cells such as blocking, binding, and promoting, are due to intricate interactions between proteins. If we truly want to know what is exactly going on in a cell, we need to simulate the exact position of each protein, then account for the molecular interactions between proteins, then account for the atomic interactions, digging deeper and deeper until all of a sudden quantum physics is involved . Not only is such a simulator quite infeasible to design, but given the immense computing power that is required to simulate one protein folding, there is not enough computing power at the moment to perform such complete simulations.

Another challenge is that new molecular interactions and regulatory pathways are being found every year, especially at the level of mRNA transcription. The constantly increasing number of known functionality in cells is what makes a comprehensive project like BioComp so difficult. In addition, once a new functionality is found, it takes even longer to understand how newly discovered functionality affects already known functionality.

I believe the correct approach is from the top down. Model basic functionality first, and then account for more complex ones. The key is to model the basic functionality in such a way that it enables the addition of complex functionality later. This approach is beneficial because although it may be crude, basic functionality may still be used to effectively model a system. In addition, one might be able to discover which features and functionality can be black boxed by observing how well basic functionality models real systems.

2.6 Goals

An important step in creating an integrated engineering environment is to build software tools that enable system simulation so that systems can be tested before the tedious process of physically building the actual living cells. Although BioJade already interfaces with several simulators, none have been built specifically tailored to the needs of synthetic biology designers. Specifically, I see the the need as two-fold: (1) provide an interactive application that can easily alter the simulation variables and (2) design the architecture such that new features can be implemented as easily as possible. Allowing external input signals is very important from both a biological and a synthetic engineers point of view. All biological systems receive some kind of external input, and such signals are critical in the design of synthetic systems.

The new simulator will be largely based upon Tabasco. A re-designed and re-implemented version of Tabasco should be able to model generic bacterium in a manner that is constructive to synthetic biologists. T2 should be implemented in such a way that new functionality can be easily added when needed. The program needs to be highly modular and well documented. T2 will still be an extremely rough and crude estimation of the complexity of any real cell. Therefore, when new functionality is desired, the program should enable the easiest way to add such functionality. Such functionality includes different kinds of highly specialized gene regulation that are not accounted for in basic gene regulation theory.

Another purpose of synthetic biology, beyond building biological systems through computational methods, is to gain understanding about how cells work. Once the program is finished, there must be a way to measure the significance of its results. The results of a publication describing a synthetically engineered cell will be used to compare to the results of a simulated model to see if the simulation created biologically significant results. Only after I measure the significance of the simulations results can synthetic biologists use the simulator with any confidence. Hopefully, T2 will become useful enough to aid in the understanding of the living world.

In summary, the goals of my M.Eng research include:

1. Understanding the nuances of the Gillespie algorithm and affirm that it is indeed a sufficient way to model the biological systems of interest;

2. Creating a modular, robust, and well documented program that can easily be

extended to add more functionality as more complexity is desired;

3. Providing a usable interface in which user can input and alter the variables of their own custom systems.

4. Designing a new visualization method to communicate the data resulting from the simulations.

5. Testing and verifying that the simulator creates biologically significant data by cross checking results with actual experimental data.

Chapter 3

Architecture and Implementation

This chapter explains the architecture and the implementation of T2. The architecture of T2 was designed in order to achieve as much modularity as possible. The stochastic algorithm used in the simulator is also discussed in detail. Next, is a list of the functionality the simulator provides, both the supported molecules and supported reactions. Finally, the development of the GUI is explained along with examples of the simulator is use.

3.1 Design Overview

The main goal when designing the architecture of the program was to separate the algorithm from the data, the data being the reactions and the molecules involved. If the algorithm can be can successfully isolated from the data, then functionality such as more reactions and molecules can be added without altering the basic algorithm that is run. The desire is to have the flexibility to add the largest range of functionality while changing the least amount of code. With this architecture, the simulator can easily be built from the top down, starting with basic functionality and adding more complex interactions as needed.

The first step in isolating the algorithm from the data, is to commit upon an algorithm. Once the algorithm is chosen, the requirements can be extracted and interfaces can be created. As outlined by the dependency diagram (Fig. 3.1), the simulator class, T2Simulator, depends on the interfaces *Molecule* and *Reaction*. The requirements of the simulator and the details of the interfaces will be covered in following sections.

The dependency diagram summarizes the overall structure of the application. The format follows the guidelines set out by Liskov for module dependency diagrams [8]. Plain boxes represent regular classes while special boxes with lines represent either static classes or interfaces. All classes that make up the GUI are lumped into one box. The GUI mostly depends on T2Simulator, but also depends on the interfaces *Molecule* and *Reaction*. There are additional weak dependencies that exist if the GUI is programmed to display more information that is specific to a certain type of *Molecule* or *Reaction*.

The XMLParser is a static class that parses the XML input to the simulation. XMLParser uses the XMLObject class to transform the text XML into Java objects. XMLParser can then access the information in the XML input without having to parse through the whole file.

Underneath *Molecule* and *Reaction* are the classes that implement the interfaces. When adding functionality to the simulator, new classes that implement the interfaces can be created. Once the new classes programmed, T2Simulator can continue to run without being affected. The details of these classes are found in Sections 3.3 and 3.4.

T2 is written in Java mainly due to its familiarity but also for its relatively easy GUI programability. A program such as C/C++ would have been faster, but the current speed requirements are not that rigorous. In the end, the desire for a quick deployment of an interactive GUI outweighed the need for speed.

3.2 Algorithm

The stochastic formulation is modeled using Gillespie's algorithm based on Monte Carlo techniques. Gillespie has come up with several algorithms, each different variants to solving stochastic formulation. Three methods will be discussed: Direct Method, First Reaction Method, and Next Reaction Method.

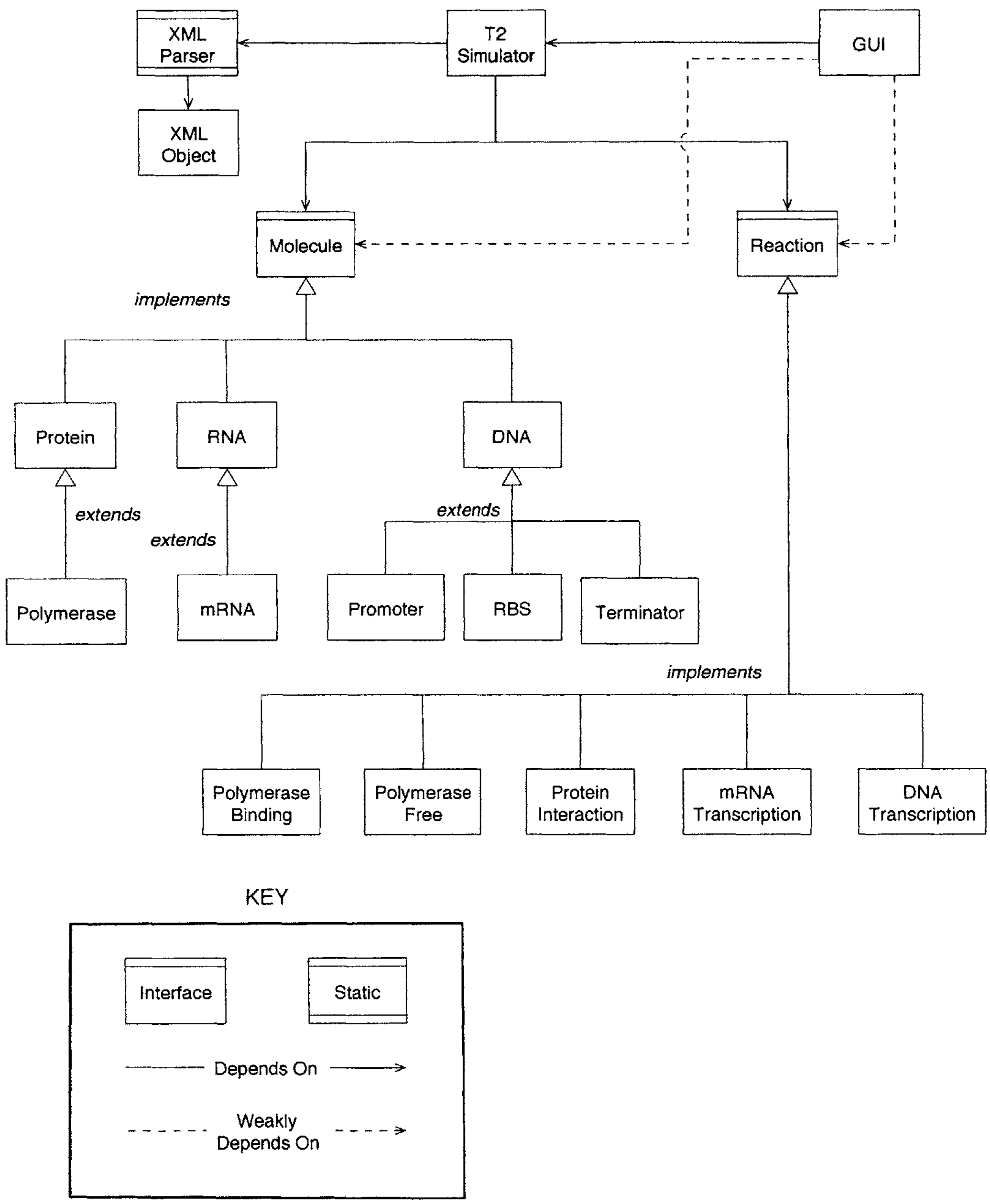

Figure 3-1: Dependency Diagram for T2. Outlines the architecture of the application.

3.2.1 Direct Method

The Direct Method attempts to find out which reaction, μ, occurs next, and when that reaction occurs, τ. The following is the probability that the next reaction is μ and occurs at time τ as determined by Gillespie:

$$P(\mu,\tau)d\tau = a_{\mu}exp(-\tau\sum_{j} a_j)d\tau \tag{3.1}$$

Integrating Equation 3.1 over all τ from 0 to ∞ gives the probability distribution for reactions:

$$Pr(Reaction = \mu) = a_{\mu}/\sum_{j} a_j \tag{3.2}$$

Summing Equation 3.1 over all μ gives the probability distribution for times.

$$P(\tau)d\tau = (\sum_{j} a_j)exp(-\tau\sum_{j} a_j)d\tau \tag{3.3}$$

Using both the probability distribution for reactions and times, the Direct Method algorithm can be run:

Direct Method Algorithm

1. Initialize - set initial numbers of molecules and set $t \leftarrow 0$.
2. Calculate propensity function, a_i for all i.
3. Choose μ according to distribution in Eq. 3.2.
4. Choose τ according to an exponential with parameter $\sum_j a_j$ as in Eq. 3.3.
5. Change the number of molecules to reflect execution of reaction μ. Set $t \leftarrow t+\tau$.
6. Go to Step 2.

3.2.2 First Reaction Method

The First Reaction Method is very similar to the Direct Method except that instead of generating μ and τ directly, it generates a putative time, τ_i, which is a time the reaction would occur if no other reaction occurred first. The putative time can be derived by the following equation where r is a random number uniformly distributed in the unit interval:

$$\tau_i = 1/(\sum_j a_j)log(1/r) \tag{3.4}$$

The Direct Method and First Reaction Method algorithms are provably equivalent [3]. Using Equation 3.4, the First Reaction Method algorithm can be run:

First Reaction Method Algorithm

1. Initialize - set initial numbers of molecules and set $t \leftarrow 0$.
2. Calculate propensity function, a_i for all i.
3. For each i, generate a putative time, τ_i, according to Eq. 3.4.
4. Let μ be the reaction whose putative time, τ_i, is least.
5. Let τ be τ_i
6. Change the number of molecules to reflect execution of reaction μ. Set $t \leftarrow t+\tau$.
7. Go to Step 2.

3.2.3 Next Reaction Method

The Next Reaction Algorithm was developed by Gibson and Bruck as an extension to the First Reaction Method as an attempt to increase the speed of the algorithm. The main improvements are recalculating only when necessary, appropriate reuse of τ_i values, absolute time scales, and organized data structures. The data structures they use are a dependency graph, D, and priority queue P. These structures and the reuse of generated random number allow this algorithm to outperform on systems with larger number of species and reactions.

Next Reaction Algorithm

1. Initialize:

    (a) set initial numbers of molecules, set $\tau \leftarrow 0$, generate a dependency graph G;

(b) calculate the propensity function, a_i, for all i;

(c) for each i, generate a putative time, τ_i, according to an exponential distribution with parameter a_i;

(d) store the τ_i values in an indexed priority queue P.

2. Let μ be the reaction whose putative time, τ_μ, stored in P, is least.

3. Let τ be τ_μ.

4. Change the number of molecules to reflect execution of reaction μ. Set $t \leftarrow t+\tau$.

5. For each edge (μ, α) in the dependency graph G,

   (a) update a_α;

   (b) if $\alpha \neq \mu$, set $\tau_\alpha \leftarrow (a_{\alpha,old}/a_{\alpha,new})(\tau_\alpha - \tau) + t$;

   (c) if $\alpha = \mu$, generate random number, p, according to Eq. 3.4 and set $\tau_\alpha \leftarrow p + t$;

   (d) replace the old τ_α value in P with the new value.

6. Go to Step 2.

This thesis initially implemented the First Reaction Method and gradually added the data structures that are used in the Next Reaction Method. After analyzing the algorithms, it was found that all these algorithms need two basic objects: *Reactions* to execute, and *Molecules* that comprise the *Reactions*.

3.3 Molecules

The *Molecule* Java Interface outlines the contract *Molecule* objects must make in order to be run in the simulation. *Molecule*'s are any species that are involved in reactions such as proteins, DNA, RNA, and so forth. The most important information a *Molecule* must keep track of is how many of those molecules are in the system, otherwise known as the copy number.

The current supported molecule types include proteins, DNA, and RNA. Each class is described in the following sections. The *Molecule* API can be found in Appendix A.1.

3.3.1 Proteins

The *Protein* implementation of *Molecule* is the most generic and simple. Molecules like ribosomes and enzymes are currently modeled by this class. Polymerase are modeled by a *Polymerase* class that extends *Protein* and includes additional information such as how many polymerase are bound to a promoter.

3.3.2 DNA

The *DNA* implementation of *Molecule* contains additional information about the DNA sequence. The sequences are divided into classes such as promoters and terminator sequences, each with its own class extending from *DNA*. Additional information includes things such as sequence length, and sequence location on a DNA strand for possible display purposes or more functionality later.

In the case of a promter, the *Promoter* class also keeps track of how many of each polymerase are bound to it and also uses that information to tell how many promoter sites are free to be bound. There are also ribosome binding sites (*RBS*) that don't hold much information at the moment. T2 does not currently simulate mRNA transcription regulation, thus the *RBS* class doesn't need to hold much information.

3.3.3 RNA

The *RNA* implementation of *Molecule* is currently very similar to that of the *Protein* class. Again, since T2 does not model RNA regulation, *RNA* currently does not need to hold any more information than is necessary to implement *Molecule*.

3.4 Reactions

The other Java Interface, the *Reaction* Interface outlines the contract *Reaction* objects must make in order to be run in the simulation. The most important information a *Reaction* must know is the next time the *Reaction* will be executed and how each

Molecule involved is affected by the reaction. A *Reaction* updates all *Molecule* objects involved each time it is executed.

The current supported reactions are enzyme and molecule degradation, mRNA transcription, polymerase binding and unbinding, and mRNA production. These reactions can be categorized into three general classes: Protein-Protein Interaction, DNA-Protein Interactions, and RNA-Protein Interactions. Each class of events are further described below.

3.4.1 Protein-Protein Interactions

Events such as enzyme and molecule degradation constitute as Protein-Protein interactions. Molecule degradation events model the natural degradation of molecules while enzyme degradation events model degradation of one protein caused by another protein. The execution of these events typically results in the copy number of the molecules involved to be decreased.

3.4.2 DNA-Protein Interactions

DNA-Protein interactions include the polymerase binding/unbinding and mRNA transcription events. Polymerase binding events take a polymerase and a promoter and produce a bound polymerase and bound promoter. The polymerase unbinding event does the opposite. The mRNA transcription events take a bound promoter bound to the right polymerase, and create an mRNA while unbinding the polymerase and the promoter.

3.4.3 RNA-Protein Interactions

RNA-Protein interactions currently only include the mRNA production event. The event takes a ribosome and a mRNA molecule and produces a protein molecule.

3.4.4 Possible Extensions

The Reaction Interface can be used to create events that are quite different from a traditional chemical reaction. For example, Tabasco was written to model T7 bacteriophage entering a bacterium. These events can be modeled by creating a new *T7DNA* class that extends *DNA*. The *T7DNA* can be brought into the system by first creating a new protein that pulls in the *T7DNA*. Then, a new event is created that takes the new protein and the *T7DNA* and produces a more exposed *T7DNA*. All sorts of more complex functionality can be modeled in a similar fashion, by creating the necessary new *Molecule*'s and *Reaction*'s.

3.5 Input and Output

Input to the simulation is through an XML file. The XML format is based off the standards set by Systems Biology Markup Language (SBML) [9]. Although the general format of XML input for T2 is the same as Tabasco, there are several differences. The simulator is loaded by first parsing the XML and inputing all the relevant *Molecule* objects. The XML is then parsed again, and the relevant *Reaction* objects are added. All information on the *Molecule*'s and *Reaction*'s involved and how they interact are in the XML.

The XML is parsed by a static class, *XMLTabasco*. When new functionality is desired, not only must the necessary *Molecule*'s and *Reaction*'s be created, but *XMLTabasco* and the XML file format must be modified as well. An example of an XML input file can be seen in Appendix B.

At any point in the simulation, a text version of the simulation can be exported. The format of the simulation is such that the state of the simulation is recorded in each row while columns hold the values of each Molecule object. Such a format can be easily imported into spreadsheet programs where the data can be analyzed and graphed. The format is also consistent with a visualization program written alongside Tabasco. An example of a text output can be seen in Appendix C.

3.6 Implementation

Both the back end and the front end GUI was implemented in Java. The back end implementation follows the architecture and algorithm detailed in the previous sections. The goal for the GUI was to give the user intuitive interaction with the simulation and effectively display the necessary information. The GUI is organized into three general areas: the menu bar, the control panel, and the display area.

The menu bar contains *File* options: Load, Close, and Quit. The options New and Save are not implemented because models are created outside T2. The only *Export* option is to save the contents of the *Text Output* pane to a text file. The text file can be named and saved to a location desired by the user.

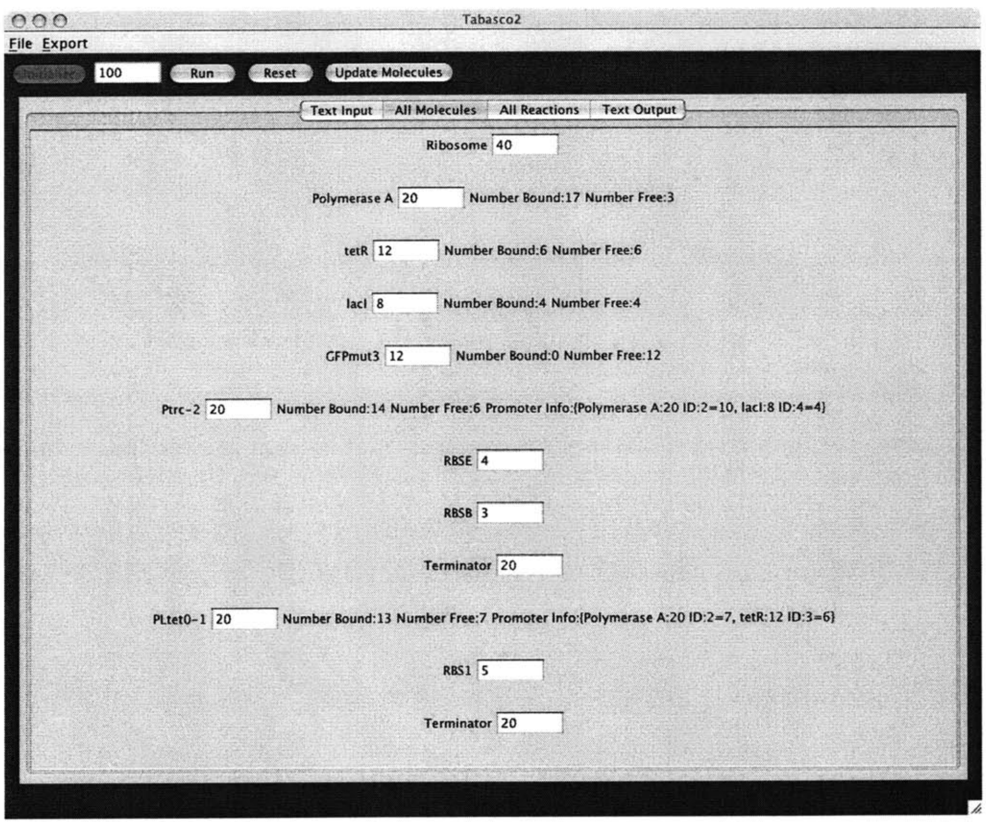

Figure 3-2: All Molecules Panel displays the details of all *Molecule* objects involved.

Right below the menu bar is a set of buttons and text fields that control and allow interaction with the simulation:

Initialize initializes the data once loaded before simulation begins. A *Reaction* dependency map is created and all the initial τ values are calculated.

TextPanel indicates the number of iteration the algorithm is to be run when the run button is clicked.

Run iterates through the simulation algorithm for the specified number of times.

Reset sets all molecule copy numbers to their original values.

Update Molecules assigns the copy number value in the *All Molecules* pane to each of the *Molecules*.

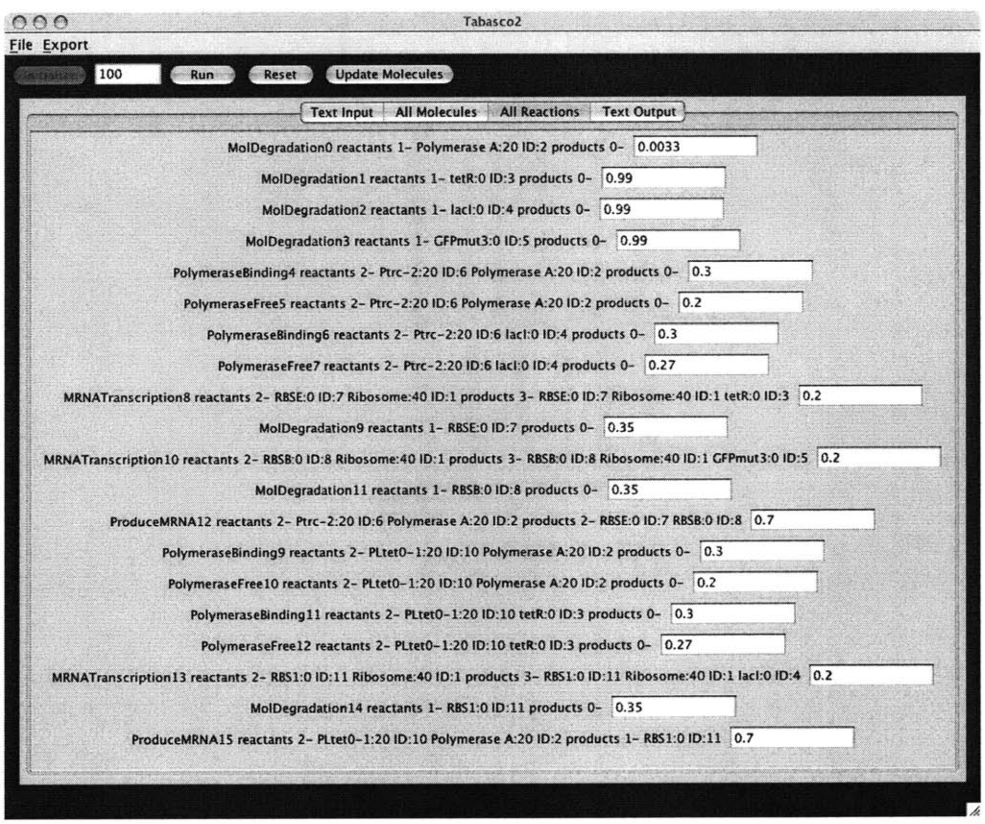

Figure 3-3: All Reaction Panel displays the details of all *Reaction* objects involved.

In order to maximize the information in the viewing area, a tabbed pane was used. Each pane shows a different set of information. If more detailed information such as a DNA graphic is desired, additional panes can be easily added. There are currently four panes:

Text Input pane displays the all the *Molecule*'s and *Reaction*'s inputed into the system in a text format.

All Molecules pane displays the name, copy number, and additional information for each *Molecule*. The copy number is displayed in a TextPanel. Changing the value and clicking the Update Molecule button will have the simulation use the new values when run again (Fig. 3.6).

All Reactions pane displays the name, reactants, products, and rate for each *Reaction* (Fig. 3.6).

Text Output pane displays the state of each output after each reaction (Fig. 3.6).

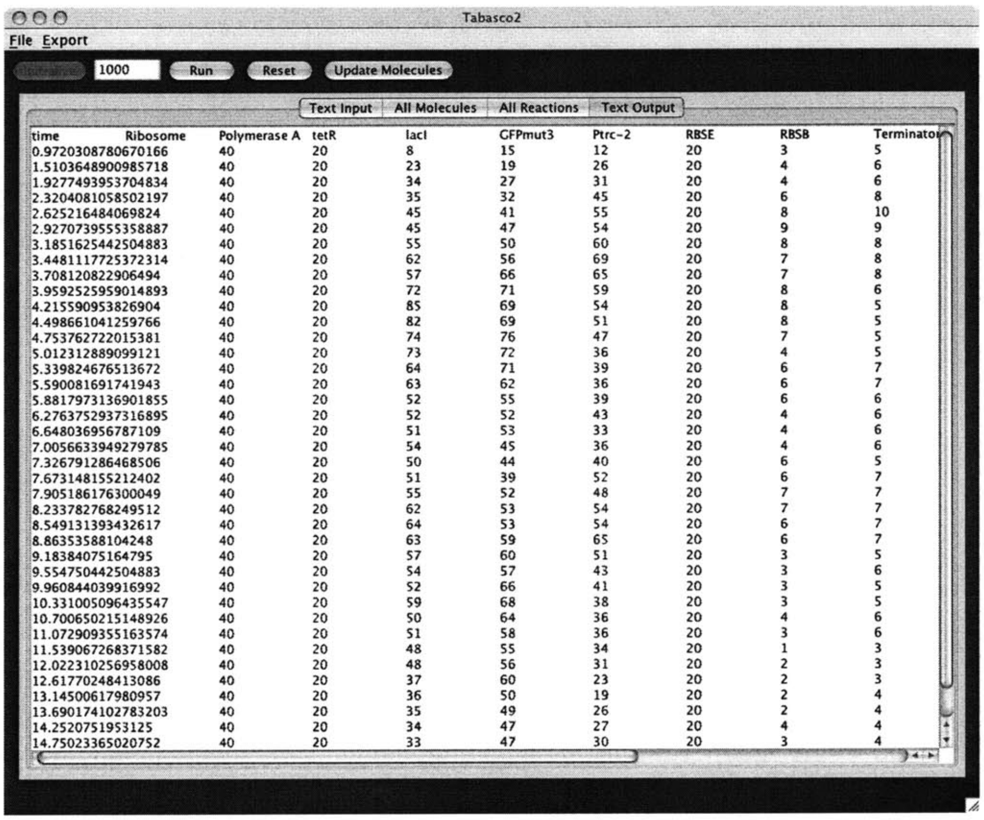

Figure 3-4: Text Output Panel displays the state of the simulation across time as text.

Chapter 4

Modeling Biology

In order to gain an understanding of how effective the simulator is at modeling real systems, real observations from a synthetic cell are compared to the results of simulating the same cell system. This chapter presents a paper that constructed a genetic toggle switch in Escherichia coli, how the cell was modeled in the simulator, and the results of that simulation.

4.1 Constructing a Genetic Toggle Switch

In 2000, Gardner, Cantor, and Collins announced that they had constructed a genetic toggle switch in Escherichia coli, marking one of the first times a synthetic, bistable gene-regulatory network was successfully created [12]. The implications of the potential practical uses of a toggle switch are quite significant, as the toggle switch is effectively a synthetic addressable cellular memory unit, a cellular bit.

The toggle switch design used by Gardner et. al. is composed of two repressors and two promoters, where each promoter is inhibited by the repressor that is transcribed by the other promoter (Fig. 4.1)[1]. This particular design was chosen for its simplicity and robustness. If Promoter 1 is transcribing Repressor 2 then Promoter 2 is most likely not transcribing Repressor 1 and the Reporter. Therefore, there are two distinct states; one where Promoter 1 is transcribing Repressor 2, and the other

[1]Figures 4.1, 4.2, 4.3, 4.4 taken from Gardner(2000) [12].

where Promoter 2 is transcribing Repressor 1 and the Reporter.

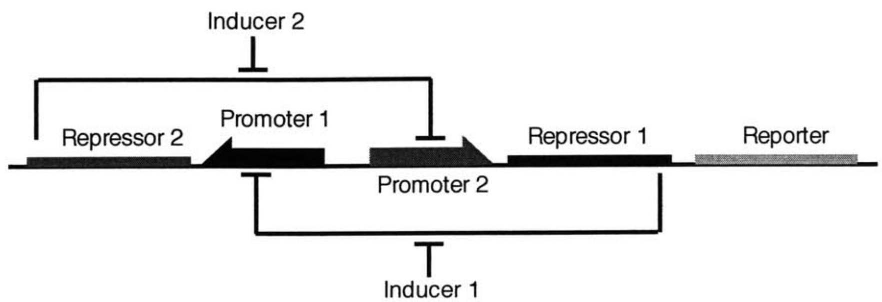

Figure 4-1: Toggle switch design. Repressor 1 inhibits transcription from Promoter 1 and is induced by Inducer 1. Repressor 2 inhibits transcription from Promoter 2 and is induced by Inducer 2.

The cell is flipped between stable states by either chemical or thermal induction. A chemical or temperature breaks down a certain Repressor, preventing it from repressing a promoter. For example, by introducing a chemical that breaks down Inducer 2 in Figure 4.1, more polymerase are likely to bind to Promoter 2 and transcribe Repressor 1 and Reporter. This, in turn creates more Inducer 1 and prevents Promoter 1 from being transcribed.

In order to tell which state a cell is in, the Reporter eventually transcribes green fluorescent protein, (GFP)mut3, which can be optically measured in a cell. By measuring the degree of fluorescence, one can determine which state a group of cells is in.

The behavior of the toggle switch can be expressed in the following equations:

$$\frac{du}{dt} = \frac{\alpha_1}{1 + v^\beta} - u \tag{4.1}$$

$$\frac{dv}{dt} = \frac{\alpha_2}{1 + u^\gamma} - v \tag{4.2}$$

where u and v are the concentrations of Repressor 1 and 2 while α_1 and α_2 are parameters that describe the effective rate of synthesis of Repressor 1 and 2. β and γ are parameters that describe the how well the promoter gets repressed, the cooperativity of the promoter.

The rates, α_1, α_2, β and γ, must be balanced in order for bistability to exist. Figure 4.2 geometrically shows the relationship between the rates. Both repressors must be transcribed at a similar rate, otherwise, a mono-stable system occurs. For example, if Repressor 1 barely gets transcribed, even if Repressor 2 is broken down temporarily, once the transient chemical is removed, Repressor 2 will get transcribed again, and block Repressor 1. Since proteins continuously break down, it is important to have positive similar rates of transcription.

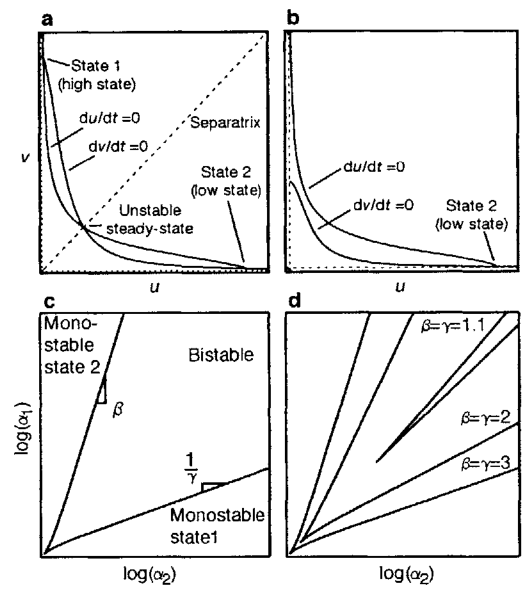

Figure 4-2: Geometric structure of the toggle equations. **a**, A bistable toggle network with balanced promoter strengths. **b**, A monostable toggle network with imbalanced promoter strengths. **c**, The bistable region.**d** Reducing the cooperativity of repression.

4.2 Modeling a Genetic Toggle Switch

Gardner created several different plasmids using different combinations of promoters and repressors. There are two major different types, pTAK, which is toggled by a chemical and heat, and pIKE, which is toggled by only chemicals. Since T2 does not simulate fluctuation in temperature, the pIKE plasmid was modeled. In Figure 4.3, the pIKE plasmid uses PLtet0-1 for P1 and tetR for R1. The strength of how well a gene was transcribed was regulated by switching in and out different ribosome binding sites.

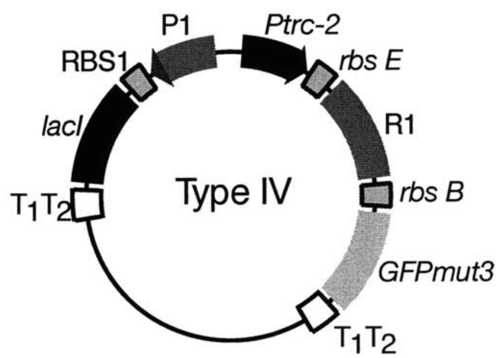

Figure 4-3: Toggle switch plasmid. Promoters are marked by solid rectangles with arrowheads. Genes are denoted with solid rectangles. Ribosome binding sites are and terminators (T1T2) are denoted by outlined boxes.

On the DNA level, the pIKE plasmid was modeled by creating two sets of *DNA SYSTEMS*, each with a *Promoter*, one more more *RBS* sites, and a *Terminator*. The RNA was modeled by creating *mRNA* whenever certain RBS sites were transcribed. The mRNA transcription was modeled with the presence *ribosome* molecules. In addition, two proteins were introduced to the system as enzymes to break down certain polymerase. IPTG was a chemical used to break down the repressor lacI and aTc was a chemical used to break down the repressor tetR.

4.3 Experimental Results

In order to test the bistability of the engineered cell, Gardner grew the cells in several different environments. First, to induce the cells to produce GFP, Gardner grew the cells in a medium filled with IPTG, which degrades the polymerase lacI. This forced the majority of the cells to produce large amounts of GFP. After some time, the IPTG was removed from the medium. Those cells that were bistable continued to produce GFP at a high level, while the cells that were monostable dropped their GFP expression. Once stabilized, aTc was introduced to the medium which degrades tetR, allowing more lacI to be transcribed resulting in the reduction of tetR and GFP expression. Figure 4.4 summarizes the data from a cell displaying bistability. The GFP expression is measure on the y axis while periods of chemical or thermal induction are shown in grey shading.

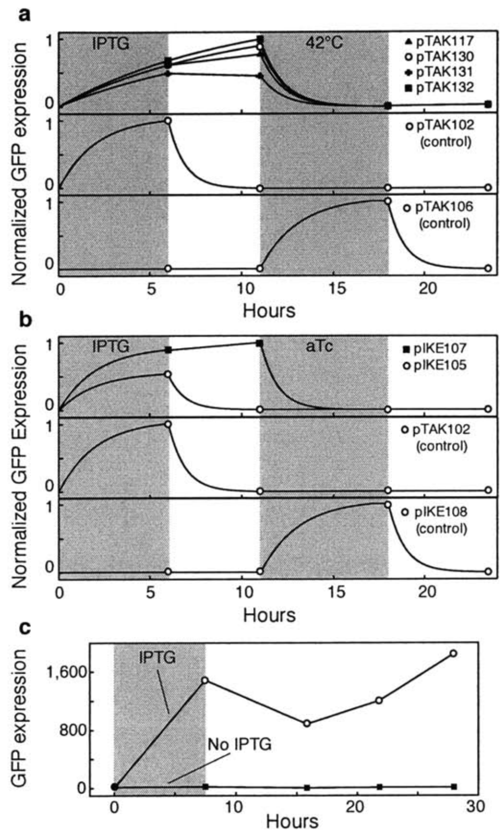

Figure 4-4: Demonstration of bistability. The grey shading indicates periods of chemical induction. **a**, pTAK toggle plasmids and controls. **b**, pIKE toggle plasmids and controls. **c**, Demonstration of the long-term stability of the separate expression states.

The same chemical inductions were simulated in T2 by introducing new molecules using the buttons in the control panel. Figure 4.5 shows 20 IPTG molecules induced in the system at the beginning of the simulation for 5000 steps. During the first 5000 steps, tetR and GFP levels shoot up while lacI levels remain stagnant. IPTG is removed for the next 5000 steps and the tetR and GFP levels remain high while the lacI levels remain low. Then, 20 aTc molecules are induced in the system for the next 10000 steps. During this transition period, the tetR molecules are being degraded by the aTc resulting in a sharp decline in the number of tetR molecules. The sharp decline results in an increase of lacI molecules and then a decrease in GFP levels. After 10000 steps, aTc is removed and the simulation is run for another 10000 steps. During this last period, the lacI levels remain high while the tetR and GFP levels remain low. Chemical induction has successfully switched the state of the cell.

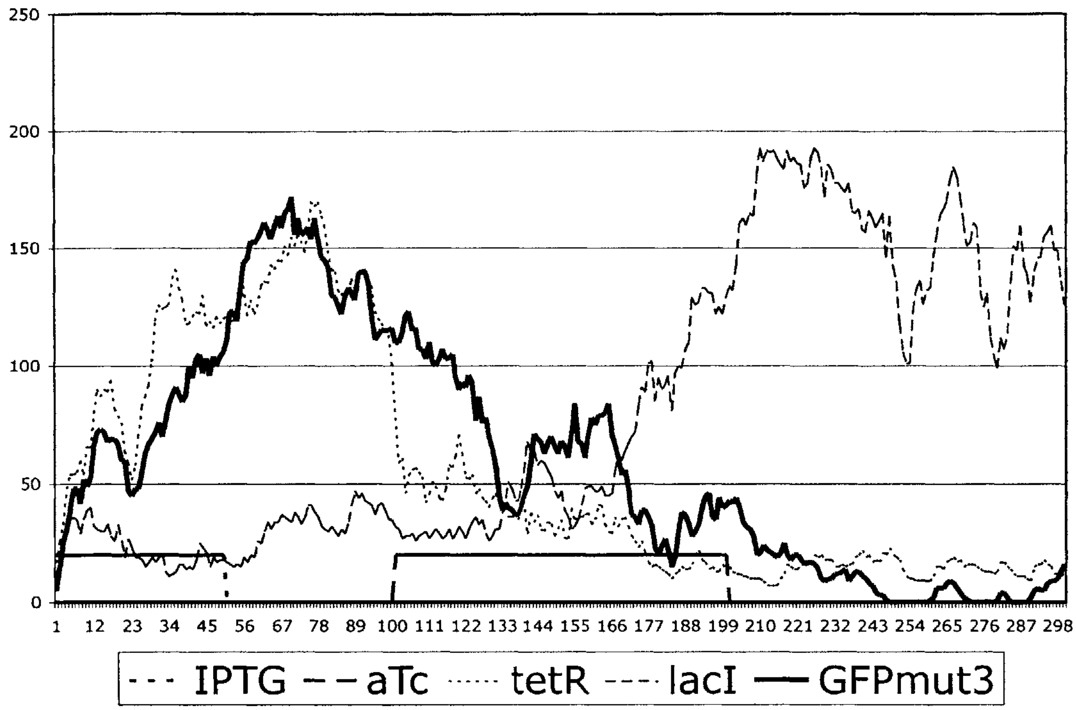

Figure 4-5: Demonstration of bistability. GFP levels go down, and stay down after the transient input of aTc.

In order to demonstrate a state of monostability, the rates of transcription and promoter repression (α_1, α_2, β, and γ in Eq. 4.1) are changed such that the system becomes a monostable system as graphically seen in Figure 4.2. Once the system rates are changed, the same chemical induction is induced. The results are the same until the aTc is removed in the final period. Instead of lacI levels staying high and GFP levels staying low, lacI levels decrease while tetR and GFP levels increase again. The system with unbalanced rates refuses to remain in a state of low GFP expression (Fig. 4.6).

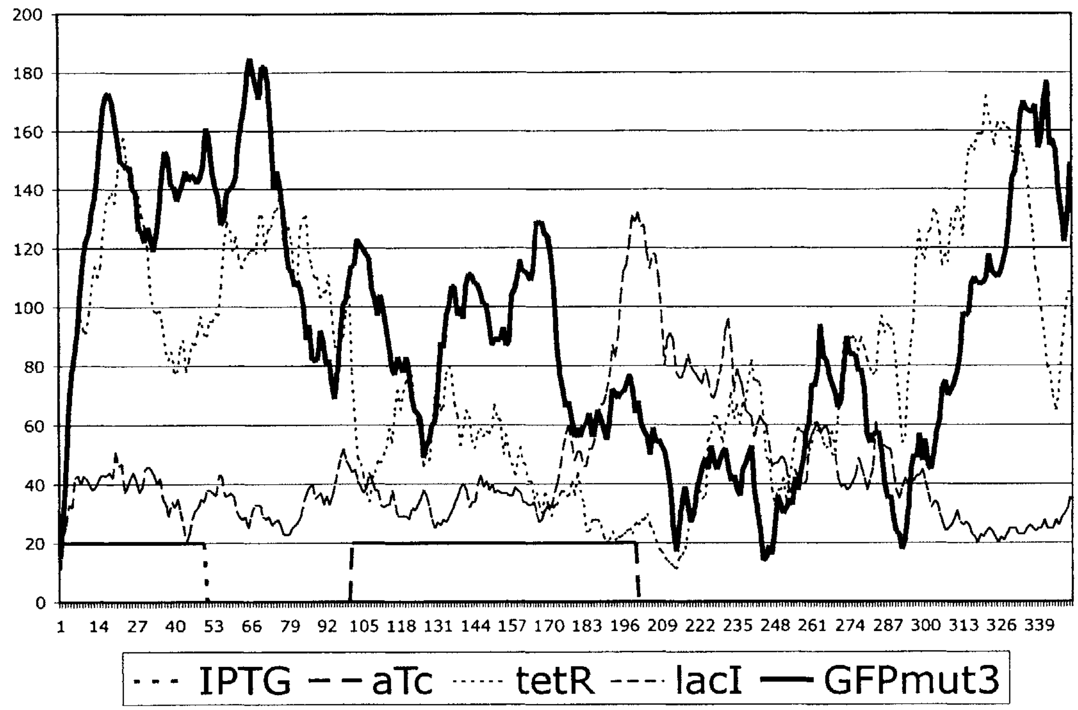

Figure 4-6: Demonstration of monostability. GFP levels go back up even after the transient input of aTc.

Chapter 5

Discussion

This chapter discusses the results of modeling a real synthetic cell and analyzes the design and implementation of the simulator. First, is an analysis of how well T2 modeled the cellular toggle switch built by Gardner. Then, there is a discussion on the assumptions that were made and how they affect the model. Next is a discussion on the rates of reactions used in the simulation. Finally, there is a discussion about the choices in architecture design.

5.1 Analysis of Modeling Results

T2 correctly modeled the general behavior and trends of a simple cellular toggle switch. In this sense, T2 was successful. The simulation demonstrated bistability given balanced rates and showed monostability when those rates were changed, just like the synthetic cellular toggles. Although the trends were correct, the actual number of molecules are probably far off. One indication that the quantities are inaccurate is that the change in GFP levels were more extreme in the simulation than in the real cell, most likely due to missing regulatory features and inaccurate reaction rates. In order to achieve better results, realistic reaction rates as well as possible additional modeling of more functionality must be used.

Initial attempts at modeling the system resulted in unreasonable molecule levels due to the model having unlimited resources. The model was then given additional functionality to simulate limited resources and some reaction rates were adjusted. In order to model a limited number of amino acids, the cell was given a protein production limit. Resources such as amino acids were all lumped into one category where a uniform use of amino acids were assumed, both in type and quantity. Each time a mRNA is transcribed, it decreases the resource while each time a protein is degraded, the resources increase. This modification resulted in more reasonable protein levels.

The ability of T2 to model a simple system gives us an idea of that what functionality is needed to simulate a cell. In other words, this experiment reveals what is the minimum functionality that must be used to effectively simulate a simple system. General trends and cellular responses can be modeled by simulating simple polymerase binding, DNA transcription, mRNA transcription, protein degradation, resource limitations, and a DNA system. System trends and behavior can be simulated without modeling extremely complex interactions. Even the relationships between reaction rates can be analyzed and simulated. The results give support to the top down approach which proposes that more can be learned by gradually adding functionality rather than trying to account for as much detail as possible.

5.2 Assumptions

Stochastic algorithms are not without their own assumptions. In particular, with biological systems, stochastic algorithms assume:

- Molecules obey the laws of quantum mechanics.
- Molecules obey Newton's laws of motion.
- Solution is well mixed: nonreactive collisions occur more often than reactive collisions.

All three assumptions are needed if one wants to represent a system simply by the number of each molecule [3]. However, the first two assumptions do not hold when dealing with complex macromolecules such as biological proteins, long time scales, or interactions between several different types of molecules. Only when dealing with simple systems and molecules

The third assumption is certainly not held in eukaryotes, where the nucleus, organelles, and intracellular transport system create highly location specific concentrations. Mixed solutions cannot be assumed for prokaryotes either. Many prokaryotes respond to gradients of concentration. Such gradients that enable chemotaxis may cause the third assumption to break down.

It is unknown to what extent bacterium fit the assumptions needed for stochastic modeling. Bacterium may or may not be well mixed with simple enough molecules. Simple systems such as toggle switches do seem to be simple enough. However, stochastic modeling beyond yeast or simple bacterium will most likely prove to be unsuccessful. One possible hack would be divide large complex systems into smaller simpler systems where assumptions are maintained. Using such a strategy, even complex eukaryotes might be simulated by separating the nucleus and organelles into subsystems that can be modeled stochastically.

5.3 Rate of Reaction

The rate of stochastic reactions are related to, but different from macroscopic or deterministic rate constants (Sec. 2.3.2). Therefore, determining the stochastic rate of reaction is a large part of modeling a system. It was shown that the rates are very important in determining system behavior. Unbalanced rates can cause a bistable system to become monostable (Sec. 4.3). Some rates are completely unreasonable and results in model behavior that is unrealistic.

The degree of sensitivity to rate constants is unknown, although after changing several rates, it seems that the simulations are quite sensitive. Some values are probably easier to determine than others. Rates such as molecule degradation are most likely to be within a small range of possible values whereas other rates may be from wider range of values. It must also be questioned whether or not the rates are constant. Certainly, with the presence of complex molecules, there will be changes in reaction rates. However, these changes are the result of additional interactions that are unaccounted for, such as temperature changes, pH level changes, and molecules or proteins that somehow affect the reaction. Identifying these underlying interactions will enable varying reaction rate to become more predictable. Hopefully, most reaction rate variations can be accounted for by modeling the interactions that cause the change.

5.4 Reflection on Architecture

The main goals in designing the application was modularity and usability. These goals were met with various levels of success. As good as the initial design for modularity was thought to be, flaws and room for improvement were found during implementation. Some of these flaws were improved upon as decisions were made, while others were found only after much of the implementation was finished.

One of the first design decisions was to determine if *Molecule* objects were allowed to keep track of states. For example, a promoter is either free or bound to a polymerase. Likewise, a polymerase is either floating about or bound to DNA. Many reaction rates depend not on the total number of molecules available, but the number of molecules that are bound to certain other molecules. Allowing tracking of states would complicate the interfaces and might result in greater difficulty when expanding functionality. However, if states are not kept, then a new *Molecule* must be created for each possible bound combination. For example, if 3 different promoters and 3 different polymerase all interact with each other, than there are 9 different promoter-polymerase combinations that can exist assuming that only single polymerase binding are being modeled. It was decided that the added complexity of allowing state tracking was better than having the number of *Molecule* objects baloon.

Another decision was whether or not to limit the number of reactants to two. Almost all real systems can be reduced to elementary reactions that have at most two reactants. Since the reaction rates are calculated using the number of reactants, it is beneficial to always have two reactants in order to keep the reaction rate consistent. However, allowing various numbers of reactants gives the user much greater flexibility. The user may not want to break up a reaction and instead lump the reaction together for convenience. It is important though, that the user be careful in choosing reaction rates that take into account the number of reactants.

During implementation, it was found that some processes that should have been done in T2Simulator were being implemented in the classes implementing *Reaction*. For example, all *Reaction*'s were required to keep track of which other *Reaction*'s they affected, creating a dependency graph for the simulator. However, the dependency graph was used only by the simulator in order to speed up the algorithm by decreasing the number of reactions that needed to be updated for each execution. Therefore, a map was created in T2Simulator that kept track of the dependency graph instead of the *Reaction* subclasses. Moving that implementation allowed a more simple *Reaction* interface which is important because the contract that the interface outlines should be as simple as possible so that the interface can be easily implemented.

When using T2, it was found that creating and modifying models required more effort than expected. Adding functionality such as limited cell resources could not be done without programming new *Reaction*'s. Although most functionality will require coding, it was hoped that the user would have enough flexibility to create some functionality by just using the XML input. Unfortunately, much of the design was focussed on allowing a programmer to expand functionality and little attention was paid to giving a user as much modularity as a programmer. Only after much of the implementation was finished, was it realized that more specification of the XML input should have been done. By no means is the XML input totally unmodifiable, but more attention to its format would have given the application significantly greater usability.

5.5 Summary

The most salient observations made and lessons learned can be summarized as follows:

- Complex functionality is not necessary to model trends in simple systems. Basic functionality is the minimum requirement for effective simulation of simple systems.

- Rates of reaction are extremely important to a system. So much so, that instead of designing an overall system, synthetic biologist may need to spend the bulk of their effort trying to determine the correct reaction rates.

- The best way to enhance functionality is to put as much control as possible into the hands of the user. Creating a modular program does not necessarily result in improved usability or functionality. More transparency in the inputs gives users more control.

Chapter 6

Future Work

This chapter outlines possible future projects related to this thesis. The underlying theme is making this thesis work more useful to researchers. Greater usefulness can come from improvements such as a more accurate simulation, ease of use, and ease of access.

6.1 Finding Rates

Finding the true rates of reactions is one of the most important tasks in simulating cellular systems. Even the behavior of a simple cellular system was found to be heavily dependent on the rates of the reactions. Since stochastic reaction rates are different from deterministic reaction rates, the rates to be used are difficult to determine. Although many projects have attempted to model systems, few if any have attempted to find the true rates of reaction.

Projects designed to find stochastic reaction rates would be extremely useful for users of all stochastic simulators. The most likely way to approach the problem would be to take several different simple cellular systems and create the corresponding simulating models. The next step would be to find the reaction rates that best fit the models to the real systems. A possible method in finding the rates might be to use machine learning.

One would hope that the reaction rates are constant and consistent across different models. This way, the stochastic reaction rates found from one system can be applied to another system. If the reaction rates don't seem to be constant, then one would need to find the underlying reason for the variation. In either case, finding stochastic reaction rates is still a crucial part in modeling.

6.2 Adding Functionality

There are an innumerable amount of choices to consider when adding more functionality. However, in order to create a program that is tailored for synthetic biologists, it is important to add functionality that increases the flexibility of the simulator. One feature would be to create a more robust XML input format. The current format is extremely specific and cannot be altered without changing underlying code. A format that is more simple and modular would give the the user more choices in the simulations. The key is to provide good tools to the user so that the user can create simulations as complex or simple as they want.

6.3 Ease of Access

The usefulness of a program is often measured by the number of people who use it. Programs like the DARPA headed BioSPICE has the potential to be used by many researchers. Future work might prove to more useful if done within a large initiative. Such work would have exposure to a larger number of users and hopefully make a greater impact on the synthetic biology community. If the XML input format is consistent with the Systems Biology Markup Language (SMBL) definitions, then the application may be eligible to be linked to from the SBML website, *http://www.sbml.org*, which lists over 75 software systems that support SBML.

Chapter 7

Contributions

In this thesis, I have:

- designed an architecture that is expandable and modular by separating code that performs stochastic algorithms from code that represents the data.
- allowed the simulation that the user can use interactively, allowing the user to pause a simulation to add or remove transient or permanent inputs to the system.
- created an GUI that displays the state and history of the simulation and also allows loading and interaction. The GUI is easily expandable to show more detailed information.
- demonstrated that the simulator is successful at modeling simple synthetically engineered cells, in particular a cell toggle switch design.

In addition to the contributions above, I have found a range of stochastic reaction rates that are effective in modeling a simple toggle switch. I argue that finding the correct stochastic rates are important to any future work because the simulations are highly dependent upon the rates of reaction. Finally, I claim that the best design for a simulator is to place as much functionality as possible in hands of the user with the fewest number of tools.

Appendix A

Interfaces

A.1 Molecule.java

```
/*
 * Created on Jan 4, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
package tabasco2;

/**
 * author dankim
 *
 * Reaction Interface defines the contract that a reaction must meet in order to be
 * simulated by the simulator.
 */
public interface Molecule {
        /**
         *
         * return unique ID of Molecule
*/
        public int getID();

        /**
         * used to display the Molecule in the GUI
         * return the name of the Molecule. i.e. PolymeraseA.
*/
        public String getName();

        /**
         * used by the GUI to display the molecule.
```

```
     * return type of the Molecule, usually an abbreviated form of the class name.
*/
    public String getType();

    /**
     * used by the reset function
     * return number of molecules the simulation was initialized with
*/
    public int getInitCopyNumber();

    /**
     *
     * return number of molecules
*/
    public int getCopyNumber();

    /**
     *
     * return the organism ID number
*/
    public int getOrganism();

    /**
     *
     * param inc the number you want to increase or decrease the number of molecules
*/
    public void updateCopyNumber(int inc);

    /**
     *
     * param num the number you want to set the number of molecules
*/
    public void setCopyNumber(int num);

    /**
     * Reset copyNumber to the original copyNumber and unbind all proteins.
     *
     */
    public void reset();

    /**
     *
     * return String version of Molecule.
*/
    public String toString();
```

```
}
```

A.2 Reaction.java

```
/*
 * Created on Jan 8, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
package tabasco2;

import java.util.List;

/**
 * author dankim
 *
 * Reaction Interface defines the contract that a reaction must meet in order to be
 * simulated by the simulator.
*/
public interface Reaction {

        /**
         * returns the unique integer ID of the reaction.
         */
        public int getID();

        /**
         * used to display the reaction in the GUI
         * return the name of the reaction. i.e. PolymeraseAtoPromoterX.
*/
        public String getName();

        /**
         * used to cast objects in order to call more specific methods in the GUI only.
         * return type of the reaction, usually an abbreviated form of the class name.
*/
        public String getType();

        /**
         * used by the GUI to display the rate of the reaction.
         * return reaction rate of the reaction.
*/
        public double getRC();
```

```
/**
 * sets the reaction rate of the reaction.
 * used when changing or updating the reaction rate from the GUI.
 */
public void setRC(double rc);

/**
 * used only by the simulator.
 * return tau at which the reaction will occur next.
*/
public double getTau();

/**
 * calculates the the new 'a' value, usually by multiplying the reactants
 * and the reaction rate.
 */
public void calculateA();

/**
 * calculates the tau at which the reaction will occur next by using the 'a' value.
 */
public void calculateTau(double random, double t);

/**
 * executes the reaction and updates the Copy Number of each Molecule involved
 * or blocks certain reactions from occuring or moves molecules along DNA
 */
public void execute();

/**
 *
 * return List of Reactions that this reaction depends on.
*/
public List dependsOn();

/**
 * currently returns a union of reactants and products.
 * doesn't take into account of catalytic reactions because
 * it is difficult to tell which reactions are catalytic
 * the extra molecule shouldn't be a problem (both time and correctness wise)
 * return a list Molecules that are affected by reaction
*/
public List affects();
```

```
    /**
     * prints the reaction using System.out.println()
     *
     */
    public void printExecutedRxn();

    /**
     *
     * return a String version of the reaction.
*/
    public String toString();
}
```

Appendix B

XML Input Format

```
<REQUEST><EXECUTE-SIMULATION random_seed="69077" interval_for_output="10">
<CELL organism_id="1">
<RIBOSOME name="Ribosome" organism_id="1" n="40" footprint="10" speed="40"
 id="1" />

<POLYMERASE name="Polymerase A" organism_id="1" speed="40" n="20" id="2"
 polydeg="3.3e-3"/>
<POLYMERASE ID="3" name="tetR" organism_id="1" speed="0" protein="tetR"
 n="0" id="3" polydeg=".99"/>
<POLYMERASE ID="4" name="lacI" organism_id="1" speed="0" protein="lacI"
 n="0" id="4" polydeg=".99"/>
<POLYMERASE ID="5" name="GFPmut3" organism_id="1" speed="0"
 protein="GFPmut3" n="0" id="5" polydeg=".99"/>
<PROTEIN id="13" name="IPTG" organism_id="1" n="0" deg="3.3e-3"/>
<PROTEIN id="14" name="aTc" organism_id="1" n="0" deg="3.3e-3"/>

<DNA_SYSTEM n="20" entry_offsite="2990" genome_length="2990"
 entry_rate_constant="50000">

<PROMOTER startsite="49" name="Ptrc-2" stop="55" organism_id="1"
 protein="Ptrc-2" start="1" id="6">
<POLYMERASE runoff_percent=".75" aelong="10" aoff=".2" polymeraseID="2"
 aon=".3" ainiton=".7" />
<POLYMERASE runoff_percent=".75" aelong="10" aoff=".27" polymeraseID="4"
 aon=".3" ainiton=".7" />
</PROMOTER>
<RBS name="RBSE" start="55" organism_id="1" strength=".2" mrnadeg=".35"
 elongstepsize="1000" stopsite="689" startsite="73" stop="69" protid="3"
 initstepsize="8" id="7"/>
<RBS name="RBSB" start="55" organism_id="1" strength=".2" mrnadeg=".35"
 elongstepsize="1000" stopsite="689" startsite="73" stop="69" protid="5"
 initstepsize="8" id="8"/>
```

```
<TERMINATOR name="Terminator" stopsite="735" stop="810" organism_id="1"
 start="730" id="9">
<POLYMERASE ID="2" efficiency="0.90" />
</TERMINATOR>

</DNA_SYSTEM>

<DNA_SYSTEM n="20" entry_offsite="2990" genome_length="2990"
 entry_rate_constant="50000">

<PROMOTER startsite="853" name="PLtetO-1" stop="859" organism_id="1"
 protein="PLtetO-1" start="810" id="10">
<POLYMERASE runoff_percent=".75" aelong="10" aoff=".2" polymeraseID="2"
 aon=".3" ainiton=".7" />
<POLYMERASE runoff_percent=".75" aelong="10" aoff=".27" polymeraseID="3"
 aon=".3" ainiton=".7" />
</PROMOTER>
<RBS name="RBS1" start="859" organism_id="1" strength=".2" mrnadeg=".35"
 elongstepsize="1000" stopsite="2001" startsite="874" stop="873" protid="4"
 initstepsize="8" id="11"/>
<TERMINATOR name="Terminator" stopsite="2007" stop="2082" organism_id="1"
 start="2002" id="12">
<POLYMERASE ID="2" efficiency="0.90" />
</TERMINATOR>

</DNA_SYSTEM>

<ENZYME_DEGRADATION name="lacIDegradation" organism_id="1" id="20" a="0.1">
<REACT id="4"/>
<REACT id="13"/>
<PROD id="13"/>
</ENZYME_DEGRADATION>
<ENZYME_DEGRADATION name="tetRDegradation" organism_id="1" id="21" a="0.1">
<REACT id="3"/>
<REACT id="14"/>
<PROD id="14"/>
</ENZYME_DEGRADATION>

</CELL>
</EXECUTE-SIMULATION>
</REQUEST>
```

Appendix C

Text Output Sample

```
time ProtB ProtD ProtF ProtA ProtC ProtG ProtE
2.86773024527065E-6 299 350 399 250 101 150 200
3.650849612313323E-5 298 350 398 250 102 150 200
4.0357732359552756E-5 298 351 398 249 102 150 200
4.8411533498438075E-5 297 351 397 249 103 150 200
6.770170148229226E-5 296 351 396 249 104 150 200
9.082914766622707E-5 296 351 396 250 104 149 199
1.019184710457921E-4 295 351 395 250 105 149 199
1.0865218791877851E-4 294 351 394 250 106 149 199
1.1234092380618677E-4 294 352 394 249 106 149 199
1.2315464846324176E-4 293 352 393 249 107 149 199
```

Bibliography

[1] Darpa biocomp. http://www.darpa.mil/ipto/programs/biocomp/, 2005.

[2] Dizzy. http://labs.systembiology.net/bolouri/Dizzy/, 2005.

[3] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry*, 104(9):1876–1889, 2000.

[4] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, (22):403–434, 1976.

[5] D. T. Gillespie. *Markov Process: An Introduction for Physical Scientists.* Academic Press, San Diego, CA, 1992.

[6] Jonathan A. Goler. Biojade: A design and simulation tool for synthetic biological systems. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 2004.

[7] S. Kosuri J. Kelly, D. Endy. Tabasco simulation system, 2001.

[8] Barbara Liskov. *Program Devolopment in Java.* Addison-Wesley, 2001.

[9] Systems biology markup language. http://sbml.org/index.psp, 2005.

[10] Earl P. Solis. Stochastic kinetics in biological systems. Chemical Engineering Report on Stochastic Kinetics, 2005.

[11] Synthetic biology. http://www.syntheticbiology.org/, 2004.

[12] C. R. Cantor T. S. Gardner and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403:339–342, 2000.