# STOCHASTIC ANALYSIS OF SYNTHETIC GENETIC CIRCUITS

by

Curtis K. Madsen

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

August 2013

# The University of Utah Graduate School

## STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Curtis K. Madsen**

has been approved by the following supervisory committee members:

| | | |
|---|---|---|
| **Chris Myers** | , Chair | **May 2, 2013** |
| | | Date Approved |
| **Rajeev Balasubramonian** | , Member | **May 2, 2013** |
| | | Date Approved |
| **James Keener** | , Member | **May 2, 2013** |
| | | Date Approved |
| **Robert Kessler** | , Member | **May 2, 2013** |
| | | Date Approved |
| **Chris Winstead** | , Member | **May 2, 2013** |
| | | Date Approved |

and by **Al Davis**, Chair of

the Department of **School of Computing**

and by Donna M. White, Interim Dean of The Graduate School.

# ABSTRACT

Over the past few decades, synthetic biology has generated great interest to biologists and engineers alike. Synthetic biology combines the research of biology with the engineering principles of standards, abstraction, and automated construction with the ultimate goal of being able to design and build useful biological systems. To realize this goal, researchers are actively working on better ways to model and analyze synthetic genetic circuits, groupings of genes that influence the expression of each other through the use of proteins. When designing and analyzing genetic circuits, researchers are often interested in building circuits that exhibit a particular behavior. Usually, this involves simulating their models to produce some time series data and analyzing this data to discern whether or not the circuit behaves appropriately. This method becomes less attractive as circuits grow in complexity because it becomes very time consuming to generate a sufficient amount of runs for analysis. In addition, trying to select representative runs out of a large data set is tedious and error-prone thereby motivating methods of automating this analysis. This has led to the need for design space exploration techniques that allow synthetic biologists to easily explore the effect of varying parameters and efficiently consider alternative designs of their systems.

This dissertation attempts to address this need by proposing new analysis and verification techniques for synthetic genetic circuits. In particular, it applies formal methods such as model checking techniques to models of genetic circuits in order to ensure that they behave correctly and are as robust as possible for a variety of different inputs and/or parameter settings. However, model checking stochastic systems is not as simple as model checking deterministic systems where it is always known what the next state of the system will be at any given step. Stochastic systems can exhibit a variety of different behaviors that are chosen randomly with different probabilities at each time step. Therefore, model checking a stochastic system involves calculating the probability that the system will exhibit a desired behavior. Although it is often more difficult to work with the probabilities that stochastic systems introduce, stochastic systems and the models that represent them

are becoming commonplace in many disciplines including electronic circuit design where as parts are being made smaller and smaller, they are becoming less reliable.

In addition to stochastic model checking, this dissertation proposes a new incremental stochastic simulation algorithm (iSSA) based on Gillespie's stochastic simulation algorithm (SSA) that is capable of presenting a researcher with a simulation trace of the typical behavior of the system. Before the development of this algorithm, discerning this information was extremely error-prone as it involved performing many simulations and attempting to wade through the massive amounts of data. This algorithm greatly aids researchers in designing genetic circuits as it efficiently shows the researcher the most likely behavior of the circuit.

Both the iSSA and stochastic model checking can be used in concert to give a researcher the likelihood that the system will exhibit its most typical behavior. Once the typical behavior is known, properties for nontypical behaviors can be constructed and their likelihoods can also be computed. This methodology is applied to several genetic circuits leading to new understanding of the effects of various parameters on the behavior of these circuits.

To my best friend and the love of my life, Jennifer.

# CONTENTS

# LIST OF FIGURES

# LIST OF ALGORITHMS

# ACKNOWLEDGMENTS

I would first like to thank my advisor, Chris Myers. Chris began mentoring me when I was a freshman at the University of Utah and has since provided me with invaluable guidance and support in my time as an undergraduate and, subsequently, graduate researcher. I learned a lot about how rewarding a career in research can be from Chris, and it was primarily due to his encouragement that I decided to pursue a PhD. I will forever be grateful to him for assisting me in achieving this goal.

I would also like to thank the members of my committee: Rajeev Balasubramonian, James Keener, Robert Kessler, and Chris Winstead. In particular, I have appreciated Chris Winstead for his help and collaboration. Our extensive meetings, where we discussed multiple aspects of the algorithms and methods involved in this work, were invaluable in the advancement of this research.

I am greatly indebted to the wonderful faculty and staff of the School of Computing. I can honestly say that I have never taken a computer science course from a bad professor at the University of Utah. Additionally, I am thankful to the advising staff in the department for allowing me to focus on my work by handling many of the administrative requirements of my degrees.

Because I began working in the lab when I was a freshman, I got to know many other undergraduate and graduate students in the Myers Research Group. Among these students are Nathan Barker, Satish Batchu, Andrew Fisher, Kevin Jones, Dhanashree Kulkarni, Hiroyuki Kuwahara, Scott Little, Nam Nguyen, Tyler Patterson, Nicholas Roehner, Jason Stevens, Robert Thacker, Leandro Watanabe, and Zhen Zhang. My colleagues in the lab have provided me with a great source of inspiration, a needed outlet to discuss frustrations, and a nice distraction when the work was otherwise overwhelming.

I would not be the person I am today if it were not for my family. My wonderful parents, Kendall and Karen Madsen, instilled in me the value of education at a very early age and helped me set and achieve ambitious goals throughout my academic career. I cannot begin to express my gratitude for the unconditional love and support they have given me throughout my life, particularly these last few years as a graduate student.

Additionally, I have immensely appreciated the friendship of my siblings: Michelle, Kevin, and Laura. As their older brother, I have always been their role model; however, my success was only possible with their love and support.

Most of all, I would like to thank my best friend and the love of my life, Jennifer Gibson, for all of her love, support, and patience while I was in graduate school. I know that the many sleepless nights and long days were not only a sacrifice for me but were also shared by her. I cannot imagine how difficult it must have been for her to assist me in my demanding career choice, but I know that I am eternally grateful to Jennifer for helping me accomplish one of my lifelong goals.

This material is based upon work supported by the National Science Foundation under Grant Numbers 0331270, CCF-07377655, CCF-0916042, and CCF-1218095. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# CHAPTER 1

# INTRODUCTION

One of the great mysteries in our world is how biological systems work. Many researchers and scientists interested in finding answers to this problem have adopted the paradigm of systems biology. *Systems biology* is a field of study that focuses on explaining systems-level behavior by examining component-level interactions. Researchers taking this view attempt to identify the function of individual components in the system they are studying and use this information to explain how the functions of interconnected components can lead to the behavior that is observed by the entire system.

Another field in the realm of biology that has given researchers insight into how biological systems work is genetic engineering. *Genetic engineering* is a discipline that attempts to modify an organism's DNA in order to either determine how the modified parts affect the organism or to cause the organism to exhibit a new behavior. This modification is accomplished through the use of *recombinant DNA*, the construction of artificial DNA through combinations; *polymerase chain reaction*, the process of making copies of DNA; and *automated sequencing*, the act of checking a DNA sequence to ensure that it contains the desired sequence.

In recent years, there has been great interest in designing new useful biological systems instead of simply trying to understand how they work. In order to accomplish this goal, researchers have taken ideas from both systems biology and genetic engineering and combined them with the engineering principles of standards, abstraction, and automated construction to design and build biological systems leading to the creation of the field of *synthetic biology* [6, 23]. Some applications of this field include enabling designed biological systems to consume toxic waste [15], hunt and kill tumor cells [4], and produce drugs [73] and biofuels [7]. To realize these goals, researchers are actively working on better ways to model and analyze *synthetic genetic circuits*, groupings of genes that influence the expression of each other through the use of proteins. When designing and analyzing genetic circuits, researchers are often interested in building circuits that exhibit

a particular behavior. Usually, determining a circuit's behavior involves simulating a model to produce some time series data and analyzing this data to discern whether or not the circuit behaves appropriately. This method becomes less attractive as circuits grow in complexity because it ends up becoming very time consuming to generate a sufficient amount of runs for analysis. In addition, trying to select representative runs out of a large data set is tedious and error-prone motivating methods of automating this analysis. These problems have led to the need for design space exploration techniques that allow synthetic biologists to easily explore the effect of varying parameters in their systems and efficiently consider alternative designs of their systems.

## 1.1 Modeling and Analysis

The modeling and analysis of biological systems has been of great interest to researchers in synthetic biology. Many different models and methods have been proposed to represent and analyze genetic systems [47]. Probably the most prevalent modeling language for representing genetic networks is the *Systems Biology Markup Language* (SBML) [46]. 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.

The most common method of analyzing a biological system is to represent it as an *ordinary differential equation* (ODE) model and analyze the ODE using simulation. This analysis is typically done by translating the list of *chemical species* and *chemical reactions* into an ODE model using the *law of mass action* [83]. This type of analysis gives reasonably accurate results for models where the number of chemical species is large and continuous and reactions occur deterministically, which is the case in many biological systems. However, genetic circuits typically have very small, discrete molecule counts and reactions that fire sporadically leading to inaccurate ODE analysis results [61].

Due to their stochastic nature, genetic circuits must be simulated with stochastic analysis methods in order to yield satisfactory results. One such method developed for this task is *Gillespie's stochastic simulation algorithm* (SSA) [33] which is the cornerstone behind nearly all stochastic simulation methods involving chemical species today. At its core, the SSA is a Monte Carlo simulation algorithm that effectively deals with the small, discrete species counts and random firing times of reactions by stepping over time steps where no reactions occur.

Because simulating with the SSA can be very computationally intensive, the SSA has been further improved in a variety of ways. The original SSA has complexity $O(n)$ where $n$ is the number of reactions in the system. This complexity is due to the algorithm needing to loop over all of the reactions in each step. If the *first-reaction method* [32] is used, then the algorithm must draw a random number for each reaction in the system during this loop which can be a very costly procedure. In order to mitigate this expensive step, the *direct method* [33] has been developed and only requires two random numbers to be drawn per loop. The SSA's efficiency can be improved further by using a dependency graph and a priority queue to store the reactions as in *Gibson and Bruck's Next Reaction Method* [30]. After each step of the SSA, the algorithm only needs to update the priority queue leading to a complexity of $O(\log n)$. The efficiency of the SSA can be improved even further yielding a constant time algorithm known as the *composition/rejection* (SSA-CR) method [75]. This method utilizes a technique known as composition and rejection, which is accomplished by grouping reactions with similar propensities together (the composition step). Then, a reaction is randomly selected from one of the groups and the algorithm determines whether this reaction should be fired or a new one should be selected (the rejection step).

Sometimes, simulating only one reaction at a time still requires an exorbitant amount of simulation time. To further improve simulation efficiency, the *tau-leaping* method can be used to fire multiple reactions in each time increment [14, 35, 36]. The main problem with tau-leaping is it can be difficult to select a good value of tau. If it is too small, there is not much gain in simulation efficiency, but if tau is too large, then the simulation can give erroneous results as species counts can go below zero.

Even with tau-leaping, many systems with both fast and slow reactions, also known as stiff chemical systems, perform poorly. *Slow-scale SSA* (ssSSA) is an efficient approach to dealing with such systems [13]. It proceeds by simulating only the slow reactions and uses specially modified propensity functions to take into account the fast reactions. Although ssSSA is an approximation of the exact methods, it has been shown to decrease simulation time when compared to the exact SSA by two to three orders of magnitude, with no perceptible loss of simulation accuracy.

In order to obtain more gains in simulation efficiency, reaction-based abstractions can be applied to genetic circuits [56, 53]. By using assumptions like the *quasi-steady-state assumption* [74], the system can be simplified down to significantly fewer species and

reactions. These abstractions lead to some loss of accuracy when the abstracted model is simulated; however, as long as the abstractions are applied correctly, the gains in simulation efficiency typically greatly outweigh this loss as the loss is usually very small to the point of being insignificant. In addition, as models of biological systems grow in complexity, abstractions become absolutely necessary in order to enable simulation and analysis techniques to be applied to the models.

Stochastic simulation of genetic circuits can be extremely time consuming even when considering various improvements to the SSA. In addition, simulation alone does not explicitly give the probability of the system reaching a particular state in its execution prompting researchers to perform many simulation runs and to compare the data and record which states the network ended up in. If a particular event rarely occurs in the system, then this method requires an enormous amount of simulation runs. This requirement has led to the development of the *weighted SSA* (wSSA) [55], which uses importance sampling techniques to determine the probability of rare events occurring.

In addition to using simulation to determine event probabilities, methods to compute the system's event probabilities directly have been developed. Indeed, model checking approaches have been applied to biological models to check whether or not the system behaves as expected [5, 11, 42]. The first of these methods involve the use of qualitative logical models of genetic circuits generated by hand [78, 80], but producing these models is time consuming and error prone and they are incapable of yielding quantitative predictions of behavior. To address this problem, methods for generating a quantitative logical model that encodes the infinite state space of a genetic circuit into a finite number of logical levels for each chemical species have been developed [56, 53]. The resulting quantitative logical models can be thought of as Markov chains. Over the years, there have been many methods developed for analyzing finite state Markov chains [19, 20, 48, 57]. However, most interesting models are of systems that have infinite state spaces leading to abstraction techniques that can handle infinite state Markov chains with discrete time [1, 2, 3, 71, 24, 25, 26, 52]. Additionally, since the real world works in continuous-time, most recent research has focused on continuous-time Markov chains [72].

In order to deal with infinite state space continuous-time Markov chains, techniques have been developed that effectively transform the Markov chain into a discrete time finite state model. Many methods explore the chain up to a specified depth as they try to approximate the system [39, 63, 16]. Others perform this exploration but are capable of

dynamically exploring more of the chain if the explored states do not satisfy a desired level of percision [10]. Finally, some methods attempt to explore the infinite chain in sections and use a sliding window to move to different sections of the chain as necessary [43].

Infinite state space and continuous-time are not the only problems that have been encountered when analyzing Markov chains. Similar to stiff reaction networks, some Markov chains are stiff because they contain both fast and slow states and methods have been developed to approximate the chain with a nonstiff equivalent [9]. Additionally, it can often be difficult to approximate rare event probabilities [62] or probabilities of events in noisy systems [21]. Some research also focuses on hybrid analysis methods where Markov chains are analyzed with both stochastic and deterministic techniques [44].

After a circuit has been abstracted into the logical representation of a computationally tractable Markov chain, it can be efficiently analyzed using stochastic model checking [58] to quickly compare the robustness of alternative circuit designs and evaluate different design trade-offs. Stochastic model checking is a technique that utilizes either statistical (simulation) or numerical (Markov chain analysis) techniques to determine the probability that a system has a specified property [85].

Another approach is to abstract the network into a *Bayesian network* [69]. These networks are probabilistic graphical models that represent each variable in the system as a random variable with conditional dependencies on the other variables in the system. Bayesian network analysis is typically used in network learning applications [28]; however, there have been some approaches that attempt to create Bayesian networks from ODE analysis of different systems [59].

## 1.2 Contributions

This dissertation presents the idea of a new, adaptive *incremental stochastic simulation algorithm* (iSSA) that is a variant of the SSA and that can aid researchers in determining a system's typical behavior. Additionally, this dissertation presents the use of verification techniques to analyze genetic circuits to determine the likelihood that they exhibit a desired property. Due to their stochastic nature, however, genetic circuits do not lend themselves to traditional verification methods. As such, *stochastic model checking* techniques leveraging *Markov chain analysis* are utilized to obtain the desired results.

The main contributions of this research are the development of methods to more efficiently perform stochastic analysis and design space exploration of genetic circuits. These contributions include:

- The development of an iSSA.
- The translation of genetic circuits into CTMCs, and the application of stochastic model checking techniques to the resulting CTMC in order to determine the likelihood that a system exhibits an interesting behavior.
- The evaluation of these methods by their application to some interesting models including, among others, various genetic oscillators and state-holding gates.

An example work flow of using these methods in concert to analyze a genetic circuit is shown in Figure 1.1. Here, a genetic circuit is both simulated using the iSSA and abstracted using logical abstraction. Next, a property representing the typical behavior of the system is determined from the iSSA results and sent to the stochastic model checker along with the resulting CTMC from logical abstraction. Finally, the output of this work flow is the probability that the circuit exhibits its typical behavior or, perhaps even more interesting, the probability that it does not.

This work has been integrated into the tool `iBioSim` [65]. This tool is freely available to the public for download at `http://www.async.ece.utah.edu/iBioSim/`.

## 1.3 Dissertation Outline

This dissertation is organized as follows. Background information on genetic circuits is presented in Chapter 2. This chapter gives a brief overview of what makes up a genetic

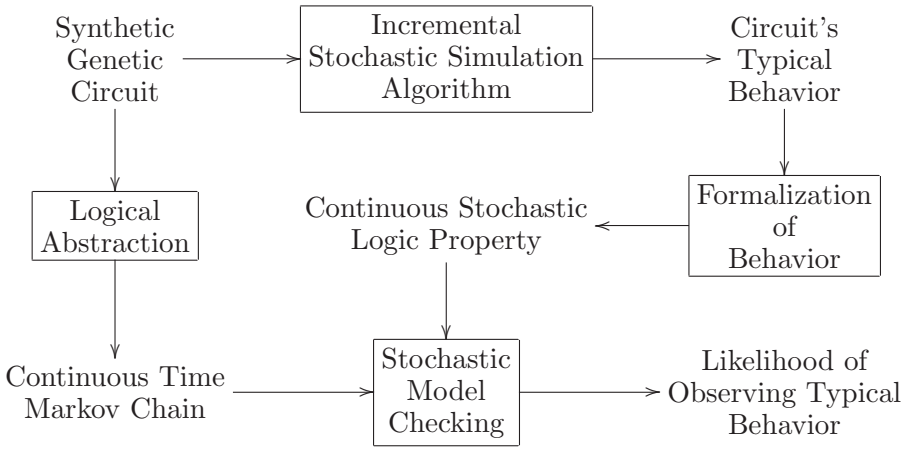



**Figure 1.1**: An example work flow of using iSSA, logical abstraction, and stochastic model checking to determine the likelihood of a system exhibiting its typical behavior.

circuit and describes how synthetic biology techniques are used to engineer a synthetic genetic circuit.

Chapter 3 describes how to model a genetic circuit as a chemical reaction network and discusses various simulation and analysis methods that have been applied to genetic circuits. This chapter concludes by discussing various model abstraction techniques that are useful for the iSSA and stochastic model checking methods presented in later chapters.

Chapter 4 presents a new simulation method known as the iSSA which is a variant of the SSA. This chapter gives the intuition behind this algorithm and describes several variations to the algorithm. These variants are then compared by using them to analyze an example model.

In Chapter 5, a methodology for abstracting genetic circuits to logical models is presented. Additionally, this chapter describes how this logical model can be augmented with transition rates resulting in a probabilistic model and explains how stochastic model checking can be used to analyze the model. Similar to the previous chapter, this methodology is then used to analyze an example model.

Case studies illustrating the utility of using both iSSA and stochastic model checking in concert to perform design space exploration are presented in Chapter 6. This chapter uses these methods to analyze several oscillator circuits and state holding gates and wraps up by presenting analysis of a synthetic genetic circuit for a quorum trigger.

Finally, Chapter 7 concludes this dissertation by giving a summary of the work and by presenting possible future research directions.

# CHAPTER 2

# GENETIC REGULATORY CIRCUITS

All living organisms are composed of one or more cells, and each of these cells carries out specialized functions in order to keep the organism alive. There are networks within the cell called *genetic regulatory circuits* which help regulate the amount of proteins that are synthesized from the cell's genes. This regulation can be used to signal when the cell should divide, when the cell should take in nutrients from the environment, when the cell should change to a defensive state, and when the cell should die, among other behaviors. Understanding these circuits can give synthetic biologists greater insight into why cells behave as they do and can in turn lead to better engineered synthetic genetic circuits. The goal of this chapter is to give a brief overview of how genetic regulatory circuits operate within a cell and how they can be synthetically created. Section 2.1 details the processes known as transcription and translation which is how proteins are ultimately produced from genes. Section 2.2 describes how gene regulation occurs through interactions of DNA, RNA, proteins, and other small molecules. Section 2.3 briefly presents how synthetic genetic circuits are synthesized and inserted into a cell in a wet-lab environment.

## 2.1 Transcription and Translation

Most biological functions are facilitated through the use of biochemical compounds known as *proteins*. Indeed, the overall state of the cell can often be discerned simply by measuring the concentrations of different proteins within a cell. For example, some proteins known as *enzymes* can affect how fast certain reactions occur in the cell and their presence may indicate that the cell is in an unstable state whereas other proteins can provide support for the cell and may indicate that it is in a stable state. Most of the time, proteins within a cell are produced by the *transcription* and *translation* of the *genes* located within the cell's *deoxyribonucleic acid* (DNA).

DNA is a nucleic acid located within the *nucleus* of a cell that contains the genetic instructions for how an organism should develop and behave. It is composed of two strands

known as the *sense* and the *antisense* strands that contain a sequence of the four bases *adenine* (A), *cytosine* (C), *guanine* (G), and *thymine* (T). These bases have an affinity to bind to each other and pair up such that A only binds with T and G only binds with C and vice versa. Due to this binding, the two strands connect to each other and the DNA forms a double helix structure.

Sequences of the DNA bases can code for various genetic components including genes. Genes encode the instructions for the production of *ribonucleic acids* (RNAs) which are single-stranded nucleic acids. Many of these RNAs contain the instructions for the construction of proteins and are known as *messenger RNA* (mRNA). The process of producing mRNA from the genes on the DNA is known as transcription and begins when an enzyme known as *RNA polymerase* (RNAP) binds to a region of the DNA near a gene known as a *promoter*. The RNAP then temporarily breaks the bonds holding the two strands of DNA together which causes the DNA to unwind or unzip. During this unwinding, the RNAP walks along the DNA sequence that codes for the gene building up the mRNA one base pair at a time as shown in Figure 2.1. As seen in this figure, the mRNA is made up of the same base pairs as DNA except that thymine is replaced by *uracil* (U), an unmethylated form of thymine. This process continues until the RNAP reaches a region of the DNA known as a *terminator*. At this point, the RNAP detaches from the DNA, the newly formed mRNA is released, and the DNA winds back up again to reform the double helix.

Once transcription is complete, the mRNA is transported out of the cell's nucleus and into its *cytoplasm* where it is free to interact with *ribosomes*, protein-RNA complexes located within the cell. This interaction is called translation and eventually leads to the production of a protein. Figure 2.2 depicts the process of translation. Ribosomes bind to groups of three bases on the mRNA known as *codons* which code for one of twenty-one *amino acids*. Once the ribosome binds, the appropriate amino acid is delivered to the ribosome with the help of *transport RNA* (tRNA). The amino acids are then chained together in a sequence which eventually produces a protein.

The combination of transcription and translation to first convert DNA into mRNA and then convert the mRNA into proteins in known as the *central dogma of molecular biology* [18]. It is this dogma that allows for the function of genetic circuits. However, these steps alone do not allow for proteins to interact and affect each other.

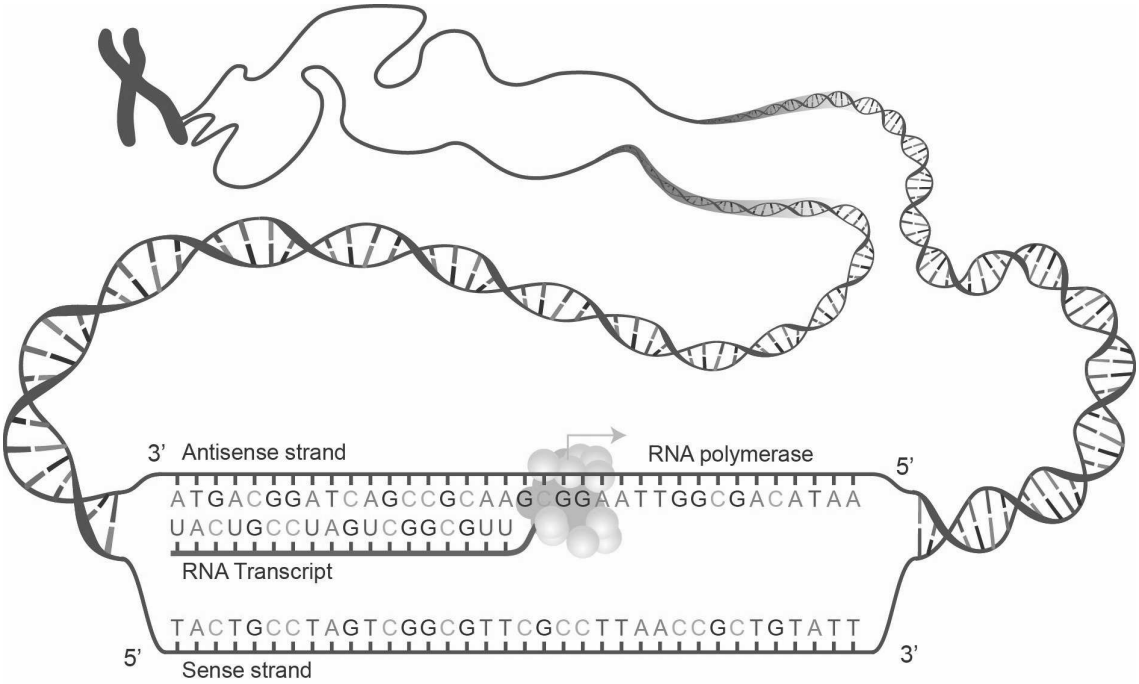



**Figure 2.1**: Diagram illustrating the process of transcription. RNAP walks along the unwound DNA to build up a strand of mRNA. This mRNA can later be translated into a protein through the process of translation. (Courtesy of the National Human Genome Research Institute)

## 2.2 Genetic Regulation

With no regulation, each gene in a genetic circuit is transcribed and translated to produce proteins at an unrestrained rate which ultimately leads to cells being unable to function. In fact, having too high of a concentration of some proteins can even lead to cellular death. For this reason and many others, genetic circuits contain mechanisms for regulating the amount of each protein that is produced in a cell.

There are many steps in the transcription and translation cycle where regulation can occur. For example, there are tiny RNAs known as *micro RNAs* (miRNAs) that can bind with mRNA to prevent the translation of the mRNA into proteins. These same miRNAs can also bind directly with DNA to prevent RNAP from transcribing a particular gene. However, in general, genetic regulation refers to regulation of transcription through the use of *transcription factors*. A transcription factor is typically a regulatory protein that binds to an *operator site*, an area of DNA near a promoter. By doing so, the transcription factor ends up either blocking RNAP from initiating transcription (in this

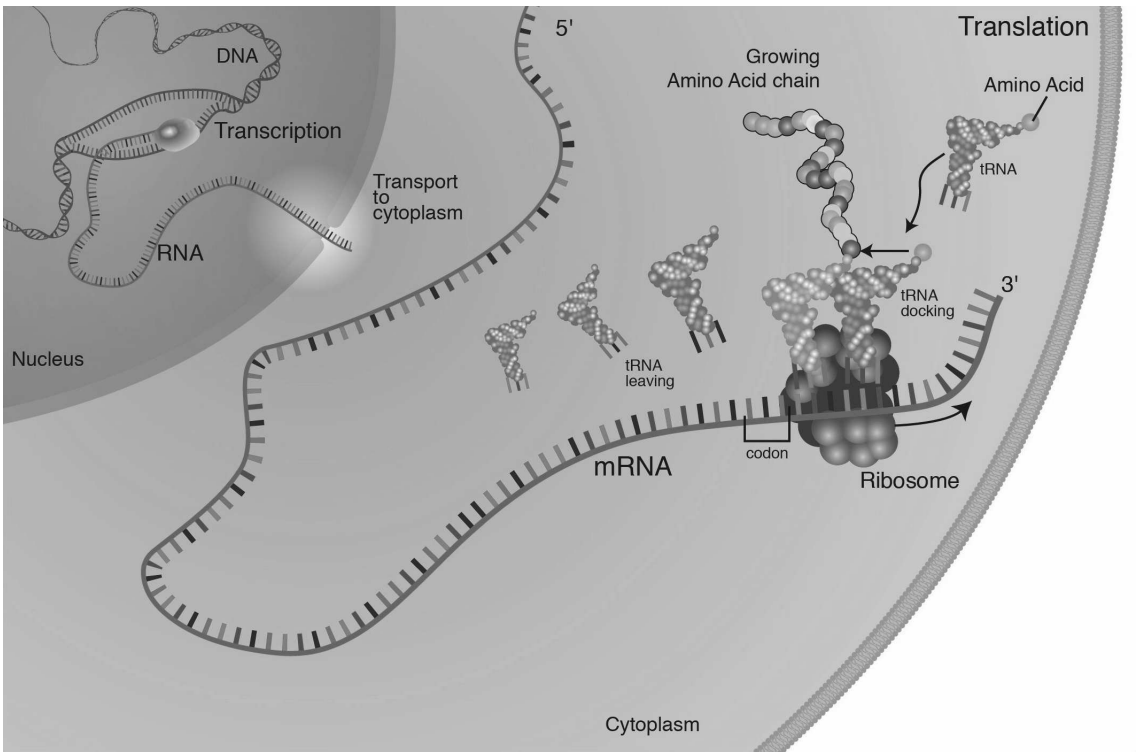



**Figure 2.2**: Diagram illustrating the process of translation. Following transcription, the newly formed mRNA is transported outside of the nucleus into the cytoplasm. Here, ribosomes bind to codons on the mRNA and signal tRNA to bring the appropriate amino acid. These amino acids are then chained together to create a protein. (Courtesy of the National Human Genome Research Institute)

case the transcription factor is known as a *repressor*) or attracting RNAP to the promoter to initiate transcription (in this case the transcription factor is known as an *activator*). For convenience, the functions Rep($p$) and Act($p$) return the transcription factors that repress and activate promoter $p$, respectively. Additionally, the set of proteins that are produced from the transcription and translation of genes that are downstream from a promoter $p$ can be determined by using the function Prod($p$).

In addition to regulation through transcription factors, there are small molecules located within a cell that can also influence gene activity. These molecules, known as *chemical inducers*, bind to transcription factors to form complexes in order to prevent them from either activating or repressing the promoters that they are typically associated with. This interaction could be viewed as repression or activation depending on the type of transcription factor it is influencing. However, there are also instances where the

presence of chemical inducers is necessary because the complexes that are formed act as transcription factors for other promoters. In this situation, one could think of the chemical inducer as assisting in either activation or repression.

An example of a genetic regulatory circuit representing the genetic toggle switch described in [29] is shown in Figure 2.3. This circuit has two stable states: either LacI is high and TetR is low (the OFF state) or LacI is low and TetR is high (the ON state). To switch from one stable state to the other, the values of IPTG and aTc can be altered. In 2000, Gardner et al. successfully designed and constructed this circuit and inserted it into *Escherichia coli* bacteria where they were able to observe this switch-like behavior. For simplicity's sake, the steps of transcription and translation have been collapsed into a single step where a gene simply produces proteins instead of producing mRNA that would then have to be translated to produce the proteins in Figure 2.3. In this figure, Prod($P_{trc-2}$) returns TetR and *green fluorescent protein* (GFP), a reporter that causes the cell to glow indicating whether the toggle is in the ON or OFF state, while Prod($P_{LtetO-1}$) returns LacI. The LacI protein is returned by Rep($P_{trc-2}$) as it binds to the operator site associated with the $P_{trc-2}$ promoter to repress the production of TetR and GFP. Similarly, the TetR protein is returned by Rep($P_{LtetO-1}$) as it binds to the operator site associated with the $P_{LtetO-1}$ promoter to repress the production of LacI. The other molecules in the diagram, IPTG and aTc, are chemical inducers. IPTG represses LacI's ability to act as a repressor by binding with it to form a complex, C1, preventing LacI from being able to repress TetR production. Similarly, aTc represses TetR's ability to act as a repressor by binding with it to form a complex, C2, preventing TetR from being able to repress LacI production.

## 2.3 Synthesis of Genetic Circuits

In order to build a synthetic genetic circuit and insert it into a cell, researchers follow some basic steps. First, the researchers design a useful circuit. For example, the circuit in Figure 2.3 can be used as a switch where a researcher can set it into the OFF state by supplying it with aTc and then can observe whether or not a medium contains IPTG by observing whether or not the circuit switches to the ON state when placed in the medium. The next step of the process is to query a database of genetic parts such as the `Registry of Standard Biological Parts` [12] to determine if it contains suitable parts that can be stitched together to create the designed circuit. If all the parts are in

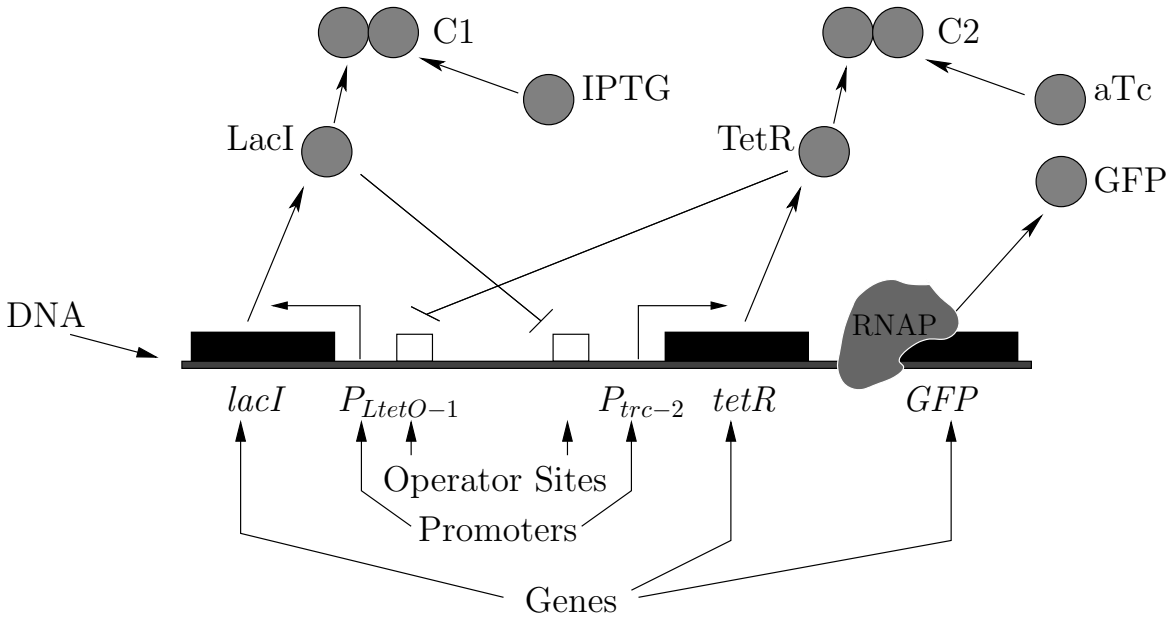



**Figure 2.3**: The genetic toggle switch circuit where LacI and TetR repress each other (denoted by the ⊣ arrows). In this circuit, LacI can be sequestered by IPTG, TetR can be sequestered by aTc, and GFP is the reporter protein causing the cell to glow indicating whether the toggle is in the ON or OFF state.

the database, the researcher can obtain the parts and join them together. This process is known as assembly and is shown in Figure 2.4. In this diagram, parts B0034 and C0010 from the `Registry of Standard Biological Parts` are joined together. This process is accomplished by using a *restriction enzyme* to cleave the part B0034 free from its *plasmid* (i.e., a continuous loop of DNA). Another restriction enzyme is then used to cleave a hole in the plasmid containing part C0010. Finally, these parts are mixed together where through *DNA recombination* they *ligate* (i.e., bind together) to form a new plasmid. Figure 2.5 shows how the restriction enzyme cleaves the DNA in order to form "sticky ends" that allow this ligation to take place. At this point, the genetic circuit is integrated into the host cell and begins functioning using the host cell's genetic machinery, and the researcher can begin performing experiments with the circuit.

Sometimes not all of the necessary parts are available in a database of genetic parts. In this case, the researcher may have to determine the DNA sequence for the desired genetic circuit. Once the this sequence is determined, the researcher can send the sequence off to a DNA-sequencing service where the circuit can be synthesized one base pair at a time. At this point, the completed circuit can be inserted into a cell and experimented upon.

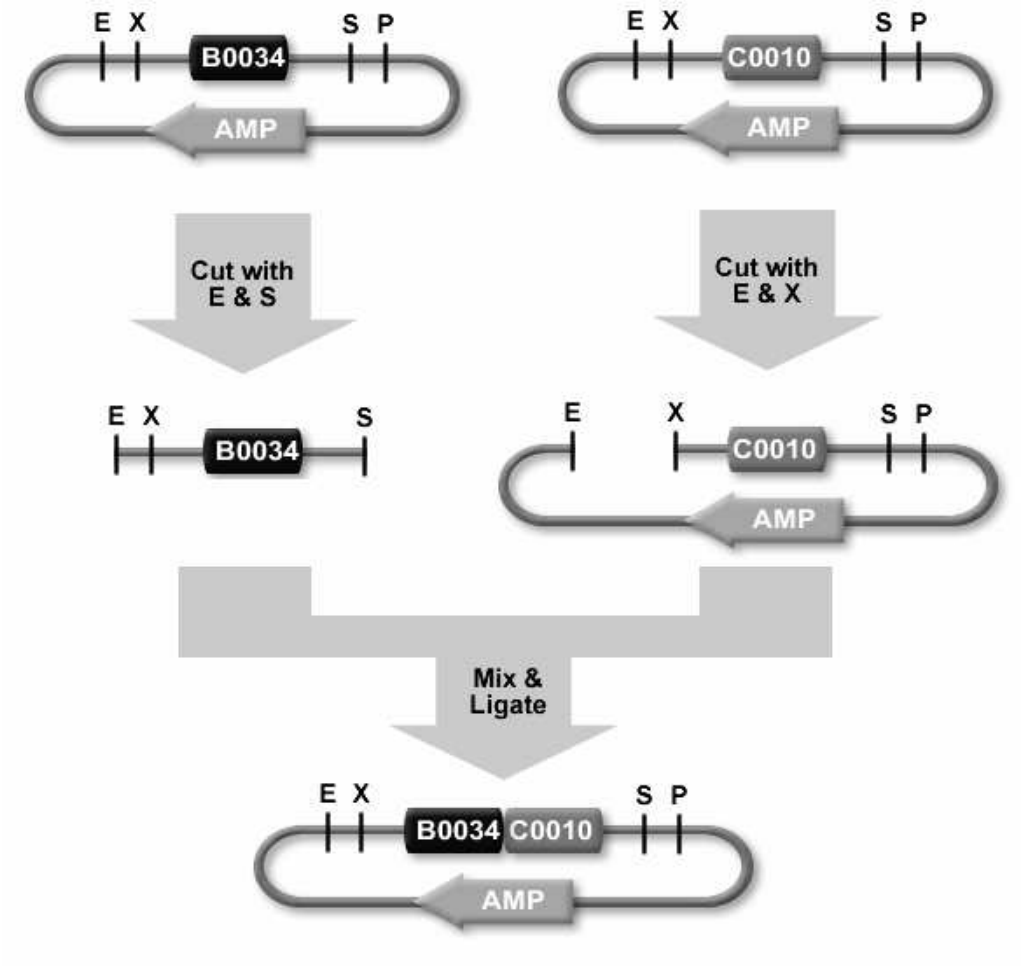



**Figure 2.4**: Diagram showing the process of assembling two biological parts together. In this diagram, part B0034 is cleaved out of its plasmid by using a restriction enzyme to cut the plasmid at points E and S. Additionally, the plasmid holding part C0010 is cleaved with a restriction enzyme at points E and X. The remaining parts are then allowed to mix and ligate to form a new part where part B0034 has been joined to part C0010. (Courtesy of the `Registry of Standard Biological Parts`)

The insertion process typically involves a researcher selecting a host cell that has a simple enough genome that the inserted circuit does not take up too much of the cells resources causing the cell to die. Also, it is important to select a host cell that does not contain the same genes or promoters as the designed circuit so that the circuit does not interfere with other processes in the cell and vice versa. A cell that is often chosen as the host is *Escherichia coli* or *E. coli* which is a mostly harmless, rod-shaped bacterium commonly found in the lower intestine of warm-blooded organisms.

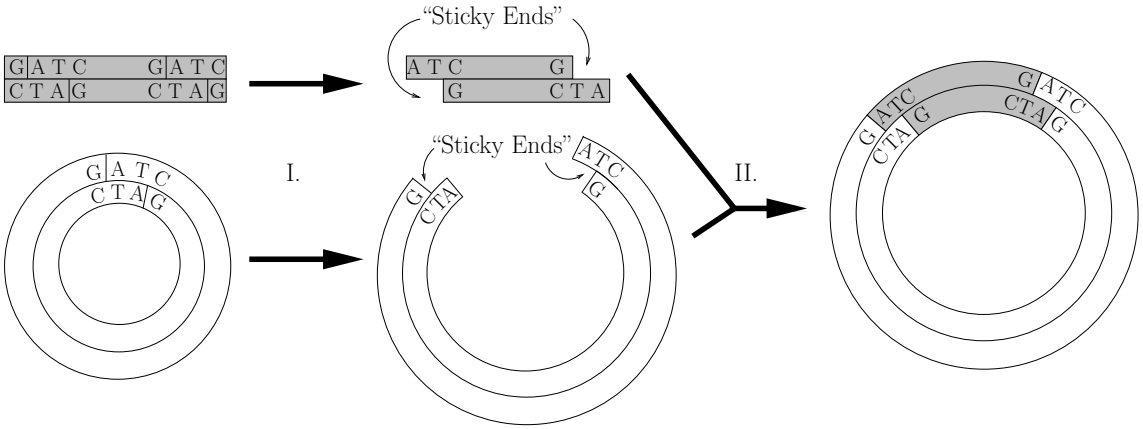



**Figure 2.5**: Diagram depicting the insertion of a segment of DNA into a plasmid. This insertion is accomplished by: I. Restriction enzymes cleaving the DNA of the plasmid and of the segment of DNA creating "sticky ends." II. DNA recombination where the "sticky ends" bind to each other causing the segment of DNA to be incorporated into the plasmid.

# CHAPTER 3

# MATHEMATICAL MODELING AND ANALYSIS

In order to reason about genetic regulatory circuits, researchers need to be able to model and analyze them. Traditionally, researchers have used mathematical models to describe systems of interest as these models can be analyzed and simulated using well known techniques to give a researcher a better idea of how the system behaves. However, in order to use mathematical models, the system must first be broken down into basic chemical reactions. Section 3.1 describes chemical reactions, the fundamental processes that allow for a genetic circuit to change state. Section 3.2 describes how a genetic circuit can be represented as a collection of chemical reactions. Section 3.3 introduces a method for analyzing a chemical reaction network using traditional *classical chemical kinetics* (CCK) that assume that the system behaves continuously and deterministically. Section 3.4 presents an alternative method for analyzing a chemical reaction network utilizing *stochastic chemical kinetics* (SCK) that assume that the system behaves discretely and stochastically. Section 3.5 exhibits some model abstractions that can reduce the size and complexity and improve the analysis efficiency of a genetic circuit.

## 3.1 Chemical Reaction Networks

The fundamental process of converting a subset of species in a system to another subset of species is known as a *chemical reaction*. The species that are consumed by the reaction are known as the *reactants* and the species that are produced by the reaction are known as the *products*. Additionally, there can be species that are neither produced nor consumed by a reaction but must be present in order to catalyze the reaction. These species are known as *modifiers*. For example, the chemical reaction:

$$s_1 + s_3 \rightarrow s_2 + s_3 \tag{3.1}$$

has one reactant, species $s_1$, one product, species $s_2$, and one modifier, species $s_3$.

The reaction shown in Reaction 3.1 is known as a *bimolecular reaction* because it requires two species to collide in order to initiate the reaction. Although it is possible for a reaction to occur when more than two species collide, it is extremely unlikely. Because of this fact, bimolecular reactions and *unimolecular reactions*, reactions that contain only one reactant, are typically known as *elementary reactions* and cannot be broken down into smaller reactions. Sometimes, for simplicity's sake, researchers approximate a system of elementary reactions with nonelementary reactions. Although this approximation can lead to erroneous results, a good approximation can enable faster analysis with the addition of little to no error.

There are many different types of bimolecular and unimolecular reactions. For example, the chemical reaction:

$$s_1 + s_2 \rightleftharpoons c_1 \tag{3.2}$$

is known as a *complex-formation* reaction. These reactions occur when two molecules of different species combine together to form a molecule of a third species. The double arrow symbol in the complex-formation reaction in Reaction 3.2 means that this reaction is *reversible* and is used as a shorthand for two reactions:

$$s_1 + s_2 \rightarrow c_1 \tag{3.3}$$

$$c_1 \rightarrow s_1 + s_2 \tag{3.4}$$

In any chemical reaction network, almost all of the reactions can be thought of as being reversible. Indeed, even the irreversible reactions can occur in the reverse direction. The reason that they are typically not written as reversible reactions is that the rate at which the forward reaction occurs is far greater than that of the reverse reaction causing the reverse direction to be negligible.

Another common bimolecular reaction is a specialized version of a complex-formation reaction known as a *dimerization* reaction. An example dimerization reaction is given in Reaction 3.5:

$$2s_2 \rightleftharpoons s_4 \tag{3.5}$$

The 2 in front of species $s_2$ is the *stoichiometry* and indicates that this reaction takes two molecules of species $s_2$ as a reactant. When a number is not present, as in Reactions 3.1 and 3.2, then the stoichiometry is assumed to be one.

The most common type of unimolecular reaction is the *degradation* reaction. An example degradation reaction is shown in Reaction 3.6:

$$s_1 \rightarrow \emptyset \tag{3.6}$$

This reaction takes one molecule of $s_1$ and destroys it by breaking it down. Although this reaction is written as having no products, its products are components that can be used to create new molecules due to the conservation of mass; however, they are omitted to abstract away unnecessary details. This type of reaction is very important as basically all molecules eventually degrade. Degradation reactions are also unique in that they are one of the few reactions that is typically not reversible because once the molecule is broken down, it cannot simply be put back together but must be produced through other means.

## 3.2 Genetic Circuits as Chemical Reaction Networks

Genetic regulatory circuits as they are presented in Chapter 2 can be analyzed by converting them into chemical reaction networks. This conversion can be accomplished by representing important molecules and constructs in the circuit as chemical species and by representing their interactions as chemical reactions.

The conversion process begins by creating a species for each of the proteins, promoters, and chemical inducers in the circuit. Additionally, a species is created for RNAP and for each protein's mRNA if the steps of transcription and translation are going to be represented by separate reactions in the model. The next step is to create a degradation reaction for each of the protein species. For instance, the degradation reactions for protein species in the genetic toggle switch in Figure 2.3 are shown in Reactions 3.7 to 3.11:

$$\text{LacI} \rightarrow \emptyset \tag{3.7}$$

$$\text{TetR} \rightarrow \emptyset \tag{3.8}$$

$$\text{GPF} \rightarrow \emptyset \tag{3.9}$$

$$\text{C1} \rightarrow \text{IPTG} \tag{3.10}$$

$$\text{C2} \rightarrow \text{aTc} \tag{3.11}$$

Note that the degradation reaction for C1 produces IPTG and the degradation reaction for C2 produces aTc. The reaction are written this way because IPTG and aTc are chemical inducers and should not degrade away, even when they are part of a complex.

The next step is to create the *open complex formation* reactions. These are reactions that represent the binding of RNAP to a promoter to perform transcription and translation in order to produce the proteins determined using Prod($p$). When creating open complex formation reactions, researchers have the option to either explicitly represent transcription and translation as two separate steps or to collapse them together into one step. For example, the open complex formation reactions for the $P_{LtetO-1}$ promoter can be represented with explicit steps for transcription and translation as in Reactions 3.12 to 3.15:

$$P_{LtetO-1} + \text{RNAP} \rightleftharpoons \text{S1} \tag{3.12}$$

$$\text{S1} \rightarrow P_{LtetO-1} + \text{RNAP} + \text{mRNA}_{\text{LacI}} \tag{3.13}$$

$$\text{mRNA}_{\text{LacI}} \rightarrow \text{mRNA}_{\text{LacI}} + \text{LacI} \tag{3.14}$$

$$\text{mRNA}_{\text{LacI}} \rightarrow \emptyset \tag{3.15}$$

Here, RNAP binds with $P_{LtetO-1}$ to form the complex S1 which can either fall apart into RNAP and $P_{LtetO-1}$ or can be involved in a transcription reaction to produce the mRNA for LacI, determined by using Prod($P_{LtetO-1}$). At this point, the newly formed mRNA can act as a modifier in a reaction that represents its translation to produce the LacI protein, or it can degrade away.

If the steps of transcription and translation are collapsed together, then fewer reactions are necessary when creating open complex formation reactions. In Reactions 3.16 to 3.17, RNAP still binds with $P_{LtetO-1}$ to form the complex S1, but instead of producing mRNA, this complex is a modifier in a reaction that directly produces the LacI protein:

$$P_{LtetO-1} + \text{RNAP} \rightleftharpoons \text{S1} \tag{3.16}$$

$$\text{S1} \rightarrow \text{S1} + \text{LacI} \tag{3.17}$$

For simplicity's sake, the genetic regulatory circuits presented in this dissertation use chemical reaction networks that collapse the transcription and translation steps together. The rest of the open complex formation reactions for the genetic toggle switch are shown in Reactions 3.18 to 3.19:

$$P_{trc-2} + \text{RNAP} \rightleftharpoons \text{S2} \tag{3.18}$$

$$\text{S2} \rightarrow \text{S2} + \text{TetR} + \text{GFP} \tag{3.19}$$

Next, dimerization and complex formation reactions are created for the proteins and chemical inducers that form dimers and complexes. In the genetic toggle switch, none of

the species form dimers, so no dimerization reactions are created; however, LacI forms a complex with IPTG, and TetR forms a complex with aTc. These complex formation reactions are shown in Reactions 3.20 to 3.21:

$$\text{LacI} + \text{IPTG} \rightleftharpoons \text{C1} \tag{3.20}$$

$$\text{TetR} + \text{aTc} \rightleftharpoons \text{C2} \tag{3.21}$$

The final steps are to create reactions representing the activation and repression of the promoters by the species that influence them determined using Act($p$) and Rep($p$) respectively. Repression reactions are relatively simple to generate as they are a form of complex formation reactions where the repressor binds to the promoter to form a complex, thus, preventing RNAP from binding to it to produce proteins. In the genetic toggle switch, there are two repression reactions shown in Reactions 3.22 to 3.23:

$$\text{LacI} + P_{trc-2} \rightleftharpoons \text{S3} \tag{3.22}$$

$$\text{TetR} + P_{LtetO-1} \rightleftharpoons \text{S4} \tag{3.23}$$

Activation reactions, on the other hand, are similar to open complex formation reaction except that they include the activator as one of the reactants in the complex formation. The genetic toggle switch does not contain any activators, so its chemical reaction network does not contain any activation reactions. However, the simple one gene circuit in Figure 3.1 contains a promoter that is activated by a species. Here, Prod($P_1$) returns B and Act($P_1$) returns A yielding the open complex formation and activation reactions shown in Reactions 3.24 to 3.27:

$$P_1 + \text{RNAP} \rightleftharpoons \text{S2} \tag{3.24}$$

$$\text{S2} \rightarrow \text{S2} + \text{B} \tag{3.25}$$

$$P_1 + \text{A} + \text{RNAP} \rightleftharpoons \text{S1} \tag{3.26}$$

$$\text{S1} \rightarrow \text{S1} + \text{B} \tag{3.27}$$

It should be noted that although the open complex formation and activation reactions look similar, the likelihood of S1 yielding B is higher than the likelihood of S2 yielding B. This increase in likelihood is due to the reactions occurring at different rates as discussed in Section 3.3, and the activation reaction's activated rate is higher than the open complex formation's *basal* rate.

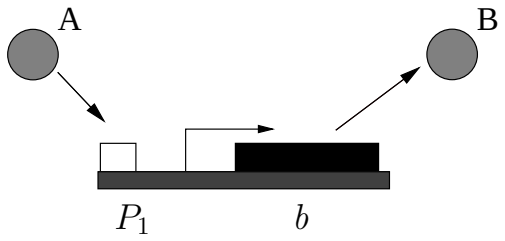



**Figure 3.1**: A simple one gene circuit where species A activates the production of species B on promoter $P_1$.

## 3.3 Classical Chemical Kinetics

Once the genetic circuit is converted into a chemical reaction network of $n$ species and $m$ reactions, it can be analyzed using CCK. CCK are well-known methods which assume that the network reacts continuously in a well-stirred medium and that there are a large amount of molecules in the system. In CCK, the rate of each reaction is considered to be the speed at which it alters the amount of the species that participate in it per unit time. To compute these rates, CCK converts the network into a set of ODEs using the law of mass action. By utilizing this law, it is able to leverage many well-established theories that deal with systems of ODEs.

The law of mass action states that the rate of each chemical reaction is proportional to the product of the *concentrations* (i.e., the amount of molecules divided by the volume) of the reactant species in the reaction. This statement means that the rate of a reaction can be computed by multiplying the concentration of each of its reactant species by the reaction's *rate constant*, a coefficient that can change the rate of a reaction based on factors such as temperature. As an example, consider Reaction 3.1. To show that this reaction should be considered using mass action kinetics, this reaction can be rewritten to include the reaction's rate constant, $k_1$:

$$s_1 + s_3 \xrightarrow{k_1} s_2 + s_3 \tag{3.28}$$

The reaction rate, $V$, of this reaction can then be written as:

$$V = k_1|s_1||s_3| \tag{3.29}$$

where $|s_i|$ is the concentration of species $s_i$.

If a reaction is reversible, then it has a rate constant for the forward reaction and a rate constant for the reverse reaction. For instance, Reaction 3.5 can be annotated with the forward rate constant $k_2$ and the reverse rate constant $k_{-2}$:

$$2s_2 \underset{k_{-2}}{\overset{k_2}{\rightleftharpoons}} s_4 \tag{3.30}$$

The reaction rate for Reaction 3.30 is slightly more complicated than an irreversible reaction. Basically, it is derived by computing the rate of each direction of the reaction. Then, the rate of the reverse direction is subtracted from the rate of the forward direction as shown in Equation 3.31:

$$V = k_2|s_2|^2 - k_{-2}|s_4| \tag{3.31}$$

In this equation, the concentration of $s_1$ is squared because two molecules of $s_1$ combine to make a molecule of $s_2$. In general, a reactant species involved in a reaction is raised to the power of its stoichiometry when computing the reaction rate.

Using the law of mass action, the general equation for computing the reaction rate $V_j$ of reaction $r_j$ is given by Equation 3.32:

$$V_j = k_+^j \prod_{i=1}^{n} |s_i|^{v_{ij}^r} - k_-^j \prod_{i=1}^{n} |s_i|^{v_{ij}^p} \tag{3.32}$$

where $k_+^j$ is the forward rate constant of the $j^{\text{th}}$ reaction, $k_-^j$ is the reverse rate constant of the $j^{\text{th}}$ reaction, $v_{ij}^r$ represents the number of species $s_i$ molecules consumed by the $j^{\text{th}}$ reaction, and $v_{ij}^p$ represents the number of species $s_i$ molecules produced by the $j^{\text{th}}$ reaction.

Figure 3.2 shows the reaction graph of the chemical reaction network of the genetic toggle switch after adding rate constants from Table 3.1 and applying the law of mass action to Reactions 3.7 to 3.23.

### 3.3.1 Deriving an ODE Model

Using mass action kinetics, one can derive a reaction rate for each reaction in a system and can use these rates to write a system of ODEs that define the behavior of each species. For example, in a system where Reactions 3.28 and 3.30 are the only reactions, four ODEs can be constructed that describe the evolution of species $s_1$, $s_2$, $s_3$, and $s_4$. Since species

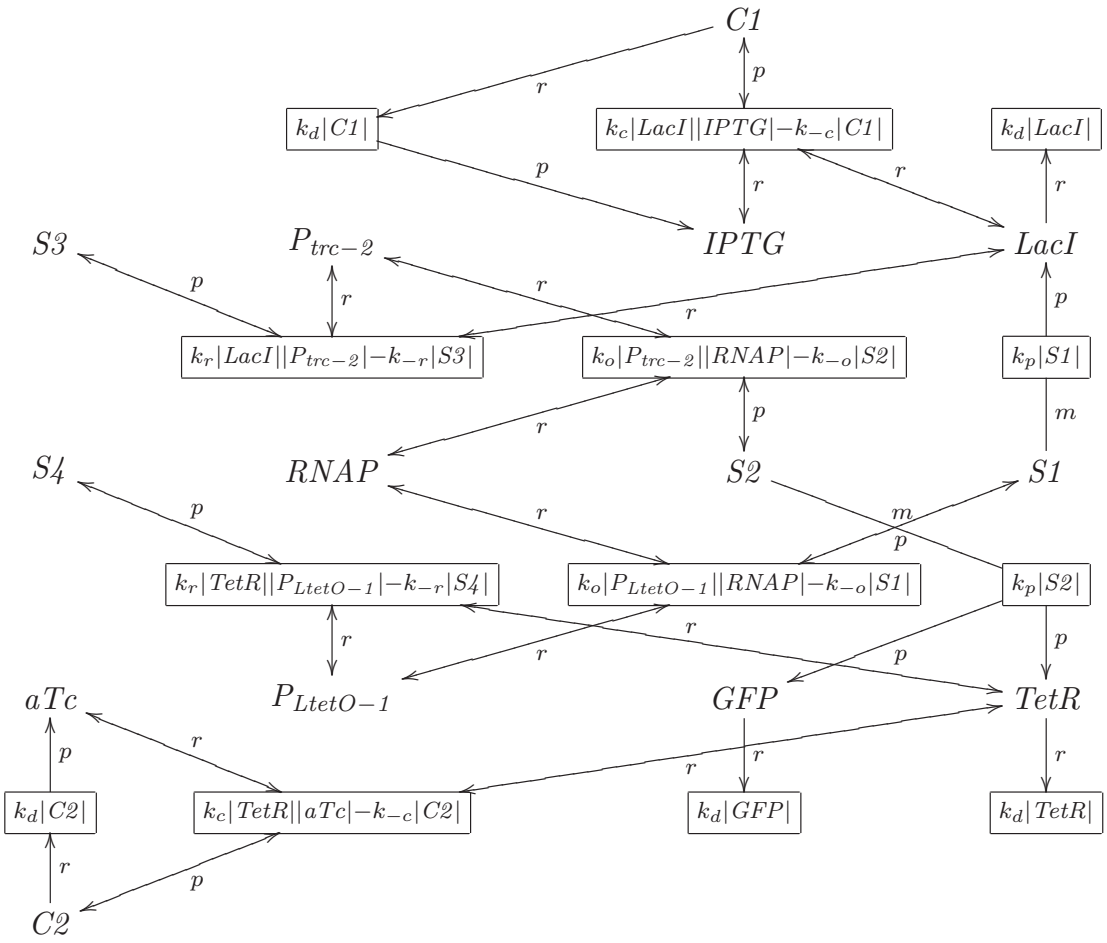



**Figure 3.2**: Reaction graph for the genetic toggle switch. Reaction rate equations are shown in the boxes representing each reaction. The edges between species and reactions are labeled $r$ if the species is a reactant in the reaction, $p$ if the species is a product of the reaction, and $m$ if the species is a modifier in the reaction.

$s_1$ is a reactant and is consumed by Reaction 3.28, the ODE that describes its behavior is the negation of this reaction rate:

$$\frac{d|s_1|}{dt} = -k_1|s_1||s_3| \tag{3.33}$$

On the other hand, species $s_2$ is a product of one reaction and is a reactant in the other which means that its ODE is the sum of the rate for Reaction 3.28 and the negated rate for Reaction 3.30:

$$\frac{d|s_2|}{dt} = k_1|s_1||s_3| - 2(k_2|s_2|^2 - k_{-2}|s_4|) \tag{3.34}$$

**Table 3.1**: Parameter list

| Parameter | Symbol | Value | Units |
|---|---|---|---|
| Degradation rate | $k_d$ | 0.0075 | $\frac{1}{sec}$ |
| Complex formation equilibrium | $K_c$ | 0.05 | $\frac{1}{molecule}$ |
| Stoichiometry of binding | $n_c$ | 2 | $molecule$ |
| Repression binding equilibrium | $K_r$ | 0.5 | $\frac{1}{molecule}$ |
| Activation binding equilibrium | $K_a$ | 0.0033 | $\frac{1}{molecule}$ |
| RNAP binding equilibrium | $K_o$ | 0.033 | $\frac{1}{molecule}$ |
| Activated RNAP binding equilibrium | $K_{oa}$ | 1 | $\frac{1}{molecule}$ |
| Basal production rate | $k_b$ | 0.0001 | $\frac{1}{sec}$ |
| Open complex production rate | $k_p$ | 0.05 | $\frac{1}{sec}$ |
| Activated production rate | $k_a$ | 0.25 | $\frac{1}{sec}$ |
| Stoichiometry of production | $n_p$ | 10 | $unit-less$ |

The rate for Reaction 3.30 is multiplied by two in this equation because two molecules of $s_2$ are consumed by this reaction. Species $s_3$ only participates in Reaction 3.28 as both a reactant and a product, so the ODE describing its evolution over time becomes:

$$\frac{d|s_3|}{dt} = k_1|s_1||s_3| - k_1|s_1||s_3| = 0 \tag{3.35}$$

The equation evaluating to zero means that the concentration of $s_3$ does not change as time advances. Finally, $s_4$ is a product of Reaction 3.30, so its ODE is:

$$\frac{d|s_4|}{dt} = k_2|s_2|^2 - k_{-2}|s_4| \tag{3.36}$$

Deriving an ODE model from a chemical reaction network using mass action kinetics can be generalized in the following way. For each species, the ODE describing the behavior of the species is equal to the sum of all the reaction rate stoichiometry products of reactions where the species participates as a product minus the sum of all the reaction rate stoichiometry products of reactions where the species participates as a reactant. Formally, this derivation is stated in Equation 3.37:

$$\frac{d|s_i|}{dt} = \sum_{j=1}^{m}(v_{ij}^p - v_{ij}^r)V_j \tag{3.37}$$

Using Equation 3.37, the full ODE model for the genetic toggle switch can be derived from the reaction graph shown in Figure 3.2. This ODE model is shown in Figure 3.3.

Once the full ODE model has been derived, the model can be analyzed using some well known techniques [70]. These techniques typically involve linearizing the system

$$
\begin{aligned}
\frac{d|\text{LacI}|}{dt} &= k_p|\text{S1}| - k_c|\text{LacI}||\text{IPTG}| + k_{-c}|\text{C1}| - k_r|\text{LacI}||P_{trc-2}| + k_{-r}|\text{S3}| - k_d|\text{LacI}| \\
\frac{d|\text{TetR}|}{dt} &= k_p|\text{S2}| - k_c|\text{TetR}||\text{aTc}| + k_{-c}|\text{C2}| - k_r|\text{TetR}||P_{LtetO-1}| + k_{-r}|\text{S4}| - k_d|\text{TetR}| \\
\frac{d|\text{GFP}|}{dt} &= k_p|\text{S2}| - k_d|\text{GFP}| \\
\frac{d|\text{C1}|}{dt} &= k_c|\text{LacI}||\text{IPTG}| - k_{-c}|\text{C1}| - k_d|\text{C1}| \\
\frac{d|\text{C2}|}{dt} &= k_c|\text{TetR}||\text{aTc}| - k_{-c}|\text{C2}| - k_d|\text{C2}| \\
\frac{d|\text{IPTG}|}{dt} &= -k_c|\text{LacI}||\text{IPTG}| + k_{-c}|\text{C1}| + k_d|\text{C1}| \\
\frac{d|\text{aTc}|}{dt} &= -k_c|\text{TetR}||\text{aTc}| + k_{-c}|\text{C2}| + k_d|\text{C2}| \\
\frac{d|P_{LtetO-1}|}{dt} &= -k_o|P_{LtetO-1}||\text{RNAP}| + k_{-o}|\text{S1}| - k_r|\text{TetR}||P_{LtetO-1}| + k_{-r}|\text{S4}| \\
\frac{d|P_{trc-2}|}{dt} &= -k_o|P_{trc-2}||\text{RNAP}| + k_{-o}|\text{S2}| - k_r|\text{LacI}||P_{trc-2}| + k_{-r}|\text{S3}| \\
\frac{d|\text{S1}|}{dt} &= k_o|P_{LtetO-1}||\text{RNAP}| - k_{-o}|\text{S1}| \\
\frac{d|\text{S2}|}{dt} &= k_o|P_{trc-2}||\text{RNAP}| - k_{-o}|\text{S2}| \\
\frac{d|\text{S3}|}{dt} &= k_r|\text{LacI}||P_{trc-2}| - k_{-r}|\text{S3}| \\
\frac{d|\text{S4}|}{dt} &= k_r|\text{TetR}||P_{LtetO-1}| - k_{-r}|\text{S4}| \\
\frac{d|\text{RNAP}|}{dt} &= -k_o|P_{LtetO-1}||\text{RNAP}| + k_{-o}|\text{S1}| - k_o|P_{trc-2}||\text{RNAP}| + k_{-o}|\text{S2}|
\end{aligned}
$$

**Figure 3.3**: The full ODE model of the genetic toggle switch derived from the chemical reaction network shown in Figure 3.2.

by evaluating the rate of change of each of the species' concentrations using the initial concentration of each species. The system is then evolved by selecting a small time step and computing a new concentration for each species at the end of the time step. This process is then repeated by computing new rates with the new concentration values and a new time step. Since it is often the case that the ODE rate equations are dependent on the concentration of many species, the results of ODE analysis may be erroneous if the time step is too large. However, if the time step is too small, then the analysis takes an extremely long time as many unnecessary steps are taken to evolve the system. To address this problem, there have been improvements that try to adapt the time step selection so

that it is able to make progress in evolving the system but is never so large that the ODE rates for each species change too dramatically.

### 3.3.2 Disadvantages of CCK

As stated previously, CCK assumes that the system behaves continuously and deterministically. When the amounts of each species are very large, this assumption is reasonable as changes in the population size of each species are relatively small and can be viewed as continuous changes. In addition, small fluctuations of species counts in a system where the population sizes are large can be safely disregarded as they do not affect the system significantly. Therefore, the dynamics of the system can also be viewed as a deterministic process.

In genetic regulatory circuits, the species counts are typically very small and each reaction can cause large fluctuations in species counts. Because these circuits violate the assumptions of CCK, the true behavior of the system is not able to be captured when analyzing it using CCK. Thus, researchers have turned to other methods to aid in the analysis of genetic regulatory circuits.

## 3.4 Stochastic Chemical Kinetics

In contrast to CCK, SCK assumes that a chemical reaction network behaves as a discrete-stochastic process. It achieves this assumption by treating the network as a well-stirred system with a discrete number of molecules. SCK can then simulate the time evolution of the system by stochastically firing reactions to change the amounts of each species in the system.

SCK keeps track of species' amounts instead of species' concentrations. Indeed, the system state in SCK is simply the population of each species in the system denoted $\vec{x} = (x_1, ..., x_N)$. Changes in the system state occur through the firing of reactions. When a reaction $r_j$ fires, it applies a *state change vector*, $\vec{v}_j = (v_{1j}, ..., v_{Nj})$, to the system state where each $v_{ij}$ is the amount of change reaction $r_j$ causes to species $s_i$'s population. The new system state is then defined by $\vec{x} + \vec{v}_j$.

### 3.4.1 Propensities

In order to simulate a system using SCK, the next reaction event and the time of next reaction event must be determined. This selection is governed by a *propensity function* $a_j(\vec{x})$ for each reaction $r_j$ in the system. Each $a_j(\vec{x})$ is the probability that, given $\vec{x}$,

reaction $r_j$ occurs within the next infinitesimal time interval that is so small that at most one reaction event can occur.

Each propensity function is quantified by multiplying the amount of each reactant to the reaction by the probability that a randomly chosen combination of molecules in the system are the reactants. It turns out that these probabilities are proportional to the rate constants used in CCK. For example, the propensity function for Reaction 3.28 is defined as $a_1(\vec{x}) = k_1 s_1 s_3$. As can be seen, the propensity is similar to the CCK reaction rate presented in Equation 3.31. This similarity is due to the fact that SCK actually approximates CCK when the molecule counts of species are very large.

### 3.4.2 Chemical Master Equation

When simulating a genetic circuit using SCK, all that matters are the current molecule counts as they are the only values that can change and affect the propensity functions. Therefore, each transition of the system only depends on the current state of the system and does not depend on the history of the system. This fact means that the circuit can be treated as a *temporally homogeneous jump Markov process* [34].

The Markov process described by SCK can be defined as the probability, $P(\vec{x}, t + dt|\vec{x}_0, t_0)$, that the system state is $\vec{x}$ at time $t + dt$ given that it was $\vec{x}_0$ as time $t_0$. In order to compute this probability, all the possible states that are only one step away from the system being in state $\vec{x}$ are considered. The probability of moving from each of these states to state $\vec{x}$ (i.e., each reaction propensity) is multiplied by the probability that the system reached state $\vec{x} - \vec{v}_j$ at time $t$ and these values are summed together. Additionally, the probability that the system is already in state $\vec{x}$ and no reactions occur is added to this probability to give the final Markov process probability. Formally, this probability is defined as:

$$\begin{aligned} P(\vec{x}, t + dt|\vec{x}_0, t_0) =& P(\vec{x}, t|\vec{x}_0, t_0) \left[1 - \sum_{j=1}^{M} a_j(\vec{x})dt\right] \\ &+ \sum_{j=1}^{M} [P(\vec{x} - \vec{v}_j, t|\vec{x}_0, t_0) a_j(\vec{x} - \vec{v}_j)dt] \end{aligned} \tag{3.38}$$

Taking the limit as $dt$ goes to 0 gives the *chemical master equation* (CME):

$$\frac{\partial P(\vec{x}, t|\vec{x}_0, t_0)}{\partial t} = \sum_{j=1}^{M} [P(\vec{x} - \vec{v}_j, t - dt|\vec{x}_0, t_0) a_j(\vec{x} - \vec{v}_j) - P(\vec{x}, t|\vec{x}_0, t_0) a_j(\vec{x})] \tag{3.39}$$

### 3.4.3 Stochastic Simulation Algorithm

The integral to the CME gives a probability that captures the time evolution of the state probabilities for the genetic circuit. However, computing this integral is typically infeasible for most genetic regulatory circuits due to the fact that Equation 3.39 is a set of coupled ordinary differential equations for each system state and most realistic systems have state spaces that are infinite. Due to this infeasibility, methods to approximate the CME have been developed including the SSA.

The SSA is an algorithm that generates a time course simulation trajectory of a chemical or biochemical system [32, 33]. The main claim to fame of the SSA over other stochastic simulation approaches is that it steps over time steps where no reactions occur. Algorithm 3.1 presents an implementation of the SSA algorithm known as the direct method. This algorithm begins by initializing the simulation time to $t_0$ and the current state vector which holds counts for each molecule to $\vec{x}_0$ (line 1). The algorithm then enters a loop (lines 2-11) where the state vector is updated and time is advanced until time reaches the time limit $T$. During each iteration of this loop, the propensity function of each reaction is computed and summed (lines 2-4), two random numbers are drawn (line 5), the time of the next reaction and the next reaction are determined using these random numbers and the propensity functions (lines 6-7), and the state vector and time are updated and recorded (lines 8-9).

Since the development of the SSA, there have been many variations and improvements to the algorithm. Among these are the first-reaction method, the next-reaction method, the SSA-CR method, tau-leaping, ssSSA, and wSSA. Each method is described briefly below.

The first-reaction method is the same as the direct method except instead of generating two random numbers, this method generates a random number $r_i$ for each reaction. The time until each reaction is then calculated as $\tau_i = \frac{1}{a_i(\vec{x})} \ln\left(\frac{1}{r_i}\right)$ and the reaction $R_i$ with the smallest corresponding $\tau_i$ is selected and fired. This method is exact just like the direct method and performs nearly as well except for that fact that if the system has more than two reactions, then it must draw more random numbers, a process that can sometimes be expensive [32].

An improvement to the first-reaction method is the next-reaction method. This method takes advantage of the fact that the propensities of only a few reactions change after any given reaction fires. Instead of recalculating all of the $\tau_i$'s and $a_i(\vec{x})$'s, this method stores these values so that they do not all have to be recalculated every time step. In

**Algorithm 3.1**: SSA(Initial state $\langle \vec{x}_0, t_0 \rangle$; Reactions $R_{i \in 1 \ldots m}$; Time limit $T$)

1 Initialize the current simulation time $t = t_0$ and the current state vector $\vec{x} = \vec{x}_0$.
2 **forall** Reactions $R_{i \in 1 \ldots m}$ **do**
3  Evaluate the propensity functions $a_i(\vec{x})$ at state $\vec{x}$.
4 Evaluate the sum of all the propensity functions:

$$a_0(\vec{x}) \quad = \quad \sum_{i=1}^{m} a_i(\vec{x})$$

5 Draw two unit uniform random numbers, $r_1$, $r_2$.
6 Determine the time, $\tau$, until the next reaction:

$$\tau \quad = \quad \frac{1}{a_0(\vec{x})} \ln\left(\frac{1}{r_1}\right)$$

7 Determine the next reaction, $R_\mu$, where $\mu$ is the smallest integer satisfying

$$\sum_{i=1}^{\mu} a_i(\vec{x}) > r_2 a_0(\vec{x})$$

8 Determine the new state after firing $R_\mu$: $t = t + \tau$ and $\vec{x} = \vec{x} + \vec{v}_\mu$ where $\vec{v}_\mu$ is the update vector after firing $R_\mu$.
9 Record $(\vec{x}, t)$.
10 **if** $t < T$ **then**
11  Go to step 2.

addition, the $\tau_i$'s are changed to incorporate the value of $t$ so that they represent real simulation times instead of relative times. In order to keep track of which values need to be updated during a time step, a reaction dependency graph is used to keep track of which reactions are affected when a reaction fires. Finally, every time a reaction fires, all of the unaffected $\tau_i$'s are renormalized and a priority queue is used so that updating and finding the next reaction to fire is easy and efficient. As long as the priority queue is indexed and implemented well, this method significantly improves the efficiency of SSA as the number of species and reactions goes up because it only requires that the propensities and next-fire times of affected reactions be updated and the next-fire times of all other reactions be renormalized [30].

Using a technique known as composition and rejection, the SSA can be improved to a constant time algorithm regardless of the number of reactions. This improvement is known as the SSA-CR and works by drawing four random numbers instead of two and segregates the reactions into groups based on how large their propensities are (this segregation is

known as the composition step). The first random number is still used to select which time the next reaction should be fired; however, the second random number is used to select a group to draw a reaction from. The third random number is then used to pick a reaction from the selected group. If the selected reaction's propensity is larger than the final random number times the largest propensity in the group, then the reaction is fired. Otherwise, the third and fourth random numbers are redrawn and a new reaction is potentially selected to be fired (this redrawing is known as the rejection step). After firing the reaction, the groupings are updated based on the newly updated propensities and the algorithm repeats.

To further improve simulation efficiency, the tau-leaping method selects $\tau$ such that multiple reactions fire in each time increment. This selection is done by the introduction of $m$ random functions $K_i(\tau, \vec{x}, t)$ (one for each reaction) that determine how many times each reaction $R_i$ fires in the time increment $[t, t + \tau]$. Since all of these random functions are dependent on each other making them very difficult to compute, they are often approximated with Poisson random variables taking into account the leap condition that ensures that $\tau$ is small enough that none of the propensities change by a significant amount. The value of $\tau$ is also limited by the fact that if it is too large then too many critical reactions may fire causing a species count to become negative. Indeed, the main challenge of the tau-leaping method is to pick $\tau$ such that enough reactions occur in each time step to speed up simulation while ensuring that the leap condition is not violated and that too many critical reactions do not fire. To solve this problem, the tau-leaping method introduces an accuracy control parameter $0 < \epsilon << 1$ that helps determine $\tau$. If accuracy is more of a concern, then $\epsilon$ should be set to a lower value; however, if runtime efficiency is desired, then $\epsilon$ should be set to a higher value. For reasonable values of $\epsilon$, the tau-leaping method outperforms exact SSA methods while providing a fairly accurate approximation of the system's behavior [36].

Even with tau-leaping, many systems with both fast and slow reactions, also known as stiff chemical systems, perform poorly. Inspired by the Michaelis-Menten approximation in deterministic chemical kinetics, the ssSSA is an efficient approach to dealing with such systems. It proceeds by first partitioning reactions into two groups: reactions with large propensities (i.e., fast reactions) and reactions with small propensities (i.e., slow reactions). It then continues by partitioning species into fast species if their counts are changed by any fast reaction and slow species otherwise. Next, the algorithm creates a virtual fast

process that represents the fast species counts evolving under only the fast reactions with none of the slow reactions. Finally, the ssSSA then simulates only the slow reactions, using specially modified propensity functions taking into account the virtual fast process. Although slow-scale SSA is an approximation of the exact methods, it has been shown to decrease simulation time when compared to the exact SSA by two to three orders of magnitude, with no perceptible loss of simulation accuracy [13].

Sometimes there are interesting events in a system that rarely occur. Even with the various improvements to the SSA listed above, it can still take an extremely large amount of simulation runs to observe these rare events. This fact led to the development of the wSSA, which uses *importance sampling* techniques to determine the probability of rare events occurring. Basically, this method works by guiding the simulation to be more likely to produce a run where the rare event occurs. It then weights the sample paths by a likelihood factor to produce a statistically correct and unbiased result.

## 3.5 Reaction-Based Abstractions

Even with all of the improvements to the SSA, analysis of genetic regulatory circuits using SCK can still be very computationally intensive. One way to alleviate this problem is to try to reduce the model to a smaller, less complex model that still preserves the behavior of the original model. This reduction can be done by applying abstractions to the model before it is analyzed. *Reaction-based abstractions* attempt to reduce the number of species and reactions in the model. These abstractions typically improve simulation time as they attempt to eliminate time scale separation in the model. Namely, they try to remove fast reactions that slow down the simulation because they fire often preventing the simulation from making significant progress.

There are many reaction-based abtractions [53]. Among these are a collection of abstractions that are well suited to work with chemical reaction networks generated from genetic regulatory circuits. These abstractions include *operator site reduction*, *complex formation abstraction*, *sequestering abstraction*, and *reaction splitization*.

In reaction networks representing genetic regulatory circuits, there are often a lot of fast reversible reactions for the binding of RNAP and either an activator or repressor to a promoter's operator site. The goal of operator site reduction is to remove these fast reactions so that the analysis can focus on interesting reactions in the system instead of spending computation time on these uninteresting reactions. This abstraction begins by first determining the species that represent the operator sites of the network (i.e.,

the promoters). It then finds how many different binding configurations the operator site can be in and computes a *quasi-steady-state approximation* value for each of these bindings. This approximation is essentially the values of the species counts of the species that form each binding multiplied by an *equilibrium constant* for the reaction (the forward rate constant over the reverse rate constant). The sum of these values and 1 become the denominator of the new reaction rates for the reduced reactions. For each binding reaction that leads to production of another species, the reduced reaction rate is the product of the rate of the original production reaction, the total species count of the operator, and the quasi-steady-state approximation value for the binding over the summation described above. The function rate($p$), shown in Equation 3.40, can be derived which returns the rate of production initiated from promoter $p$.

$$\text{rate}(p) = \begin{cases} \dfrac{n_p k_p |p| K_o |RNAP|}{1 + K_o |RNAP| + \sum_{s_r \in \text{Rep}(p)} (K_r |s_r|)^{n_c}} & \text{if } |\text{Act}(p)| = 0 \\ n_p k_b |p| K_o |RNAP| + \sum_{s_a \in \text{Act}(p)} n_p k_a |p| K_{oa} |RNAP| (K_a |s_a|)^{n_c} & \\ \dfrac{}{1 + K_o |RNAP| + \sum_{s_r \in \text{Rep}(p)} (K_r |s_r|)^{n_c} + \sum_{s_a \in \text{Act}(p)} K_{oa} |RNAP| (K_a |s_a|)^{n_c}} & \text{otherwise} \end{cases} \tag{3.40}$$

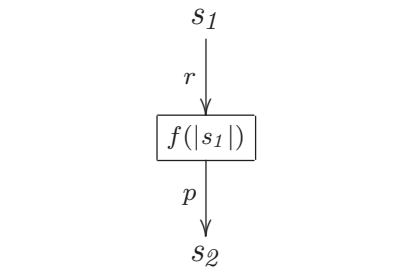
The rate($p$) function is made up of constants which can be found in Table 3.1 and variables for the repressing species, $s_r$, and activating species, $s_a$, for this promoter. This function is broken down into two cases. The first case is for a promoter that does not have any species which are activating it. In this case, it is assumed that the promoter is *constitutive* which simply means that it can initiate transcription at a significant rate without the aid of another activating species. Assuming that there are no repressor molecules present, this rate is approximately $n_p k_p |p|$ where $n_p$ is the number of proteins produced per mRNA produced, $k_p$ is the transcription rate for a constitutive promoter, and $|p|$ is the number of copies of the promoter and gene. However, this rate is reduced as the number of repressor molecules, $s_r$, increases. The second case is for a promoter that must be activated for significant transcription. Assuming that there are no activator or repressor molecules present, the rate of production of this promoter is $n_p k_b |p|$ where $k_b$ is a low basal rate of production which is typically much smaller than $k_p$. In this case, as the number of activator molecules, $s_a$, increases so does the rate of production from this promoter. Like the first case, this production can also potentially be inhibited, if there exist species which can repress this promoter [64].

If there are complex formation reactions between species and chemical inducers such as that between LacI and IPTG, complex formation and sequestering abstractions can be applied to the model. These abstractions are related as they both deal with complex formation reactions. The complex formation abstraction uses a steady state approximation to remove complexes from the original model. This abstraction replaces the value of a complex, $c_i$, in a rate function with the expression $K_c s_i s_j$, where $s_i$ and $s_j$ are species that bind to form $c_i$ and $K_c$ is the complex formation equilibrium constant. For instance, applying this abstraction to complex C1 in the genetic toggle switch would replace each instance of it with $K_c|\text{LacI}||\text{IPTG}|$.

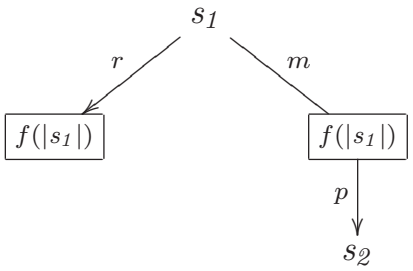
The sequestering abstraction, on the other hand, uses the quasi-steady-state approximation in addition to the law of mass conservation and replaces the value of a species, $s_i$, in a rate function with the expression $\frac{s_{i_{total}}}{1+K_c s_j}$, where $s_{i_{total}}$ is the variable for the total amount of the species (free and in complex) and $s_j$ is the variable for the other species that binds with $s_i$ to form the complex. This rate shows that as the amount of $s_j$ increases, the effective amount of $s_i$ decreases. In the genetic toggle switch, everywhere TetR appears in a rate equation, it is replaced by $\frac{|\text{TetR}|}{1+K_c|\text{aTc}|}$ when using this abstraction.

Reaction splitization is useful in cases where the analysis requires that all reactions only have at most one reactant or product. This type of abstraction is especially beneficial when performing analysis where only one of the species values can change at a time. This abstraction works by creating a copy of a reaction for each reactant or product that is involved in the reaction. For each of these new reactions, it assigns one of the reactants or products to the reaction and all other reactants become modifiers. An example of how this abstraction works with a reaction that has a single reactant and a single product is shown in Figure 3.4. Additionally, this abstraction is illustrated in Figure 3.5 for a reaction with more than one reactant and in Figure 3.6 for a reaction with more than one product.

Applying the operator site reduction and sequestering abstractions to the genetic toggle switch reaction network shown in Figure 3.2, the model is reduced from 14 species and 13 reactions to 5 species and 5 reactions. This resulting reaction graph is shown in Figure 3.7.

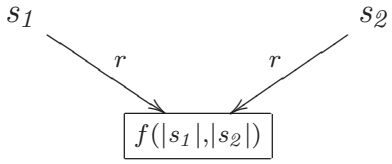



(a)

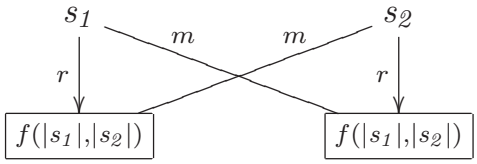



(b)

**Figure 3.4**: Single reactant single product reaction splitization: (a) original reaction and (b) split-up reactions.




(a)




(b)

**Figure 3.5**: Multiple reactants reaction splitization: (a) original reaction and (b) split-up reactions.

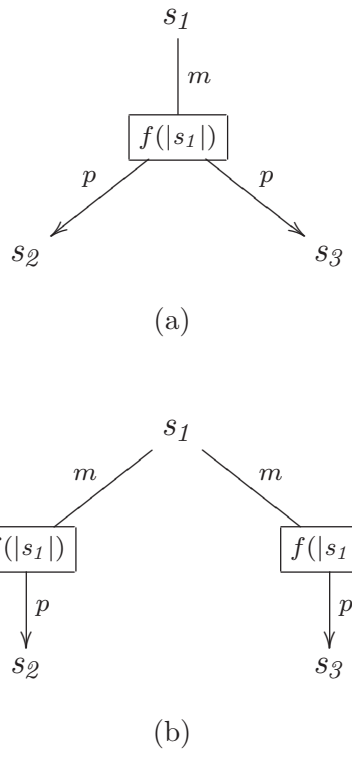



**Figure 3.6**: Multiple products reaction splitization: (a) original reaction and (b) split-up reactions.

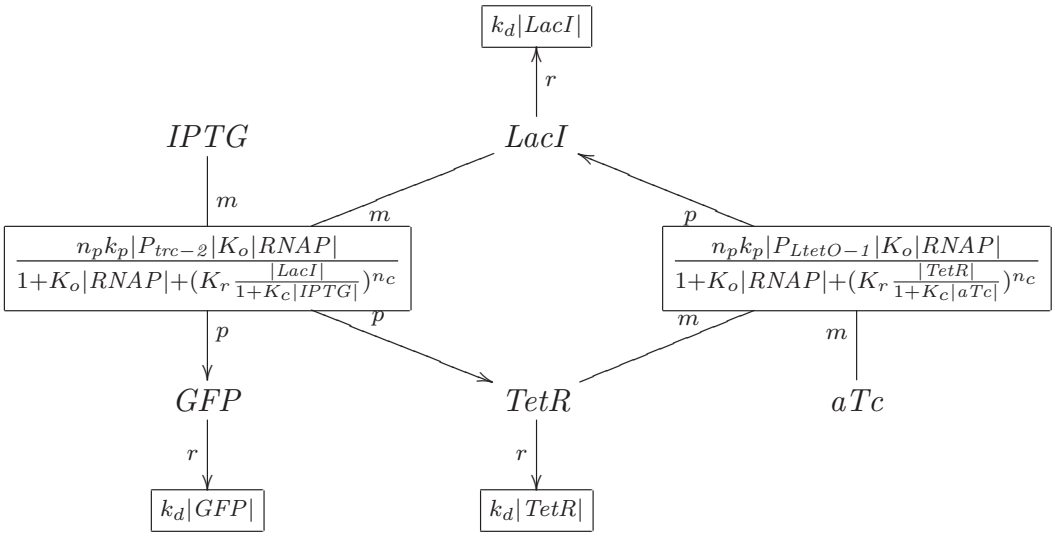



**Figure 3.7**: Reaction graph for the genetic toggle switch after applying reaction-based abstractions to the chemical reaction network. These abstractions reduce the model from a model of 14 species and 13 reactions to a model of 5 species and 5 reactions.

# CHAPTER 4

# INCREMENTAL STOCHASTIC SIMULATION ALGORITHM

When analyzing a genetic regulatory circuit, researchers are often interested in the behavior of the circuit. In order to discern this behavior, researchers perform various types of analyses including ODE and SSA simulations. These types of analysis can tell a researcher a lot about a system; however, they often are unsuccessful in producing results that show the "typical" behavior of the circuit. This inability to easily discern the circuit's typical behavior has led to the development of the iSSA, a variant of SSA which attempts to produce a simulation trace describing the typical behavior of a chemical reaction network by performing many simulation runs in small time increments. The iSSA then uses the resulting states of the runs in the current time increment to constrain the selection of starting states in the next time increment. Depending on the model being analyzed, different variants of the iSSA can be utilized to analyze it by altering the parameters or functions used by the algorithm.

Section 4.1 motivates the iSSA and gives the intuition behind the iSSA algorithm. Section 4.2 details how the iSSA algorithm simulates a chemical reaction network. Section 4.3 presents a variant of the iSSA known as the *marginal probability density evolution* (MPDE) method. Section 4.4 introduces another variant of the iSSA known as the *mean path* method. Section 4.6 presents an adaptive approach to selecting the time increment of the iSSA algorithm. Section 4.5 describes a method similar to the mean path method known as the *median path* method. Finally, Section 4.7 exhibits an approach to iSSA that allows it to produce multiple typical paths.

## 4.1 Algorithm Intuition

Typically, whenever a researcher wants to determine the behavior of a genetic regulatory circuit, he or she analyzes it using either ODE or SSA simulation. These types of analysis produce time series data that the researcher can then examine to observe the

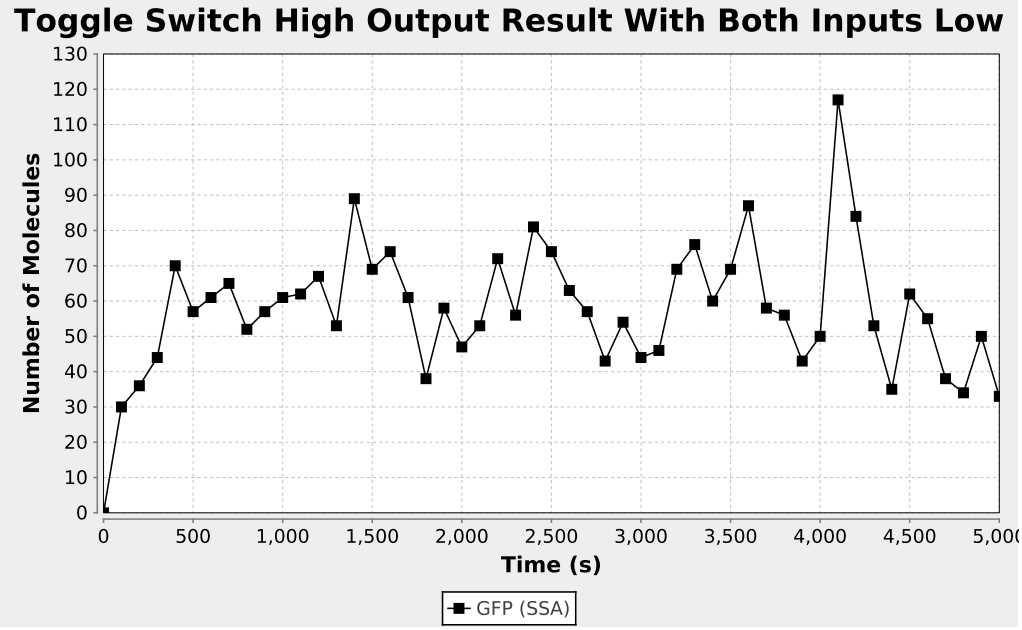
behavior of the circuit. However, these simulation techniques often produce results that hide the typical behavior of the system. This cloaking of the typical behavior is due to the fact that the deterministic nature of ODE simulation prevents it from capturing stochastic events which can be essential to the function of the genetic regulatory circuit. Although SSA simulation is capable of capturing stochastic events, in order to determine statistics on the typical behavior of a circuit, researchers perform many simulation runs and take the average of these runs as the typical behavior. The problem with this approach is that there is often a lot of noise in genetic regulatory circuits and stochastic events often occur at different time points from simulation to simulation. This averaging causes a smoothing effect where the typical behavior is washed out by noise and time-shifted stochastic events.

Simulation results for the toggle switch are expected to select either the ON or the OFF state when the circuit is initially set to a state where all the signals are low. Figure 4.1 shows that individual SSA simulation runs capture this expected behavior. However, Figure 4.2 shows an ODE simulation of the toggle switch where the circuit neither switches ON or OFF but instead stabilizes at an intermediate state. Similarly, the plot of the average of 100 SSA simulations show the toggle switch going to a false intermediate state. This plot also shows that the average of 100 runs suffers from the smoothing effect and fails to capture the noisy fluctuations observed in the original circuit.

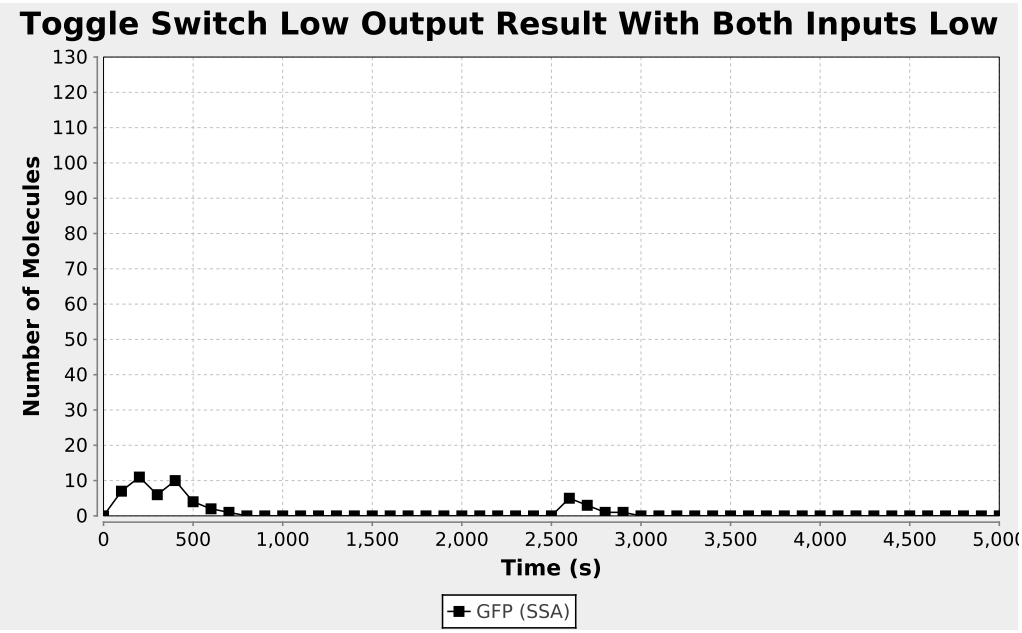
The goal of the iSSA, on the other hand, is to produce time series data that represents typical behavior of a chemical reaction network. The main idea behind the iSSA is to perform stochastic simulation runs in small time increments [84, 54]. At the end of each time increment, statistics are computed over all of the simulation runs. These statistics are then used in the next time increment to constrain and select a new starting state for each run. An example of how this process works in shown in Figure 4.3. In this figure, there are five simulation runs that are run up to a certain time point (represented by the vertical bars). At this point, the simulations stop and statistics are computed over the states of the system. A new starting state is then selected for each simulation, and this process continues until the time limit for the entire simulation is reached.

## 4.2 iSSA Algorithm Overview

The iSSA algorithm, presented in Algorithm 4.1, is essentially a wrapper around Gillespie's SSA that continually starts and stops several simulation runs at the beginning and end of each time increment. The iSSA takes as parameters a maximum number of simulation runs (`maxRuns`), a simulation time limit (`timeLimit`), an initial state-vector and




(a)




(b)

**Figure 4.1**: Individual SSA results for the genetic toggle switch initialized to a state where aTc, IPTG, LacI, TetR, and GFP are low. (a) Plot showing an individual SSA run where the circuit switches to the ON state. (b) Plot showing an individual SSA run where the circuit switches to the OFF state.

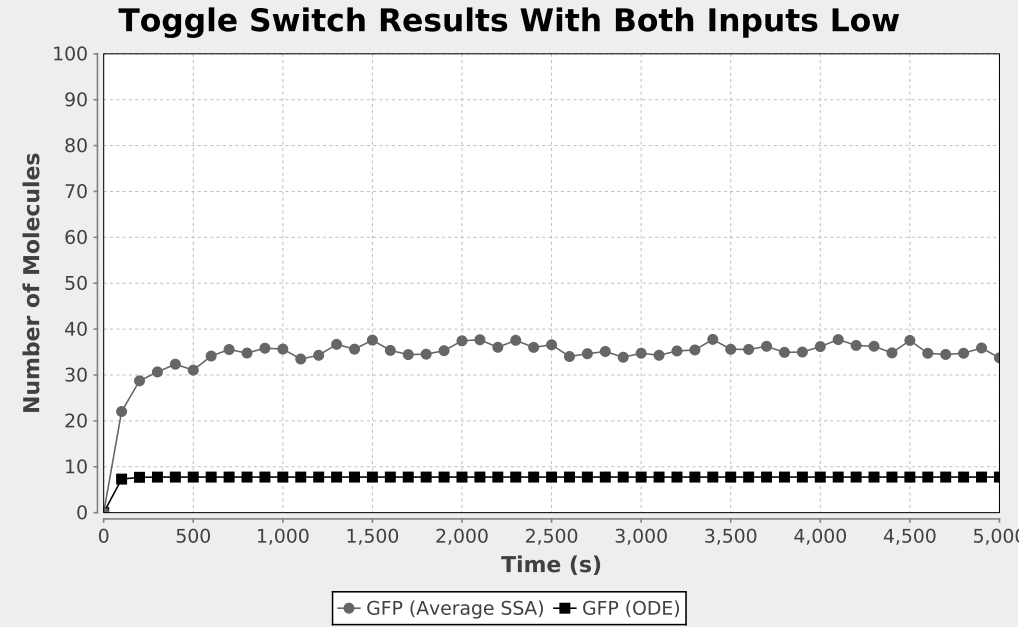



**Figure 4.2**: ODE results and the mean $\overline{x}(t)$ of 100 SSA runs for the genetic toggle switch initialized to a state where aTc, IPTG, LacI, TetR, and GFP are low.

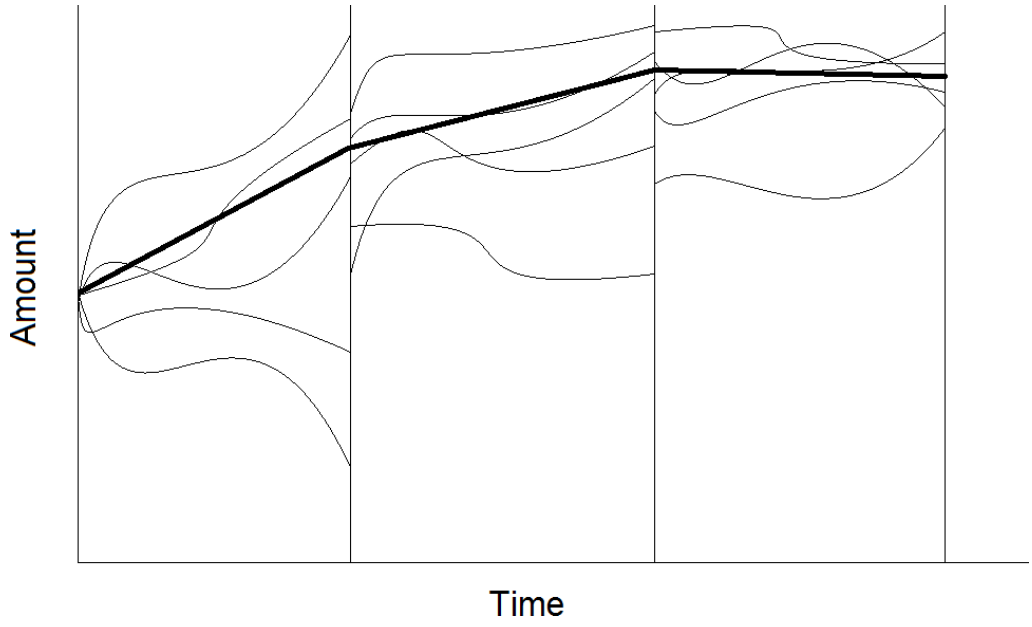



**Figure 4.3**: Illustration showing the incremental nature of the iSSA. The end of each time increment and beginning of the subsequent increment is indicated by the vertical bars. As this figure shows, simulations run continuously through an increment but are stopped at the end of the increment and may be started up again from a different state in the next increment.

**Algorithm 4.1**: `iSSA`(Maximum number of runs `maxRuns`; Time limit `timeLimit`; Initial state $\langle \vec{x}_0, t_0 \rangle$; Reactions $R_{l \in 1 \ldots m}$)

1 Initialize: $k = 1$ and $\mathbf{X}^{(0)} = \langle \vec{x}_0, t_0 \rangle$.
2 Set $i = 1$.
3 Set $\langle \vec{x}, t \rangle = \texttt{select}(\mathbf{X}^{(k-1)})$ and $\texttt{start} = t$.
4 Set $\texttt{limit} = \texttt{findLimit}(\texttt{start}, \vec{x}, t)$.
5 Execute a Gillespie SSA step:
6 **forall** Reactions $R_{l \in 1 \ldots m}$ **do**
7 Evaluate the propensity functions $a_l(\vec{x})$ at state $\vec{x}$.
8 Evaluate the sum of all the propensity functions:

$$a_0(\vec{x}) \quad = \quad \sum_{l=1}^{m} a_l(\vec{x})$$

9 Draw two unit uniform random numbers, $r_1$, $r_2$.
10 Determine the time, $\tau$, until the next reaction:

$$\tau \quad = \quad \frac{1}{a_0(\vec{x})} \ln\left(\frac{1}{r_1}\right)$$

11 Determine the next reaction, $R_\mu$, where $\mu$ is the smallest integer satisfying

$$\sum_{l=1}^{\mu} a_l(\vec{x}) > r_2 a_0(\vec{x})$$

12 Determine the new state: $t = t + \tau$ and $\vec{x} = \vec{x} + \vec{v}_\mu$.
13 **if** $t < \texttt{limit}$ **then**
14 Go to step 4.
15 $\texttt{record}(\mathbf{X}^{(k)}, \vec{x}, t, i)$.
16 **if** $i < \texttt{maxRuns}$ **then**
17 Set $i = i + 1$.
18 Go to step 3.
19 **if** $t < \texttt{timeLimit}$ **then**
20 Set $k = k + 1$.
21 Go to step 2.

system time $\langle \vec{x}_0, t_0 \rangle$, and a collection of reactions $R_{l \in 1 \ldots m}$. The iSSA begins by initializing the record table where ending states are stored ($\mathbf{X}$) to the initial state and system time (line 1). At the start of each $k^{\text{th}}$ increment, the run number, $i$, is reset to 1 (line 2). Next, a starting state is selected using the `select` function along with the starting time for each run in the increment (`start`) (line 3). In addition to computing the starting state for the increment, the ending time for the increment, `limit`, is computed using the `findLimit` function (line 4). At this point, the iSSA executes a step of Gillespie's SSA where it selects a random time and reaction event and computes a new state (lines 5-12). The newly computed simulation time is compared against `limit` to determine if the algorithm needs to compute another step of the SSA (line 13). If this limit has not been reached, then the algorithm recomputes `limit` and another SSA step is performed (line 14). Otherwise, the algorithm records the current time and state for the current run by calling the `record` function (line 15). If the maximum number of runs has not been reached, a new simulation run is started from the original state chosen by the `select` function (lines 16-18). Once all the runs have been completed for the time increment, the algorithm determines if it has reached the end of the simulation time or if it needs to compute another time increment (lines 19-21). Finally, the iSSA returns the sequence of states chosen by the `select` function that represents a "typical" simulation trace of the model.

The iSSA requires definitions for three functions in order for it to operate. These functions are `select`, `findLimit`, and `record`. The `select` function selects a starting state for a simulation run from statistics computed on the ending states in the record table (line 3). The `findLimit` function computes the ending time for the current increment given the starting time, the current state, and the current time (line 4). The `record` function records simulation data in the record table (line 15). These three functions can be defined in a number of alternative ways to produce specialized forms of the iSSA. Each specialized iSSA method delivers different statistical information. For example, the iSSA reduces to the SSA when the `findLimit` function returns `timeLimit`, the `select` function sets $\vec{x}$ to $\vec{x}_0$, and the `record` function tracks raw simulation data. This reduction is apparent in Figure 4.4 where the iSSA is tuned to output SSA runs.

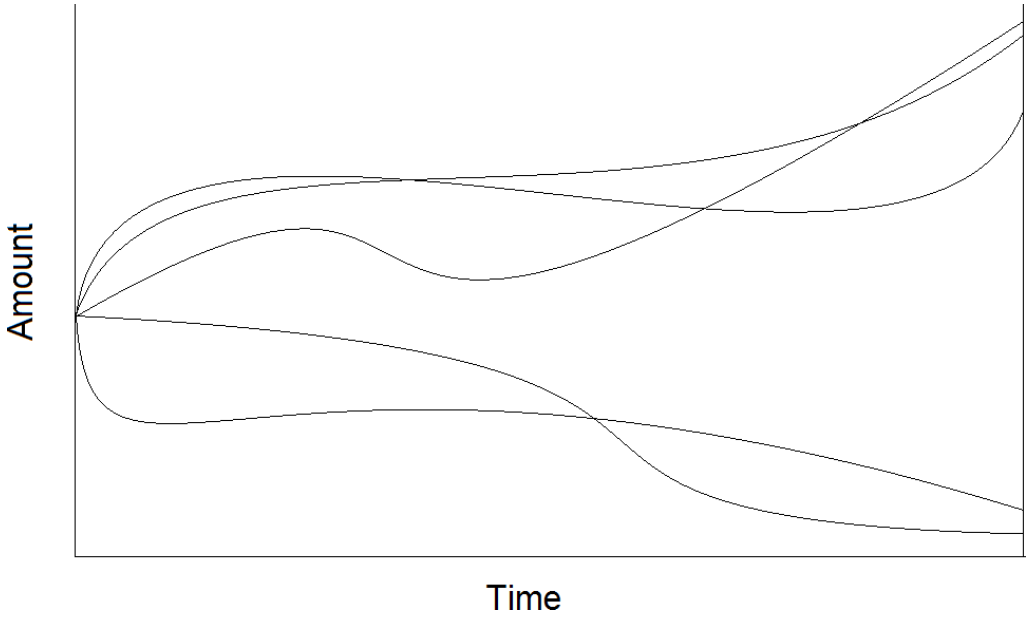



**Figure 4.4**: Illustration showing the iSSA reducing to the SSA when there is a single time increment equal to the time limit of the simulation and raw simulation data is tracked.

## 4.3 iSSA Using Marginal Probability Density Evolution

The MPDE method tracks the statistical evolution of each species in the reaction system. It is derived from the Chemical Langevin Equation (CLE), a further approximation to the tau-leaping method [31, 35]. The basic idea of how this method is derived is to apply the CLE method over a short time-increment. At the end of the increment, the CLE produces a collection of vectors of Gaussian-distributed random values. Summing the Gaussians together yeilds a joint Gaussian distribution that can be fully characterized by its mean and variance. This derivation means that the MPDE method generates a probability distribution whereas the traditional SSA generates a scalar value for each species at each time increment. It then uses this distribution to select starting states at the beginning of each time increment and uses the ending states to compute a new distribution. A plot showing how this variant of iSSA works is shown in Figure 4.5. This figure shows MPDE with the number of runs set to five. The MPDE method produces different starting points for each run in each time increment because each starting point is drawn from a distribution of the ending points from the previous increment.

The MPDE method is defined by the `findLimit`, `record`, and `select` functions as follows. The `findLimit` function simply returns `start` + `increment`, the sum of the start

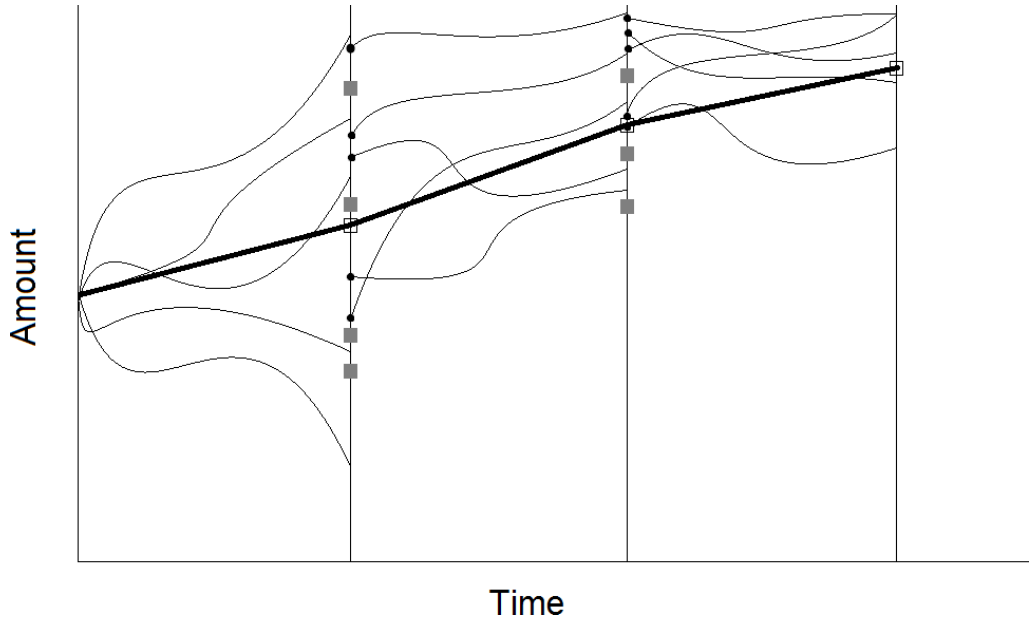



**Figure 4.5**: Illustration showing how the MPDE algorithm performs simulations. At the start of each time increment, the algorithm draws a starting state from the distribution computed from the ending states of the previous increment. The squares indicate states that are rejected due to violating correlation constraints and the circles indicate valid states.

time and a constant predetermined time interval value that is specified as a parameter at the beginning of the simulation.

Within interval $k$, at the completion of simulation run $i$, the `record` function stores the state $\vec{x}$ and the system's time $t$ at row $i$ in the record table $\mathbf{X}^{(k)}$. Once `maxRuns` is reached, the average and standard deviation of the columns of $\mathbf{X}^{(k)}$ are computed and used to approximate a Gaussian distribution. This distribution is used to estimate the *marginal probability density function* (PDF) for each species in the system.

The `select` function uses the previously-estimated PDF to randomly generate a new starting state $\vec{x}$. Let $x_j$ be the $j^{\text{th}}$ member of $\vec{x}$. Then $x_j$ is generated by randomly drawing a value from the distribution $f_j\,(x_j;\, k-1)$. By generating the $x_j$'s independently, MPDE implicitly assumes that all species are pair-wise conditionally independent, given the states of all other species in the system. However, most genetic circuits contain some pairs of highly-correlated or dependent species. In these cases, MPDE can be used if known correlations are stated explicitly as constraints in the reaction model. For example, a promoter may have three species associated with it: one where it is empty, one where it is bound to RNAP, and one where it is bound to a repressor molecule. Assuming that the

number of promoters is constant in the system, the sum of these species should always be exactly equal to the initial number of promoters. Because of this condition, the `select` function must be implemented so that it rejects any $\vec{x}$ that violates this constraint. This rejection is illustrated in Figure 4.5 where the squares indicate states that are generated from the distribution but are rejected because they violate these constraints. Finally, since `findLimit` always returns a constant value for the time increment, the `select` function simply returns the ending time of the previous time increment.

Figure 4.6 shows simulation results of the genetic toggle switch using the MPDE method with 100 stochastic runs. These results show that the MPDE simulation selects the OFF state for the circuit, and correctly reveal one of the circuit's typical behaviors as opposed to the ODE and average SSA results presented in Figure 4.2.

## 4.4 iSSA Using Mean Path

While the MPDE method performs well in a variety of examples, it relies on a statistical approximation that limits the conditions under which it can be trusted. When there are highly-correlated or dependent species in the genetic circuit, correlation constraints must be specified or the MPDE may return erroneous results. However, when they are specified,

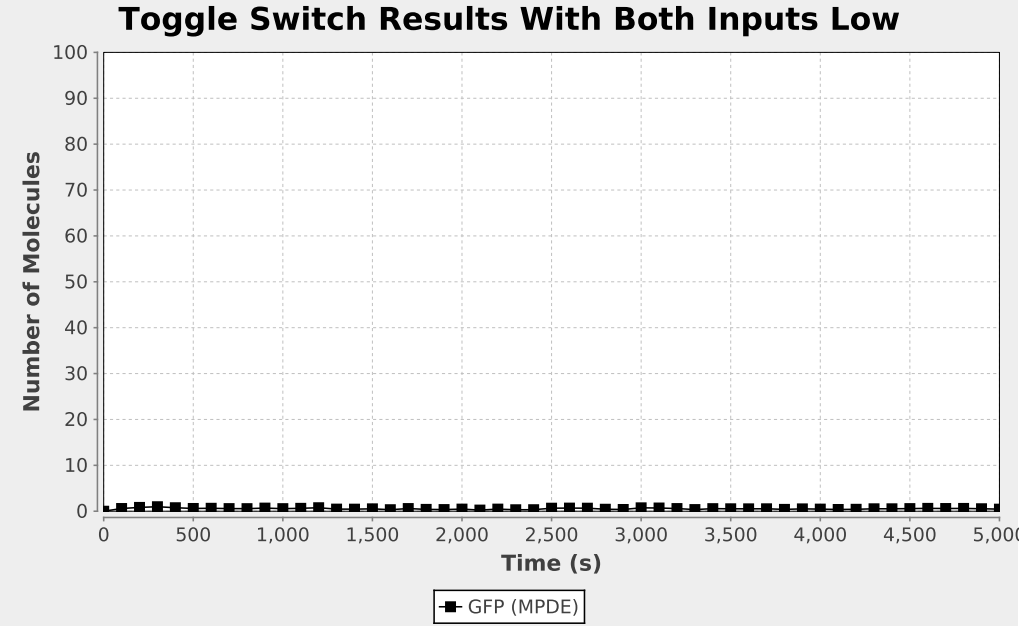



**Figure 4.6**: MPDE simulation results for the genetic toggle switch. These results use a time increment of 10 seconds with 100 runs for each time increment.

the MPDE often spends a large amount of time generating and rejecting starting states that violate these constraints.

Another disadvantage of MPDE is that it is unable to be used in tandem with reaction-based abstractions. This disadvantage due to the fact that the amount of dependent species is increased when the model is reduced by these abstractions. Since these dependencies are introduced by the abstractions, they are not explicitly defined as part of the model, and MPDE likely produces erroneous results. For example, Figure 4.7 shows a simulation of the genetic toggle switch using the MPDE method with abstraction. In this plot, the MPDE method erroneously predicts that the circuit starts in the OFF state and begins moving towards the ON state. MPDE, therefore, tends to be less attractive for use with abstracted simulation models.

In contrast to MPDE, the mean path method does not rely on a statistical approximation and is able to be used in conjunction with reaction-based abstractions. The result of the mean path method, however, still uses a statistical evolution approach to try and capture the typical behavior of the system. Figure 4.8 presents a plot indicating how a mean path simulation is carried out. At the end of each time increment in mean path, an

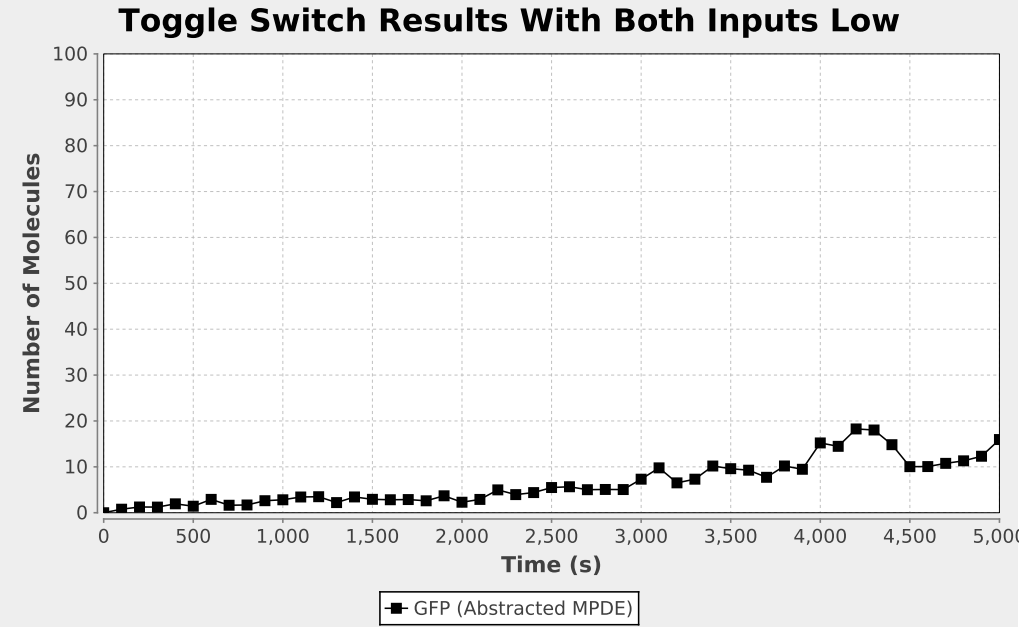



**Figure 4.7**: Abstracted MPDE simulation results for the genetic toggle switch. These results use a time increment of 10 seconds with 100 runs for each time increment. As seen in this plot, using abstraction with MPDE causes the simulation to produce erroneous results.

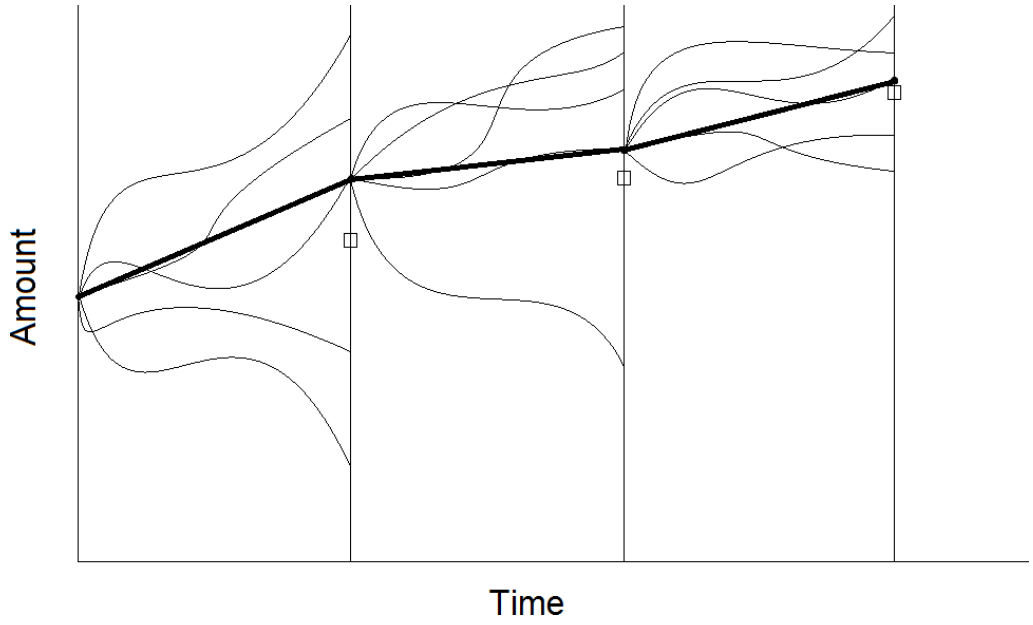



**Figure 4.8**: Illustration showing how the mean path algorithm performs simulations. At the end of each increment, the algorithm computes the average state over all the end states (indicated by a square). The state that is closest to this average state is then selected (indicated by the circle), and each simulation run uses this state as its starting state in the next increment.

average state is computed over all the ending states. Using a Euclidean distance measure, the state that is closest to this average state is selected and used as the starting state for each simulation run in the subsequent increment. When the simulation finally reaches its time limit, the method outputs a time course trace of the selected states. Since this method outputs a trace of states that are actually found during simulation, the mean path is a "real path method" which means that it generates a simulation trace that could feasibly be generated by simulating the model using the SSA.

The mean path method is defined by the `findLimit`, `record`, and `select` functions, as follows. As with the MPDE method, the `findLimit` function returns the sum of the start time and the constant that is defined for the time increment at the beginning of the simulation. The `record` function stores the SSA states and times in the record table, $\mathbf{X}^{(k)}$. After completing all runs for a time increment, the average state, $\overline{\vec{x}}^{(k)}$, is computed by averaging down each column of $\mathbf{X}^{(k)}$. For each row $\vec{x}_i^{(k)}$ in $\mathbf{X}^{(k)}$, the square Euclidean distance is computed: $d_i^{(k)} = \left|\vec{x}_i^{(k)} - \overline{\vec{x}}^{(k)}\right|^2$. During interval $k$, the `select` function for the mean path method returns the $\vec{x}_i^{(k-1)}$ for which $d_i^{(k-1)}$ is the smallest. It is this selection

scheme that causes the mean path method to return a simulation trace that could be found by the SSA.

Because mean path selects an actual state from the state table as the starting state in each increment, there is no need for correlation constraints to be added to the model. The mean state is computed by averaging over all simulation runs. A single state is then selected from these statistics which in turn is used to constrain the initial condition of each run in the next time increment. By performing simulation runs in this manner, this algorithm is able to compute meaningful statistics on a genetic circuit while ensuring that the circuit's functional behavior is preserved.

Additionally, since there is no need to throw out invalid states in the selection process, the mean path method is able to perform simulations more efficiently than the MPDE. To illustrate this efficiency, let us compare simulations of the genetic toggle switch using the MPDE method with simulations using the mean path method shown in Figure 4.9. Both MPDE and mean path are able to capture the circuit switching to the OFF state; however, the MPDE method produces these results in 1 minute and 12 seconds while the mean path method takes under 2 seconds.

## 4.5 iSSA Using Median Path

There are some cases where during a time increment, one or more simulation traces may diverge so much that the ending states become outliers in the average state calculation of mean path. In these cases, the mean path may end up selecting a state that does not represent the "typical" behavior of the system due to skewing of the average state. This skewing can cause the mean path to output a simulation trace that does not represent the functional behavior of a genetic circuit. The median path method, another "real path method," avoids this problem by computing the median state instead of the mean state. Like the mean path, it then finds the state with the smallest Euclidean distance from this state and uses this state as the starting state in the next time increment. Figure 4.10 presents a plot indicating how a median path simulation is carried out.

The median path method is functionally the same as the mean path method except for one difference. Instead of the `select` function computing the average state, this function computes the median state, $\widetilde{\vec{x}}^{(k)}$. This computation involves selecting the median value for each species for each run in the $\mathbf{X}^{(k)}$ record table. For each run $\vec{x}_i^{(k)}$ in $\mathbf{X}^{(k)}$, the square

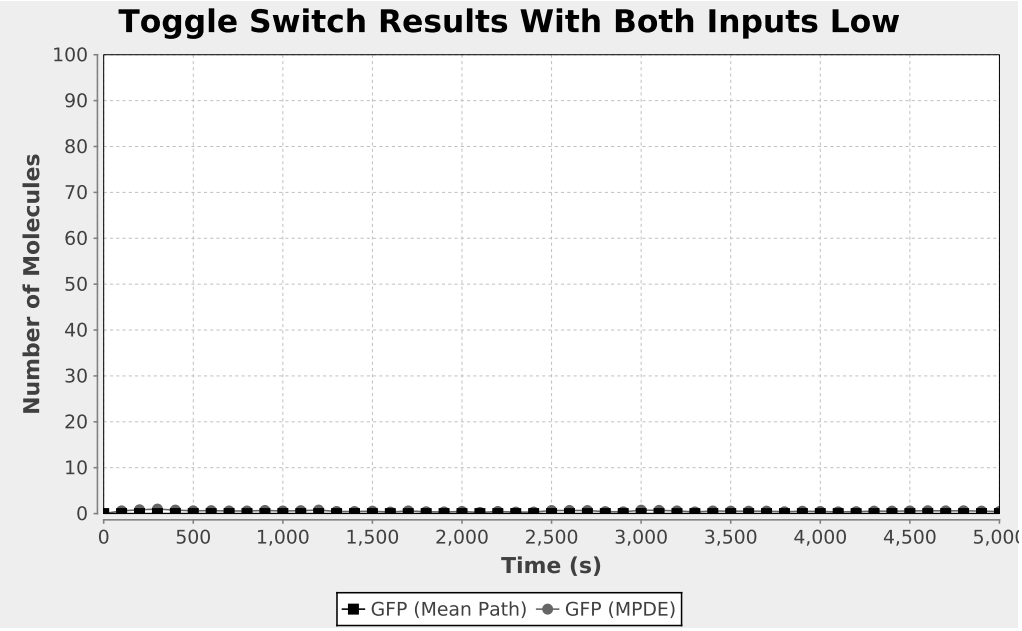



**Figure 4.9**: MPDE and mean path simulation results for the genetic toggle switch. The MPDE and mean path methods both use a time increment of 10 seconds and are obtained with 10 simulation runs. It, however, took the MPDE method 1 minute and 12 seconds to obtain results whereas the mean path method was able to utilize reaction-based abstractions and obtained results in under 2 seconds.

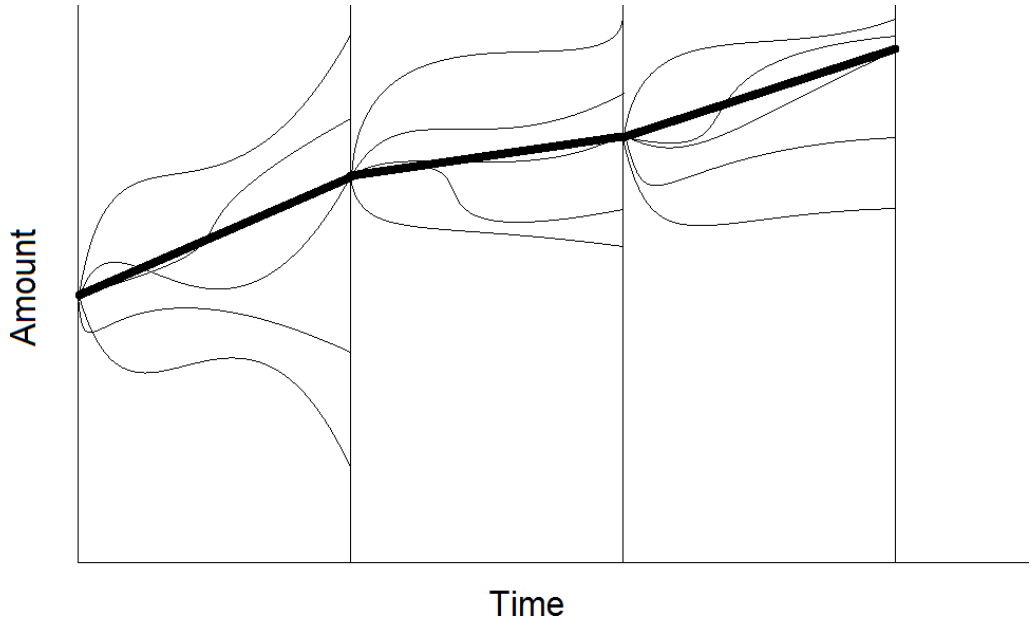



**Figure 4.10**: Illustration showing how the median path algorithm performs simulations. At the end of each increment, the algorithm computes the median state over all the end states. The state that is closest to this median state is then selected, and each simulation run uses this state as its starting state in the next increment.

Euclidean distance is computed: $d_i^{(k)} = \left| \vec{x}_i^{(k)} - \tilde{\vec{x}}^{(k)} \right|^2$, and the `select` function returns the state $\vec{x}_i^{(k-1)}$ for which $d_i^{(k-1)}$ is the smallest.

An example of comparing simulations of the genetic toggle switch produced by the median path method and the mean path method is shown in Figure 4.11. In this figure, the toggle switch starts in the OFF state. The circuit is supposed to switch to the ON state at time point 5,000 and back to the OFF state at time point 15,000 due to changes in the inputs. The mean path method fails to capture these state changes due to outlier runs where the circuit does not respond to the change in inputs. The median path method, however, is robust to these outliers and captures the correct behavior of the circuit. Because the median path method is able to capture the "typical" behavior of a circuit even when there are outlier runs skewing the data, it has proven to be the most robust of the iSSA methods.

## 4.6 Adaptive iSSA

One downside of using the mean and median path methods is that obtaining reasonable results proves to be highly dependent on the choice of time increment. This parameter must be chosen so that an appropriate number of reaction events occurs within each increment. The proper choice of increment depends on the average rate of reactions, which may vary considerably during simulation. Therefore, this method can give very different results based on the choice of this parameter. If this value is too small, then not enough progress is made during each increment and the final simulation trace is fairly flat. On the other hand, if this value is too large, then the final simulation trace exhibits the same washed out behavior as averaging several SSA runs together.

To alleviate the time increment selection problem, the adaptive iSSA dynamically adjusts the time increment to guarantee that a sufficient number of reaction events are seen per time increment. This algorithm facilitates automatic parameter choice and produces more stable results than nonadaptive variants of iSSA. Figure 4.12 shows an example of how the adaptive method is able to dynamically change the time increment. In this figure, simulation traces do not necessarily end at the same time which makes time a factor in computing the representative state in the selection process for the next time increment. This method can be applied to any of the previously presented iSSA methods as it mainly involves modifying the time increment selection for each simulation run.

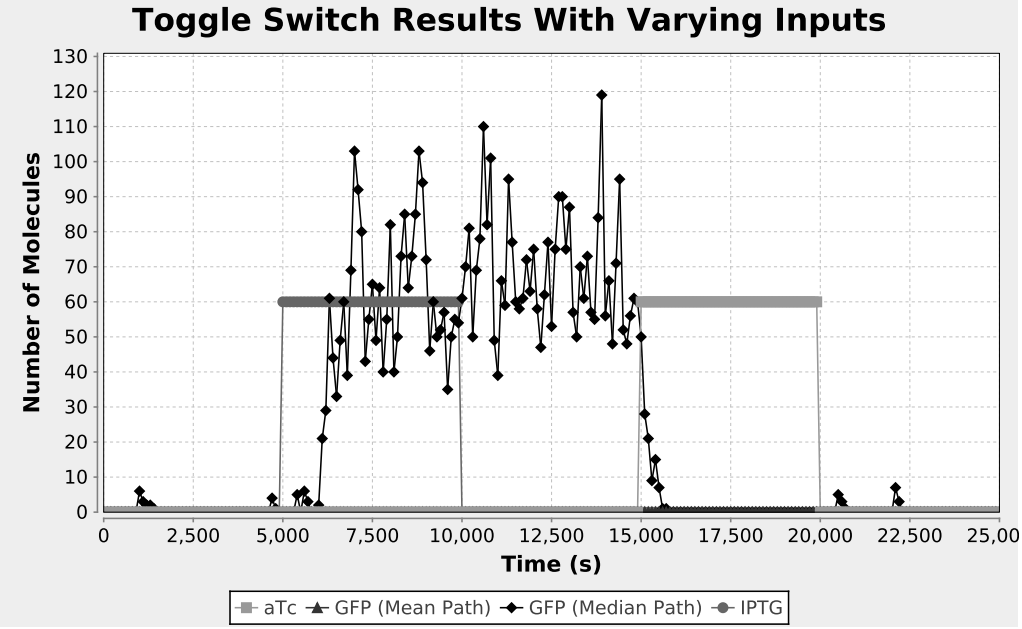



**Figure 4.11**: Mean path and median path simulation results for the genetic toggle switch. This figure shows a comparison of simulating the genetic toggle switch with both the mean path method and the median path method for 100 simulation runs. In this simulation, the toggle switch starts in the OFF state. At time point 5,000 IPTG is applied which should switch it to the ON state. IPTG is removed at time point 10,000, but the circuit should maintain state. The circuit's other input, aTc, is applied at time point 15,000, which should switch the circuit back to the OFF state where it should remain even after the signal is removed at time point 20,000. Due to outlier runs, these results show that the mean path method fails to capture these state changes and outputs a trace that stays at 0 for the entire simulation. The median path method, on the other hand, does not suffer from this problem and is capable of capturing these state changes.

The adaptive iSSA primarily involves changing the definition of the `findLimit` function. There are many ways of defining this function that yield slightly different results. One such approach is to modify the `findLimit` function to count the number of times that it is called. This counting effectively causes it to keep track of how many times a reaction event is fired in the SSA portion of the algorithm. While this number is below a desired number of events, the function simply returns the entire simulation time limit. However, once the desired number is reached, this function returns the current time which causes the algorithm to move on to the next simulation run. This approach can yield some fairly good results, but the optimal number of events can vary greatly from model to model.

Another approach is to modify the `findLimit` function to return the sum of the start time and the number of desired events divided by the sum of all the propensity values

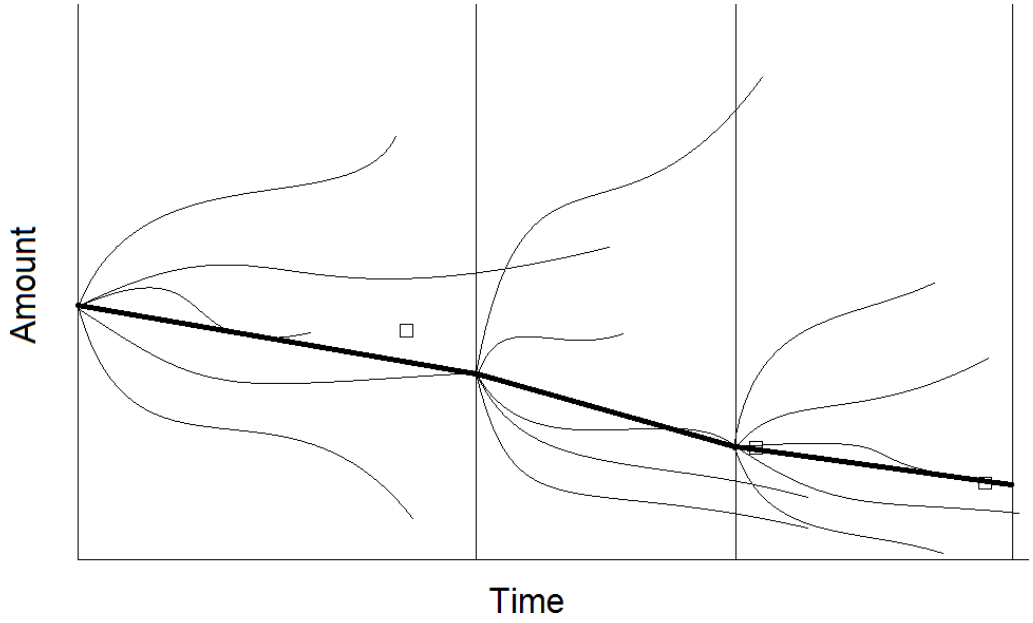



**Figure 4.12**: Illustration showing how the iSSA adaptive algorithm performs simulations. Each time increment can vary in duration based on the number of desired reaction events during each increment. At the end of each increment, the algorithm computes a representative state and uses that as the start state in the next simulation. In this figure, the algorithm is using an adaptive version of the median path method. Time is now a variable in the median state computation, and the squares represent the computed median state that is used in the Euclidean distance calculation to determine the starting state in the next increment.

($\texttt{start} + \frac{\texttt{\#events}}{a_0(\vec{x})}$). This method attempts to capture approximately `#events` reaction events, but many models have very large time scale separations between the fastest and slowest reactions which can lead to this method being dominated by the fast reactions as they have the largest propensity values.

A third way of defining `findLimit` that deals with the timescale separation problem is to divide the number of desired events by the propensity of the slowest reaction ($\texttt{start} + \frac{\texttt{\#events}}{a_{min}(\vec{x})}$). This variant tries to allow each simulation to record `#events` of the slowest reaction event. However, it can run into problems when the slowest reaction's propensity changes a lot during the time increment causing the method to either capture too many or too few events of the slowest reaction.

Finally, the most robust adaptive approach that we have developed is to make the `findLimit` function doubly adaptive. Algorithm 4.2 presents this function that continually updates the remaining time in the time increment based on changes in the slowest reaction's propensity and how much progress has been made in the increment so far.

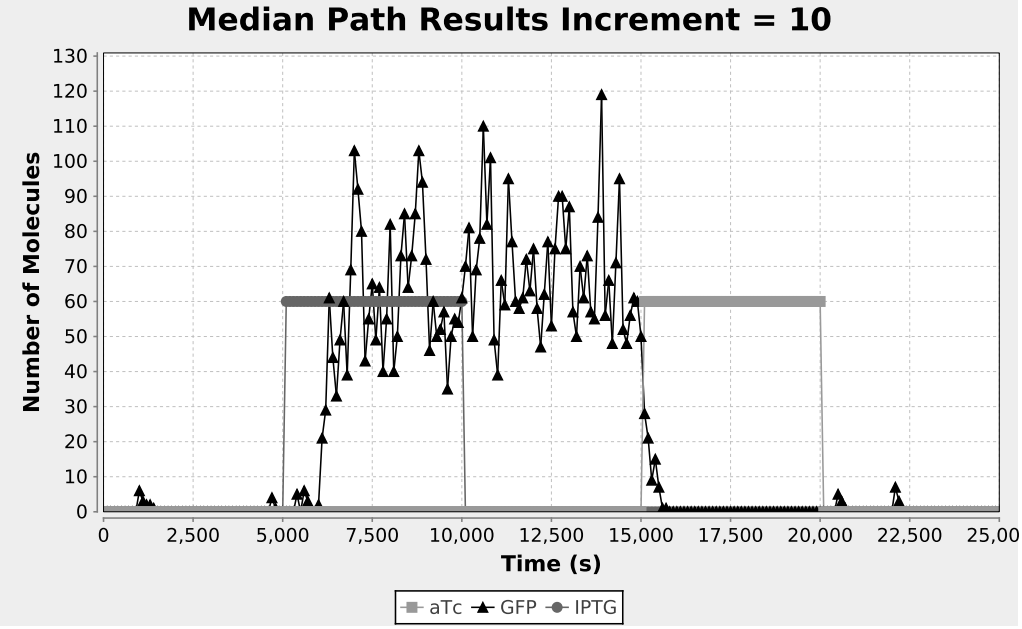
**Algorithm 4.2**: `findLimit`(Start time `start`; Current state $\vec{x}$; Current time $t$)

1 Set `progress` $= (t - \texttt{start}) \times prev\_a_{min}$.
2 Set `remaining` $=$ `#events` $-$ `progress`.
3 Store $prev\_a_{min} = a_{min}(\vec{x})$.
4 **return** `start` $+ \frac{\texttt{remaining}}{a_{min}(\vec{x})}$.

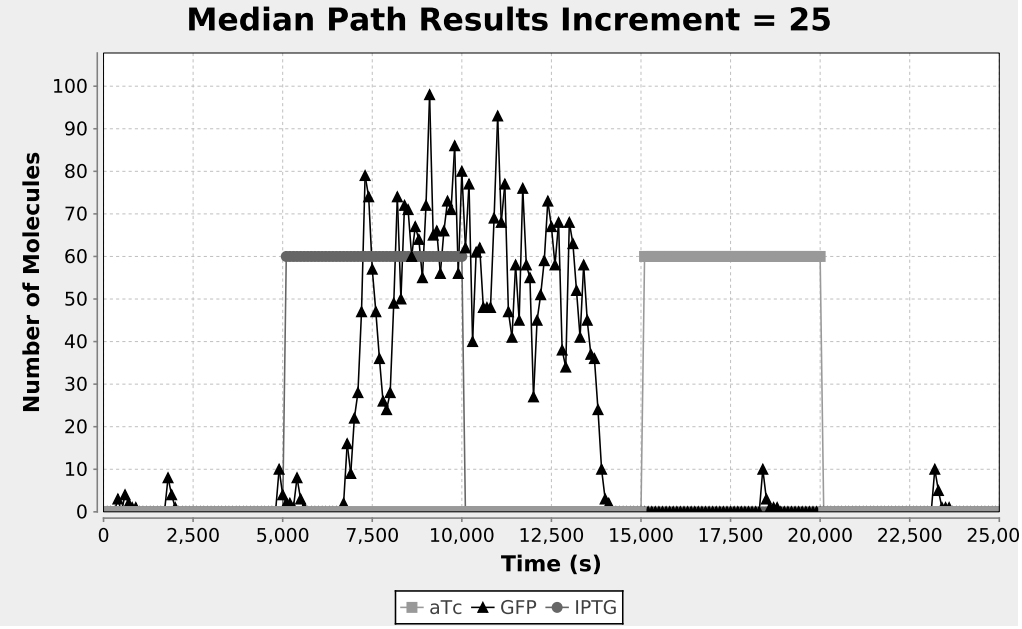
In this algorithm, the amount of progress made towards the current limit is computed by subtracting the current time, $t$, from the starting time, `start`, and by multiplying this value by the smallest propensity from the previous time increment, $prev\_a_{min}$ (line 1). Then, the remaining amount of desired events is computed by finding the difference between the desired number of events, `#events`, and the previously computed progress, `progress` (line 2). Next, the current smallest propensity value is stored into the $prev\_a_{min}$ variable for future computations (line 3), and the new time limit is returned by dividing the remaining progress, `remaining`, by the current smallest propensity value and adding this difference to the starting time (line 4). It should be noted in this algorithm that when $t$ equals `start`, `progress` equals zero and the algorithm returns a limit that tries to capture the desired number of the slowest event. By continually updating the time increment in this manner, this method is able to adjust to drastic changes in reaction propensities and capture approximately `#events` of the slowest reaction event.

An example of comparing a nonadaptive and an adaptive version of the median path method on the genetic toggle switch is presented in Figures 4.13 and 4.14. In these examples, the toggle switch again starts in the OFF state and is supposed to switch to the ON state at time point 5,000 and back to the OFF state at time point 15,000. Figure 4.13(a) and (b) show the result of simulating this circuit using the nonadaptive median path. With different time increment choices, the circuit switches at different times and actually erroneously switches too early when the time increment is 25 seconds. However, when an adaptive version of the median path method is used as in Figure 4.14, the time increments change to try and capture 25 of the slowest reaction events in each increment which leads to accurate results.

The adaptive iSSA method is very robust to switching between simulating with and without the use of reaction-based abstractions. This robustness is apparent in Figure 4.14(a) and (b) where regardless of whether or not reaction-based abstractions are applied to the toggle switch, the results are still accurate. It should also be noted that the number of the slowest reaction events per time increment in each of these simulations

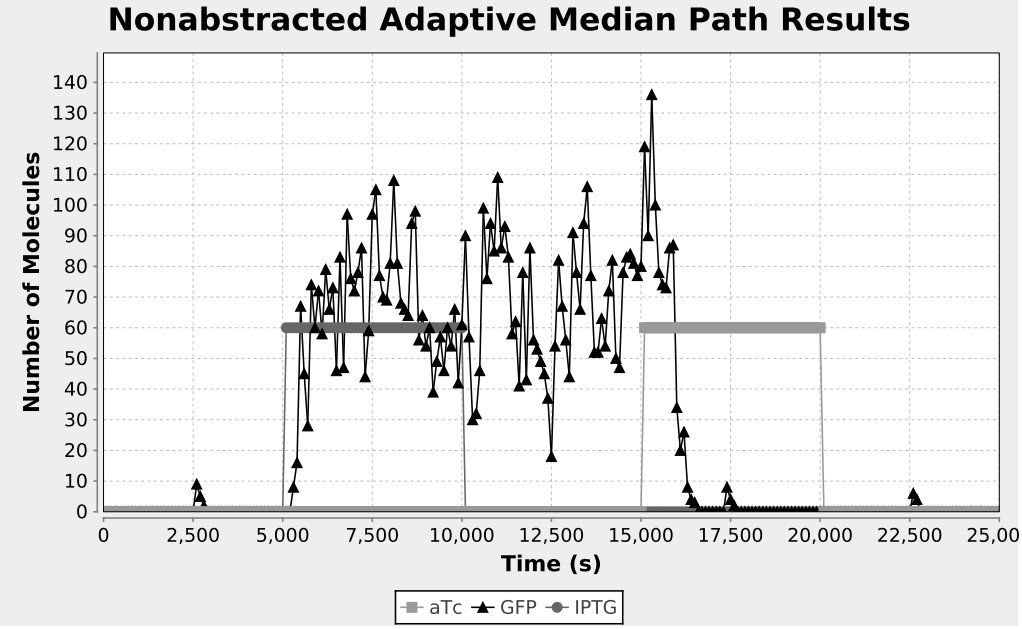



(a)

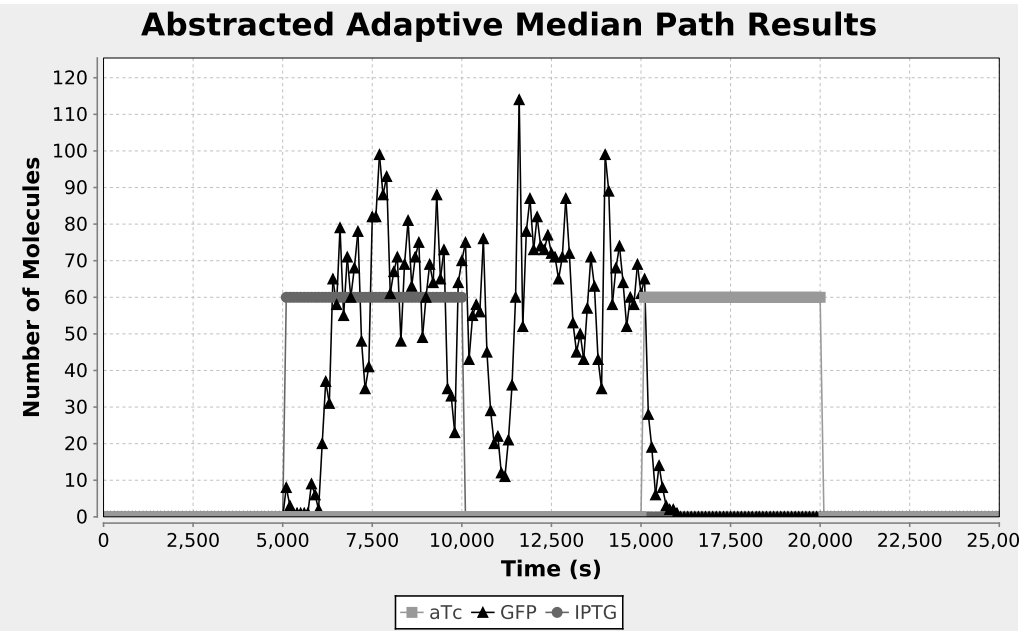



(b)

**Figure 4.13**: Nonadaptive median path simulation results for the genetic toggle switch. Each plot was produced with 100 simulation runs. (a) Results produced from the median path method with a time increment of 10. (b) Results produced from the median path method with a time increment of 25. As seen in these plots, the median path method produces different and possibly erroneous results for a different choice of time increment.




(a)




(b)

**Figure 4.14**: Adaptive median path simulation results for the genetic toggle switch produced with the desired number of events during each time increment equal to 25. This method adapts the time increment in order to try and capture 25 of the slowest reaction events in each increment. (a) Results obtained without using reaction-based abstractions. (b) Results obtained using reaction-based abstractions.

is 25 showing that the adaptive iSSA does not require parameter changes when switching between between using and not using abstractions. This method is also robust when it is used to simulate different models. Indeed, Chapter 6 presents case study results for a variety of models where the adaptive median path iSSA is used to obtain accurate results. Each of the results in Chapter 6 are obtained using 25 slow reaction events per time increment just like the toggle switch results in Figure 4.14; however, this parameter choice does not mean that this method only works with 25 slow reaction events. Values between 10 and 100 usually produce good results, but after testing the adaptive iSSA with many different slow reaction event choices, it has been determined that values closer to 25 (e.g., 20 to 30) typically yield the best results.

## 4.7 iSSA Using Multiple Paths

The previously presented methods are only capable of reporting one "typical" behavior of a genetic regulatory circuit that is most likely to occur. Many circuits, however, exhibit more than one typical behavior. Sometimes observing these behaviors depends on the inputs to the circuit and sometimes observing these behaviors may be due to noise in the circuit.

The iSSA algorithm can be adapted to select and record multiple typical paths of a system. This adaptation is done by modifying the `select` function to use the $k$-means clustering algorithm [60]. Instead of calculating one mean or median state and selecting the run that has the smallest Euclidean distance to this state, the `select` function groups the ending states into an arbitrary number of clusters and calculates a mean for each cluster. The clusters are then iteratively updated by calculating the Euclidean distance between each ending state and each mean and by reassigning each ending state to its "closest" mean. At this point, new means are calculated for each cluster and the process continues until the reassignment does not change any of the clusters. When the clusters stabilize, the `select` function proportionally returns an ending state that has the smallest Euclidean distance from one of the means or medians of each cluster based on how many ending states are in each cluster. This clustering means that starting points for the next increment may be different for some of the runs if the ending points for the previous increment are placed in multiple clusters. Finally, the method returns multiple paths by stitching the starting points of each increment with the selected ending point that had the highest proportion of runs that ended in its cluster. Figure 4.15 illustrates how the

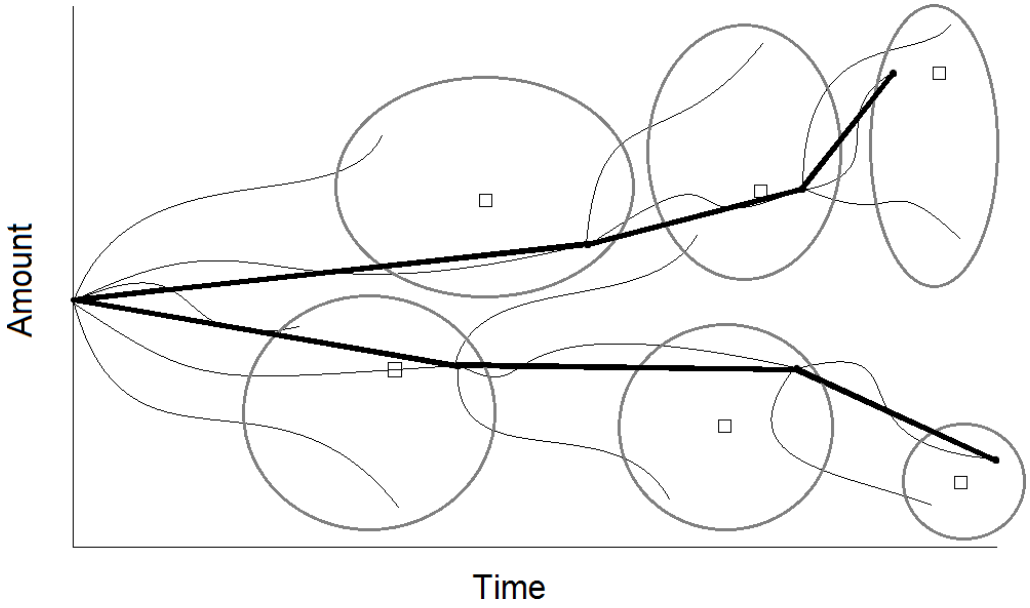
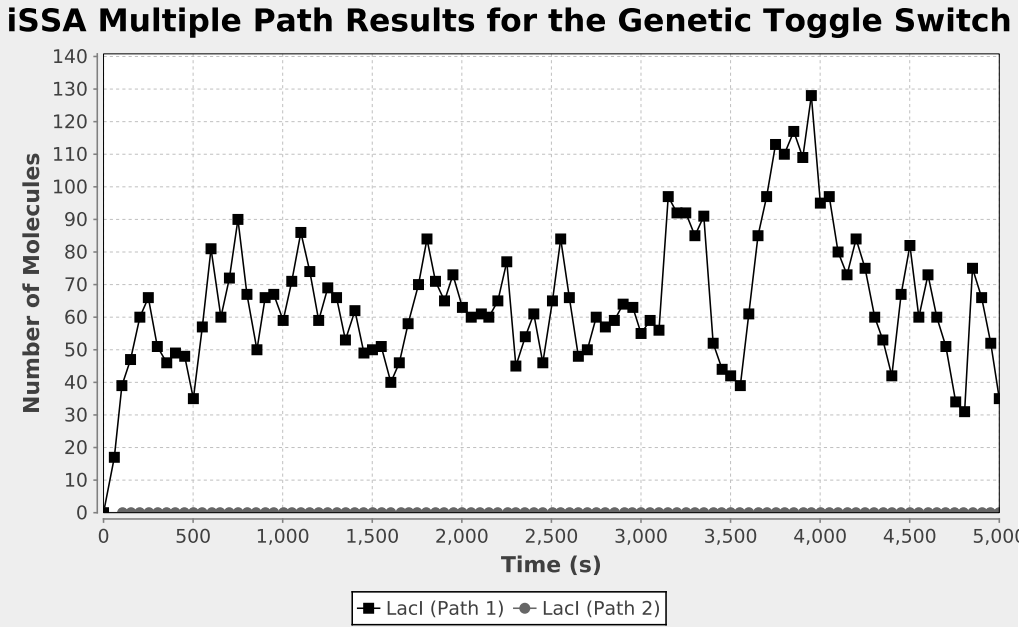



**Figure 4.15**: Illustration showing how the iSSA multiple paths algorithm performs simulations. At the end of each time increment, the ending states are clustered (clusters are shown by grey circles) and a state from each cluster is selected as a starting state for the next time increment. The algorithm then starts a number of runs from each representative state equal to the number of runs that ended in that representative state's cluster. The algorithm is using the adaptive median path method with the multiple paths.

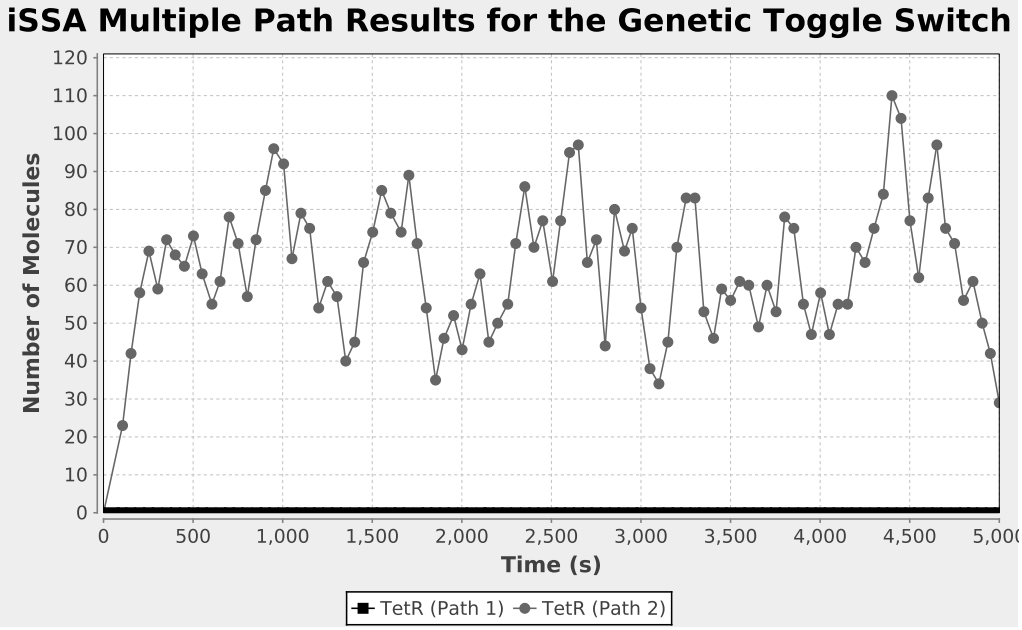
multiple paths method is able to utilize the $k$-means clustering algorithm to select two representative states in each time increment. The grey circles represent the clusters and the thick black lines represent the selected paths.

An example of using the iSSA multiple paths method to simulate the genetic toggle switch is shown in Figure 4.16. In this figure, the toggle switch starts in a state where both inputs are low and both LacI and TetR are low. The circuit has two "typical" behaviors in this case. It can either switch to the ON state or the OFF state. The iSSA multiple paths method captures both of these cases showing that it is capable of tracking multiple paths.

One additional feature of the multiple paths extension is that by using the $k$-means clustering algorithm, the algorithm can determine how many runs end up in each cluster at the end of each time increment. These statistics allow the adaptive iSSA to estimate the probability of a state being in each cluster. Once the simulation is complete, the algorithm can then use these probabilities to report how likely it is for the system to choose a given path over any other path.




(a)




(b)

**Figure 4.16**: iSSA multiple paths simulation results for the genetic toggle switch. (a) Plot showing two paths for the LacI species. (b) Plot showing two paths for the TetR species. In this simulation, the toggle switch starts out in a state with both inputs low as well as LacI and TetR being low. This initial condition leads to the circuit selecting to switch to the ON or the OFF state. As seen in the plot, the multiple paths method is able to capture both of these cases.

# CHAPTER 5

# STOCHASTIC MODEL CHECKING

When designing and analyzing genetic circuits, researchers are often interested in the probability that the system reaches a given state or satisfies a specific condition within a certain amount of time. Usually, this process involves simulating the system to produce some time series data and analyzing this data to discern the state probabilities. However, as the complexity of models of genetic circuits grows, the amount of simulation data needed to determine these probabilities within a certain confidence interval becomes prohibitively expensive to compute. To address this problem, researchers can use stochastic model checking techniques employing Markov chain analysis methods to find the state space of the system directly and compute the probability of being in each state at a given time. However, due to genetic circuits having infinite state spaces, this goal is accomplished by logically abstracting a genetic regulatory circuit into a finite-state *continuous-time Markov chain* (CTMC). This CTMC can then be analyzed using Markov chain analysis to determine the likelihood that the circuit satisfies a given CSL property. This methodology is used to determine the likelihood of certain behaviors in a genetic regulatory circuit. When compared to stochastic simulation-based analysis of the same circuit, the results agree with the reported probabilities but obtain a substantial speedup over these approaches.

Section 5.1 presents a methodology for logically abstracting a genetic regulatory circuit into a CTMC. Section 5.2 describes different probabilistic temporal logics that can be used to specify interesting properties about a genetic regulatory circuit. Additionally, this section presents both transient and steady state stochastic model checking analysis techniques that utilize Markov chain analysis to determine the likelihood that the circuit satisfies a particular property.

## 5.1 Logical Abstraction

Even after applying reaction-based abstractions to chemical reaction networks, the state space of the system can be extremely large. This problem has led to the application

of logical abstractions to these networks [80]. One such abstraction is through the use of qualitative models [78]. These models, however, typically must be produced by hand and do not lend themselves to quantitative analysis. To address the problems with qualitative models, researchers have used boolean networks to define the system [49, 50, 51, 79]. In these networks, each species is either expressed (on) or not expressed (off). Updates to these networks are typically deterministic and occur synchronously meaning that all variables update at the same time. Although this abstraction can be very efficient, especially for large systems, it often fails to capture the stochastic nature of genetic regulatory circuits.

In order to directly analyze a genetic regulatory circuit, it must first be converted into a CTMC. Directly translating the circuit into a CTMC, however, yields a Markov chain with an infinite state space since each species count can take on any value from zero to infinity. Since analytic techniques cannot deal with an infinite state space, the conversion process must logically abstract the circuit into a finite-state CTMC. The simplest way to convert to a finite state space is to select a large upper-bound for each species. One problem with this method is that, typically, a very large value is selected for each species as the upper-bound leading to an incalculable state space. Therefore, in order to avoid the state explosion problem, methods to reduce the state space so that it is small enough to be analyzed in a reasonable amount of time but is also able to give a good approximation of the original circuit must be developed.

One method of reducing the state space of the genetic circuit is to add a notion of logical levels for each species [56, 53]. These levels allow for the specification of more than just the ON and the OFF state of a species while still encoding the infinite state space into a finite state space. However, the reduced state space means that the state transition rates must be approximated in the resulting CTMC before analyzing the system with efficient stochastic analyses such as stochastic model checking [58, 85].

### 5.1.1 Threshold Selection

The first problem that must be considered when logically abstracting a genetic regulatory circuit is how to partition the state space so that it yields a good approximation of the entire state space. To solve this problem, threshold selection can be used to determine which values for each species give the best approximation. However, methods for performing threshold selection usually have to trade-off between quicker analysis times but not as accurate approximations and slower analysis times but good approximations.

One method of accomplishing threshold selection is to perform a few simulations of the circuit and then to analyze the resulting traces to figure out the range of values for each species. Then, a user can determine a number of thresholds to be selected uniformly from these ranges. If more thresholds are used, then the resulting state space grows in size leading to longer analysis times but a more accurate approximation. Similarly, fewer thresholds result in a less accurate approximation but lead to shorter analysis times. By selecting thresholds uniformly from a range, this method may end up not using enough thresholds around values of a species that cause some of the calculated transition rates to change dramatically and may use too many thresholds for values of a species that do not affect these rates significantly.

Instead of selecting thresholds uniformly, another method is to attempt to select thresholds for each species at points where the rates of the reactions in the systems change drastically. This method would likely involve graphing the rates of each reaction and searching for the inflection points of the resulting curves. This approach could then automatically select more thresholds for the species that affect these rates at the values around the inflection points and could select fewer thresholds for these species at values that are further away from the inflection points.

### 5.1.2 Translation of a Genetic Circuit into a CTMC

The critical step in preparing a genetic circuit for stochastic model checking after thresholds have been selected is the conversion of the circuit into a CTMC. This conversion process begins by finding the state space of the genetic circuit. First, a sparse matrix where each entry, $p_{i,j}$, represents the rate of moving from state $i$ to state $j$ is constructed. Next, a state is created with an encoding of the initial values of the species in the model. The conversion then performs a depth first search by changing one species encoding at a time to a higher or lower encoding if they exist in the level set $L$. Each valid change found this way is pushed onto a stack. The algorithm then pops an encoding off the stack and checks to see if a transition rate for moving from the current state to the new state exists in the matrix. If it does, the algorithm stops exploring this path and pops the next change off the stack to explore further. Otherwise, the transition rate is calculated using Equations 5.41 and 5.42 and is added to the matrix. Once the last encoding is popped off the stack, the conversion from a circuit to a CTMC is complete.

Figure 5.1 shows a graphical representation of the state space for the genetic toggle switch with thresholds selected at 0, 30, and 60 for both LacI and TetR. These threshold

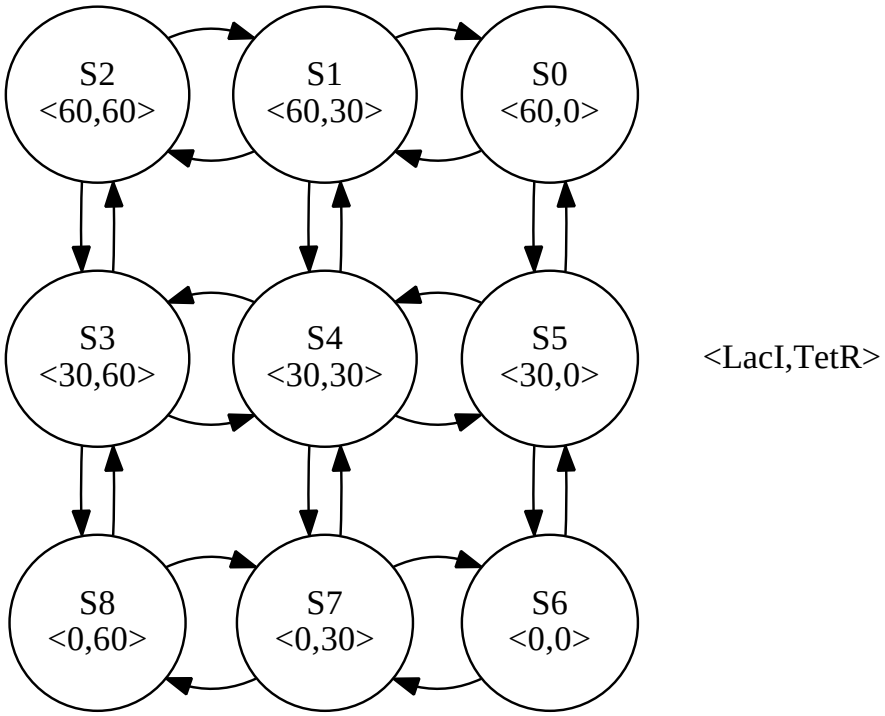



**Figure 5.1**: The state graph of the genetic toggle switch with thresholds selected at 0, 30, and 60 for both LacI and TetR. The initial state is state S0.

selections yield nine states labeled S0 through S8. State S0, the state where LacI is at its highest level of 60 and TetR is at its lowest level of 0, is the initial state.

Once the entire state space of the genetic circuit is found, the transition rates between the states are computed and inserted into the sparse matrix. These rates are determined using the reaction rates from the genetic circuit after reaction-based abstractions from Section 3.5 have been applied to the model. These rates are computed using the formulas in Equations 5.41 and 5.42:

$$\text{production}(s, l, l') = \frac{\sum_{p \in Pro(s)} n_p \cdot \text{rate}(p)}{(l'[s] - l[s])} \tag{5.41}$$

$$\text{degradation}(s, l, l') = \frac{k_d l[s]}{(l[s] - l'[s])} \tag{5.42}$$

In these equations, $s$ is the species value that is changed by the transition rate equation, $l$ is the current state array, $l'$ is the next state array, and Pro($s$) returns the set of promoters that initiate transcription of genes that lead to the production of species $s$. When the state transition increases the level of species $s$ from $l[s]$ to $l'[s]$, then the production formula is used, and when the state transition decreases the level of $s$ from $l[s]$ to $l'[s]$, then

the degradation formula is used. The rate for production is computed by determining the rate of production for each promoter $p$ which produces species $s$ using the rate($p$) function defined in Equation 3.40. To obtain a numerical value from this equation, the species variables in the rate($p$) equation are replaced by indexing into the $l$ state array to get their current values. This rate is then multiplied by $n_p$, the number of proteins produced per transcript, to convert this rate into the rate for a single protein production. The rate of degradation is computed as $k_d l[s]$ where $k_d$ is the degradation rate parameter and $l[s]$ is the starting level for species $s$ before degradation. In both cases, these rates must be normalized by the difference in the level before and after the state change. This normalization is necessary because the rates are for the production or degradation of a single molecule of $s$ while the state change only occurs after $l'[s] - l[s]$ molecules are produced or $l[s] - l'[s]$ molecules are degraded.

The conversion process coupled with the corresponding rate functions have been carefully constructed such that the CTMC generated gives a reasonable approximation of the behavior of the genetic circuit. This translation procedure allows the user to efficiently trade-off between accuracy and analysis time. Namely, the more thresholds used in the level set, the more accurate the model becomes. Of course, using more thresholds also increases analysis time, so the user should select the minimal number of thresholds necessary to perform the desired analysis. Figure 5.2 presents the CTMC that is obtained after adding transition rates to the state graph in Figure 5.1.

## 5.2 Stochastic Model Checking

Due to the inherently noisy nature of genetic circuits, any deterministic assertion that is checked would most likely fail to be true. Instead, for these systems, the more interesting question is the probability that a property is true. Determining the likelihood of properties can be accomplished using a technique known as stochastic model checking.

There are two types of stochastic model checking used to compute the likelihood that a property is true: statistical and numerical based techniques [58, 85]. Statistical techniques involve simulating a system a large number of times and terminating whenever a property is shown to be true or false. When all of the simulations are complete, statistics are calculated on how many simulations satisfied the property in the time allotted versus the number of simulations that failed to do so. One downside of using statistical techniques is that the more rare an event is, the more simulations that need to be run in order to observe it, and performing these simulations may cause the time that it takes to compute

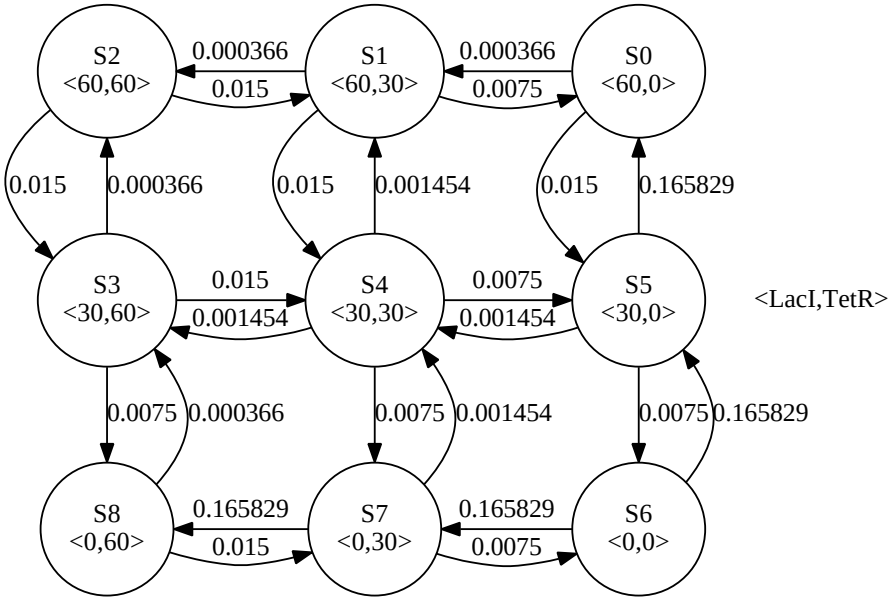



**Figure 5.2**: The state graph annotated with transition rates resulting in a continuous-time Markov chain.

a likelihood to become prohibitively expensive. Numerical methods, on the other hand, attempt to determine these likelihoods in a more direct method. They usually attempt to find the state space of the model and then employ methods such as Markov chain analysis to compute the probability of reaching a state where a given property is satisfied. These methods are often more efficient than statistical techniques; however, they require that the state space be computable. Both statistical and numerical methods have been utilized by many tools such as the probabilistic model checker PRISM [45].

### 5.2.1 Probabilistic Temporal Logics

There are several options for logics that can be used to check properties of a genetic regulatory circuit. Among these are *probabilistic linear temporal logic* (PLTL), *probabilistic computation tree logic* (PCTL), and *continuous stochastic logic* (CSL). PLTL is an extension of *linear temporal logic* (LTL) that is interpreted over linear time points [81, 68]. This logic includes operators such as "next," "until," "sometime," and "always" and can express properties that determine that a proposition becomes true sometime in the future with at least a certain probability. This property language is particularly useful when performing statistical model checking as each simulation trace is a linear execution of the system. When analyzing a system with numerical model checking on the other hand, this

logic does not perform as well if the system contains many different possible paths or executions.

A logic more suited for dealing with many different possible executions of the system is PCTL [40, 41]. This logic is an extension of *computation tree logic* (CTL) [17] and is well suited for computation on *discrete-time Markov chains* (DTMCs). It views each transition in a path as a time step of the system and is able to express properties that specify that a certain event happens with some probability in a certain number of steps of the system. This logic accomplishes this goal by replacing path quantifiers from CTL with probability quantifiers. These replacements allow for PCTL to express more general quantification properties because it is capable of expressing whether a property is true in a certain percentage of paths in addition to whether a property is true in all paths. Despite these advantages, PCTL is incapable of dealing with continuous-time.

CSL, a continuous-time variant of PCTL, has all the capabilities of PCTL and is able to express properties of CTMCs [8, 58]. The only difference between it and PCTL syntactically is that there is now next state operator. Also, since this logic works in continuous-time, time bounds of properties do not need to be defined as discrete steps but can be given as inequalities or ranges over time. As genetic regulatory circuits may contain many different possible executions and are readily converted into CTMCs, CSL is the best-suited logic for stochastic model checking of genetic circuits.

The grammar for CSL properties used by the stochastic model checking algorithm presented in this chapter is given as follows:

$$
\begin{aligned}
\mathit{Prop} &::= \mathtt{U}(\mathrm{T}, \Psi, \Psi) \mid \mathtt{St}(\Psi) \\
\Psi &::= \mathtt{true} \mid \Psi \wedge \Psi \mid \neg\Psi \mid \phi \geq \phi \mid \phi > \phi \mid \phi = \phi \\
\phi &::= v_i \mid c_i \mid \phi + \phi \mid \phi - \phi \mid \phi * \phi \mid \phi / \phi \mid \mathit{Prop} \\
\mathrm{T} &::= \mathtt{true} \mid \mathrm{T} \wedge \mathrm{T} \mid \neg\mathrm{T} \mid t \geq c_i \mid t > c_i \mid t = c_i
\end{aligned}
$$

where $v_i$ is a variable, $c_i$ is a constant, and $t$ stands for time in the system. $\Psi$ represents a state formula that must be true in a given state and can be made up of other state formulae combined together with logical connectives or can be made up of comparing a numerical expression, $\phi$, with another numerical expression. The formula $\mathtt{U}(T, \Psi_1, \Psi_2)$ represents the probability that an execution of the system satisfies the until formula $\Psi_1 \, \mathtt{U}^{\mathrm{T}} \, \Psi_2$ which means that $\Psi_1$ must remain true until $\Psi_2$ becomes true within the time frame that the time bound expression, T, evaluates to true. The formula $\mathtt{St}(\Psi)$ represents the probability

that once the system reaches its steady state, it is in a state where $\Psi$ is satisfied. It should be noted that *Prop* is a symbol in $\phi$'s grammar which allows for CSL properties to be nested within other CSL properties. As a shorthand, $\Psi$ and T can also contain `false`, $\vee$, $<$, and $\leq$ which are easily derived. The formulas are also allowed to contain the eventually operator, `F`, and the globally true operator, `G`, defined as follows:

$$\begin{aligned} \texttt{F}(\mathrm{T}, \Psi) &\equiv \texttt{U}(\mathrm{T}, \texttt{true}, \Psi) \\ \texttt{G}(\mathrm{T}, \Psi) &\equiv 1 - \texttt{F}(\mathrm{T}, \neg\Psi) \end{aligned}$$

The eventually operator is essentially used as a shorthand for describing an until property where the left-hand side of the formula is true. For example, the eventually formula $\texttt{F}(\mathrm{T}, \Psi)$ would simply require that $\Psi$ becomes true before T evaluates to false. The globally true formula $\texttt{G}(\mathrm{T}, \Psi)$ requires that $\Psi$ remains true during the time that T evaluates to true. This formula builds off of the eventually operator by requiring that $\neg\Psi$ does not eventually become true while T evaluates to true and returns one minus the resulting probability.

After the CTMC is computed and a CSL property has been specified, the stochastic model checker checks each state to determine whether it either satisfies or fails the CSL property. When checking transient CSL properties, the model checker determines if the encoding either does not satisfy the left hand side of an until formula or satisfies the right hand side of the formula. If either of these checks hold, the state is marked as absorbing and all transitions out of the state are pruned from the CTMC. An example of performing this pruning on the CTMC in Figure 5.2 is shown in Figure 5.3. Here, the user is interested in the probability of LacI going to 0 within 100 seconds which is represented by the following CSL property:

$$\texttt{F}(t \leq 100, \mathrm{LacI} = 0)$$

States $S6$, $S7$, and $S8$ become absorbing in the pruning process as they satisfy the right hand side of the transient property's formula.

### 5.2.2 Algorithm

The final step of stochastic model checking is to compute the probability of the CSL property being true for the CTMC. Algorithm 5.1 determines the probability within an error bound, $\epsilon$, of a given CSL property, $\Phi$, on a genetic circuit model, $M$. Additionally, this algorithm requires a set of levels, $L$, that includes an ordered list of threshold levels, $L_s$, for each species $s \in S$ in the model. Each level, $l_{s,i}$ represents a critical threshold in

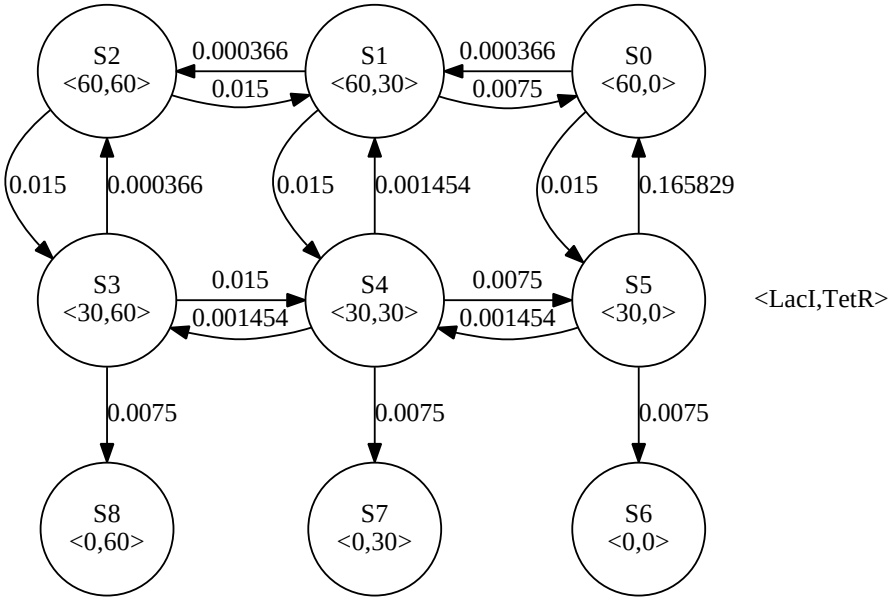



**Figure 5.3**: The CTMC after it is pruned for the CSL property, $\mathtt{F}(t \leq 100, \mathrm{LacI} = 0)$. States $S6$, $S7$, and $S8$ are absorbing since they satisfy the CSL property.

**Algorithm 5.1**: `SMC`(Model $M$; Levels $L$; CSL property $\Phi$; Error-bound $\epsilon$)

```
1  Set C = computeCTMC(M, L, Φ)
2  foreach Nested property Φ' ∈ Φ do
3      foreach State s ∈ C do
4          Set M' = M.setInitialState(s)
5          Set p = SMC(M', L, Φ', ε)
6          s.addVariable(Φ', p)
7  Set t = determineTimeLimit(Φ)
8  if t ≠ ∞ then
9      return transientAnalysis(C, t, Φ, ε)
10 else
11     return steadyStateAnalysis(C, Φ, ε)
```

the amount of the species $s$. It is assumed that $l_{s,0}$ is always 0, and $l_{s,i-1} < l_{s,i}$ for all $i > 0$.

The first step of the algorithm converts the model into a CTMC, $C$, using the logical abstraction discussed in Section 5.1 and the property pruning described in Section 5.2.1 (line 1). Next, the algorithm parses the CSL property and walks its expression tree looking for any nodes that represent nested properties (line 2). If any nested properties are found, the algorithm loops over each state in $C$ (line 3) and alters $M$ by setting its initial state

to the Markov chain state, $s$ (line 4). The algorithm then recursively calls itself for each altered model, $M'$, using the nested property, $\Phi'$, as the new CSL property to be checked (line 5). The probability of each recursive call, $p$, is stored as a variable of each $s$ where it can be referenced when determining if the state satisfies $\Phi$ (line 6). Once all of the nested properties are dealt with, the algorithm determines the amount of time necessary for the analysis, $t$, which is essentially the maximum value that time can take while still allowing for the time bound expression to evaluate to true in the transient property or infinity, $\infty$, in the case of a steady state property (line 7). Finally, the algorithm checks whether transient or steady state analysis should be performed and calls the appropriate analysis method (lines 8-11). Each of these analysis methods are presented in Sections 5.2.3 and 5.2.4, respectively.

### 5.2.3 Transient Analysis

In order to perform transient analysis to determine the probability of a property being true at a specified time, the Markov chain must be transformed using a *transient Markov chain analysis* method known as *uniformization* [76]. Algorithm 5.2 utilizes uniformization to determine the probability within an error bound, $\epsilon$, of a given transient CSL property, $\Phi$, on a CTMC, $C$. The first step of the algorithm is to derive the *infinitesimal generator matrix* from the CTMC by assigning the negation of the sum of the transition rates out of each state to the diagonal entries of the matrix (line 1). After that, the absolute value of the largest diagonal entry is selected as $\Gamma$ (line 2) and the discretized stochastic

**Algorithm 5.2**: `transientAnalysis`(CTMC $C$; Time limit $t$; Property $\Phi$; Error-bound $\epsilon$)

1 Find the infinitesimal generator matrix, $Q^X$, of $C$
2 Compute $\Gamma = \max_i |q_{ii}^X|$ where $q_{ii}^X$ is a diagonal entry of $Q^X$
3 Find stochastic transition probability matrix, $P = I + \frac{1}{\Gamma} Q^X$
4 Set $K = 0$, $\xi = 1$, $\sigma = 1$, and $\eta = \frac{1-\epsilon}{e^{-\Gamma t}}$
5 **while** $\sigma < \eta$ **do**
6  Compute $K = K + 1$, $\xi = \xi \times \frac{\Gamma t}{K}$, and $\sigma = \sigma + \xi$
7 Set $\pi(0)$ so initial state has probability 1 and all others 0
8 Set $\pi = \pi(0)$ and $y = \pi(0)$
9 **for** $k = 1$ to $K$ **do**
10  Compute $y = yP \times \frac{\Gamma t}{k}$ and $\pi = \pi + y$
11 Compute $\pi(t) = e^{-\Gamma t}\pi$
12 **return** $\sum$ of all states in $\pi(t)$ that satisfy $\Phi$

probability matrix, $P$, is computed (line 3). The remainder of the algorithm analyzes the CTMC using uniformization. The algorithm first needs to know the number of terms in its summation, $K$, which it determines iteratively (lines 4-6). The algorithm then initializes the initial state's probability to 1 and all other states' probabilities to 0 (line 7) and proceeds by iteratively performing vector-matrix multiplications and vector additions to simulate the evolution of the system's likelihood of being in any state (lines 8-10). After these computations are complete, the uniformization algorithm normalizes the final probabilities in $\pi$ (line 11). Finally, after applying this method, each state is now annotated with the probability of being in that state at the specified time. At this point, the final probability for the CSL property is determined by summing over all states in which the right-hand side of the until formula is satisfied (line 12).

An example of using this method to analyze the CTMC presented in Figure 5.3 is presented in Figure 5.4. This figure shows the CTMC annotated with probabilities after applying transient Markov chain analysis to it. The sum of the probability of reaching these states represents the probability of satisfying the property, which is about 5.9 percent.

### 5.2.4 Steady State Analysis

Steady state analysis of the genetic circuit can be performed directly on the CTMC without the need for any transformations by using the *power iteration method* [76]. Algorithm 5.3 determines the probability within an error bound, $\epsilon$, of a given steady state CSL

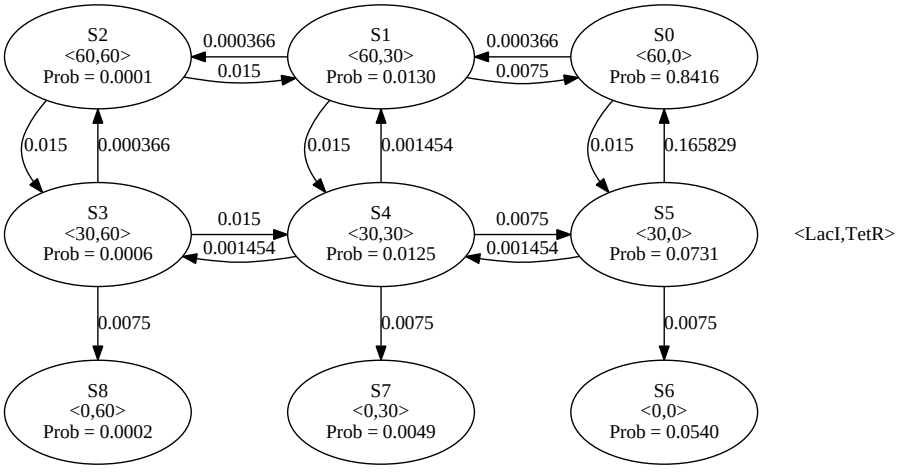



**Figure 5.4**: The CTMC annotated with probabilities after applying transient Markov chain analysis with the CSL property, $\mathtt{F}(t \leq 100, \mathrm{LacI} = 0)$.

**Algorithm 5.3**: `steadyStateAnalysis`(CTMC $C$; Property $\Phi$; Error-bound $\epsilon$)

1 Find the embedded Markov chain, $S$, of $C$
2 Set $p = \texttt{findPeriod}(S)$
3 Set $i = 0$ and diff $= \infty$
4 Set $\pi(i)$ so initial state has probability 1 and all others 0
5 **while** diff $> \epsilon$ **do**
6  Compute $\pi = \pi(i)S$
7  Set $i = (i+1)\%p$
8  Set diff $= \max[\pi - \pi(i)]$
9  Set $\pi(i) = \pi$
10 Compute $\pi = \frac{\sum_{i=0}^{p-1} \pi(i)}{p}$
11 **foreach** $\pi_i \in \pi$ **do**
12  Compute $\pi_i = \frac{\pi_i}{\sum q_{ij}}$
13 Compute $\pi = \frac{\pi}{\sum \pi_i}$
14 **return** $\sum$ of all states in $\pi$ that satisfy $\Phi$

property, $\Phi$, on a CTMC, $C$. The first step of the algorithm is to derive the *embedded Markov chain* (EMC) from the CTMC by dividing each transition rate in the chain by the sum of all transition rates out of a given state (line 1). The next step is to find the period of the EMC (line 2). The period is necessary because if the EMC is periodic, then instead of converging to an invariant distribution, the probability values of each state in the chain oscillates as iterations are performed. Next, the iteration counter and difference checking variables are set to zero and infinity, respectively (line 3). The algorithm then initializes the initial state's probability to 1 and all other states' probabilities to 0 (line 4). At this point, the algorithm iterates by continually multiplying the previous distribution, $\pi(i)$, by the EMC to compute the new distribution, $\pi$, and checks to see if the new distribution has changed over the old distribution by more than the error-bound amount, $\epsilon$ (lines 5-9). This process, however, must take into account the period of the EMC by keeping track of $p$ different distributions. The distribution that the new distribution is compared against is determined by finding the modulus of the increment counter and the period in the increment step (line 7). The difference between this distribution and the new distribution is then calculated (line 8), and the new distribution replaces the old distribution (line 9). Next, the alternating distributions are summed together and normalized by the period (line 10). After this step, the distribution must be normalized to take into account the transition rates in the original CTMC. In order to accomplish this normalization, each of

the elements of the final distribution are divided by the sum of all transition rates out of its corresponding state (lines 11-12), and the resulting distribution is divided by the sum of all the probabilities in the distribution so that they sum to one (line 13). Lastly, the final probability for the CSL property is determined by summing over all states in which the formula is satisfied (line 14).

An example of using this method to analyze the CTMC presented in Figure 5.2 is presented in Figure 5.5. This figure shows the CTMC annotated with probabilities after applying steady state Markov chain analysis to it. In this example, the user may be interested in the probability of the system being in a state where LacI is 0 in the long run which is represented by the following CSL property:

$$\texttt{St}(\text{LacI} = 0)$$

Summing over the states that satisfy this property, states $S6$, $S7$, and $S8$, results in a probability of about 48.2 percent.

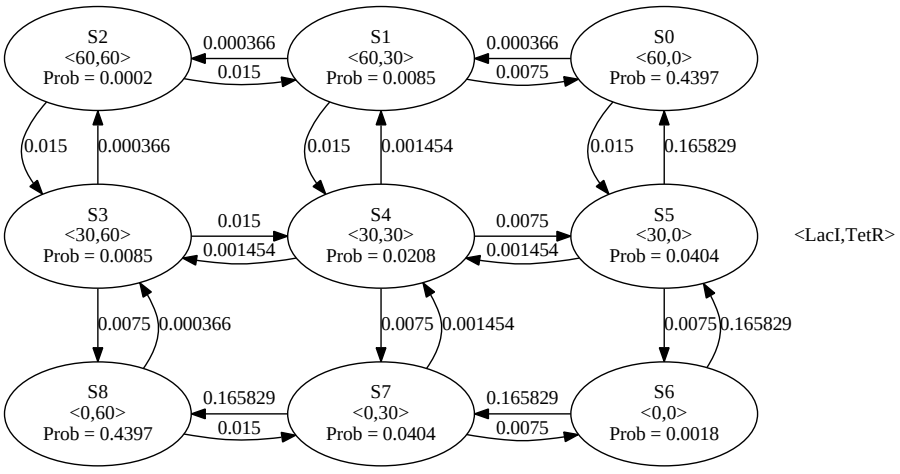



**Figure 5.5**: The CTMC annotated with probabilities after applying steady state Markov chain analysis.

# CHAPTER 6

# CASE STUDIES

The iSSA and stochastic model checking techniques are useful methods on their own for determining the validity of models of genetic circuits; however, when these methods are used in concert, they can be used to effectively perform design space exploration of genetic circuits. This chapter presents several examples where using the iSSA and stochastic model checking allows for better design choices when constructing genetic circuits. Section 6.1 describes and analyzes several genetic oscillator circuits using the design space exploration methodology. Section 6.2 shows how this methodology can be applied to genetic circuits that represent state holding gates to help design more robust circuits with switches.

## 6.1 Genetic Oscillators

Genetic oscillators are important in synthetic and systems biology as they are essentially used as clocks to ensure that important events happen periodically in genetic circuits. For example, the cell cycle is driven by a genetic oscillator that ensures that the cell grows and divides as it is supposed to. Since cells can be subject to a variety of different environments, genetic oscillators must be robust to stochastic events and noise. This section analyzes the robustness of several genetic oscillators including a circadian rhythm model, the repressilator, and the dual-feedback genetic oscillator.

### 6.1.1 Circadian Rhythm

The circadian rhythm model described in [82] is shown in Figure 6.1. In this model, species $A$ activates the promoters $D_A$ and $D_R$ leading to a spike in the amounts of both $A$ and $R$ protein. When these populations start growing, $A$ and $R$ molecules start binding to from $C$ complexes. This binding leads to $A$ being unable to activate the promoters and all the species degrade away. The system stays in a state with low counts of all the species until some spurious production of $A$ allows for the activation $D_A$ and $D_R$ again.

An example of performing ODE and SSA simulations on the circadian rhythm model is shown in Figure 6.2. Simulation results for the circadian rhythm model are expected

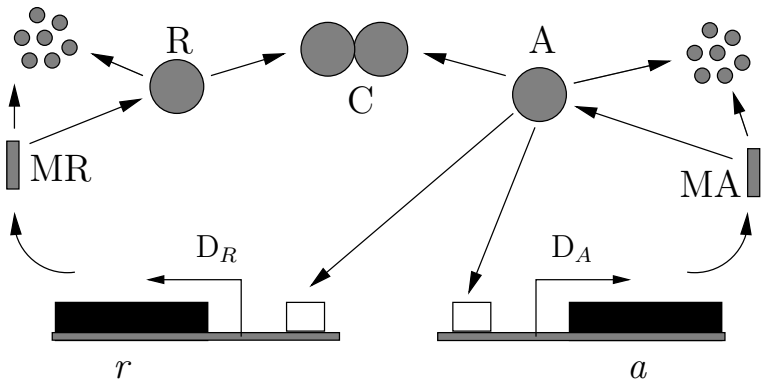



**Figure 6.1**: Genetic circuit model for circadian rhythm. In this model, $A$ activates both $A$ and $R$ through the promoters $D_A$ and $D_R$, respectively. $A$ and $R$ also bind together to form the complex $C$. This model is expected to produce oscillatory behavior of the $A$, $R$, and $C$ species.

to result in oscillatory behavior of the $A$, $R$, and $C$ species. This figure shows that an individual SSA simulation run captures the expected behavior. However, this figure also shows an ODE simulation of the circadian rhythm model where one pulse is observed and then there are no subsequent pulses due to the simulation missing stochastic events. Similarly, the plot of the average of 100 SSA simulations is only able to capture one definite pulse. The rest of this plot shows how since the subsequent pulses can come at different times, the average of 100 runs suffers from the smoothing effect and fails to capture any other pulses.

To further show the advantages and disadvantages of the iSSA MPDE method, Figure 6.3(a) shows simulation results for species A using the MPDE method with 100 stochastic runs. These results retain the pulses, and correctly reveal the circuit's typical behavior as opposed to the ODE and average SSA results presented in Figure 6.2. Figure 6.3(b), on the other hand, shows a simulation of the circadian rhythm model using the MPDE method with abstraction. In this plot, the MPDE method erroneously predicts that the circuit pulses with increasingly larger amplitudes than the previous pulses and with irregular frequencies. These plots serve as more evidence that MPDE tends to be less attractive for use with abstracted simulation models.

The adaptive iSSA using the median path method, the most robust iSSA method, correctly predicts the "typical" behavior of the circadian rhythm model. Figure 6.4 presents results of applying this method to an abstracted version of the model. This plot shows that the adaptive median path iSSA produces a plot that looks similar to

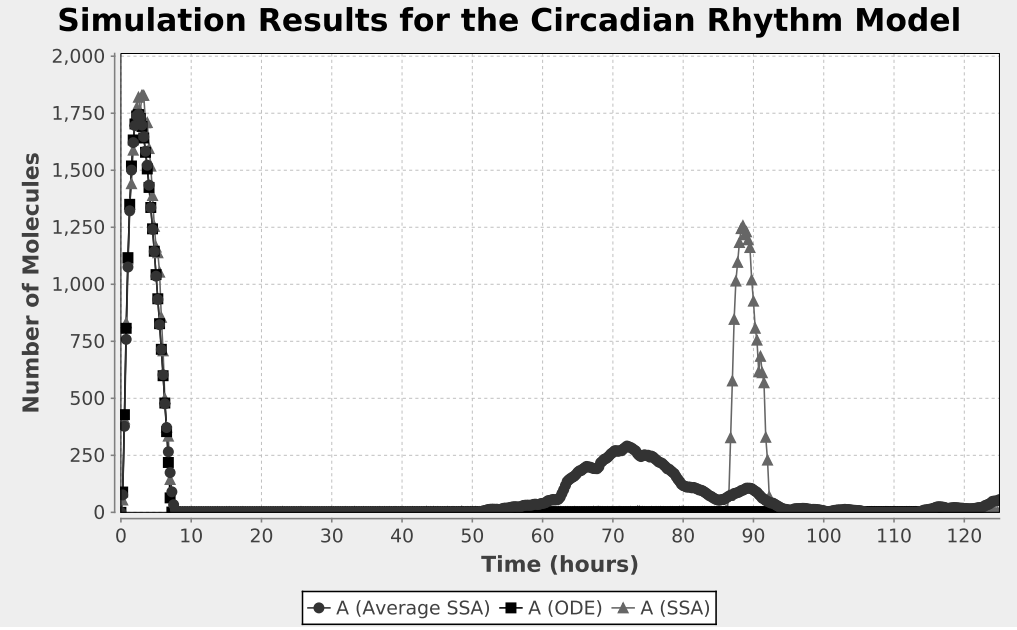
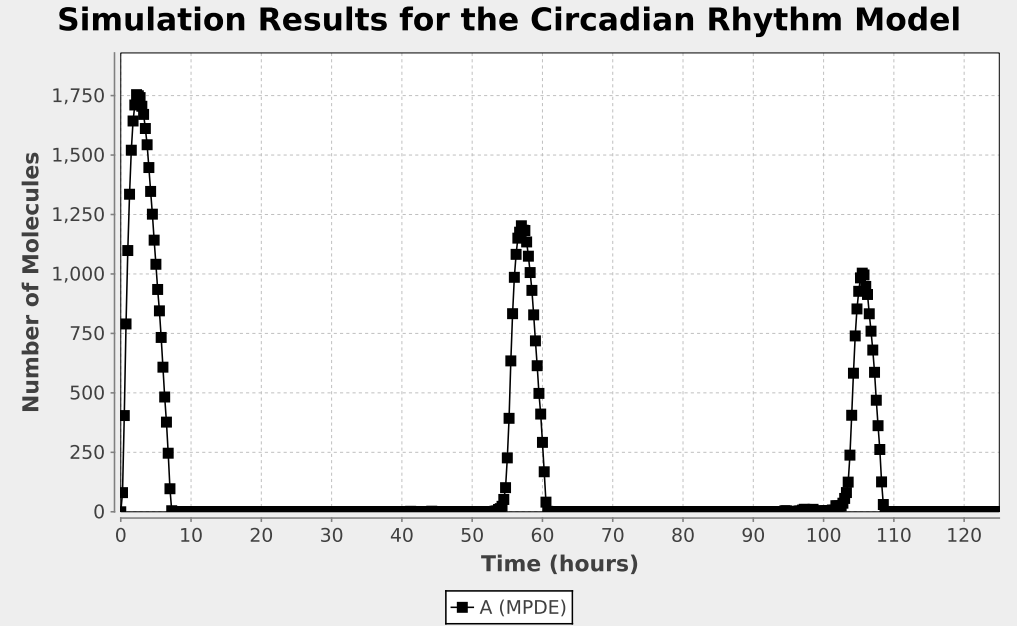
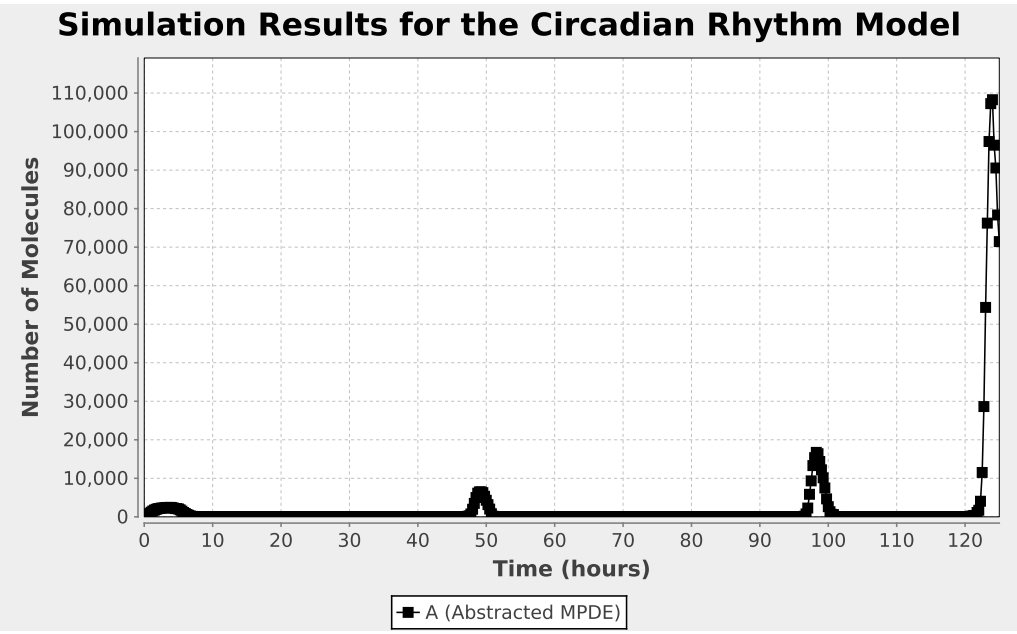



**Figure 6.2**: Simulation results for the circadian rhythm model. This plot shows an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs.




(a)




(b)

**Figure 6.3**: MPDE simulation results for the circadian rhythm model. These results use a time increment of 0.25 hours with 100 runs for each time increment. (a) These results are obtained by simulating the model without abstraction. (b) These results are obtained by simulating the model with abstraction. As seen in this plot, using abstraction with MPDE causes the simulation to produce erroneous results.

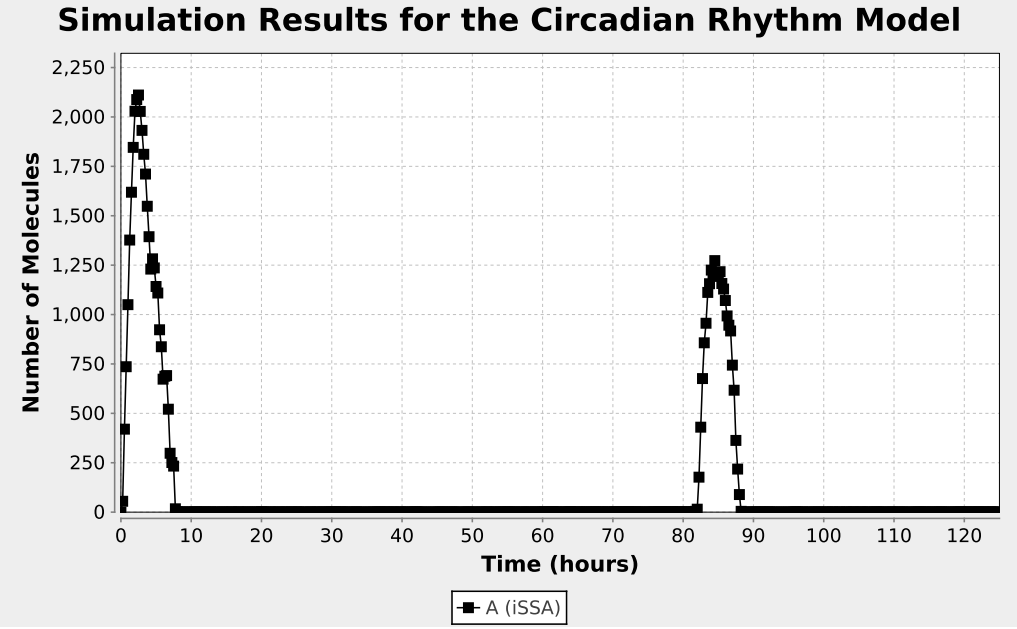



**Figure 6.4**: Adaptive median path iSSA results for the circadian rhythm model.

the individual SSA run from Figure 6.2. As far as run-times are concerned, it takes the ODE method about 3 seconds, the SSA about 1 minute and 25 seconds, the iSSA with MPDE and no abstraction about 3 minutes and 10 seconds, and the adaptive median path iSSA about 1 minute and 30 seconds to obtain results. These run-times show that the adaptive median path iSSA is comparable in efficiency to the SSA which makes sense as it is built around the SSA. These results also demonstrate that this method is more efficient than the iSSA with MPDE as it is able to obtain results in half the time. Although this comparison does not yield a significant difference in run-time for this example, run-times for other examples can be orders of magnitude greater which shows how beneficial an iSSA method that allows abstraction can be for simulation.

### 6.1.2 Repressilator

The repressilator model shown in Figure 6.5 is comprised of the species CI, LacI, and TetR that are connected in a loop [22]. This circuit is a ring oscillator where each species represses the next one forming the loop. When one of the species (e.g., LacI) is produced, it represses the next species in the chain (e.g., TetR). This repression allows the species downstream of that species to start being produced (e.g., CI) which in turn leads to the repression of the first species. This cycle continues causing this circuit to oscillate.

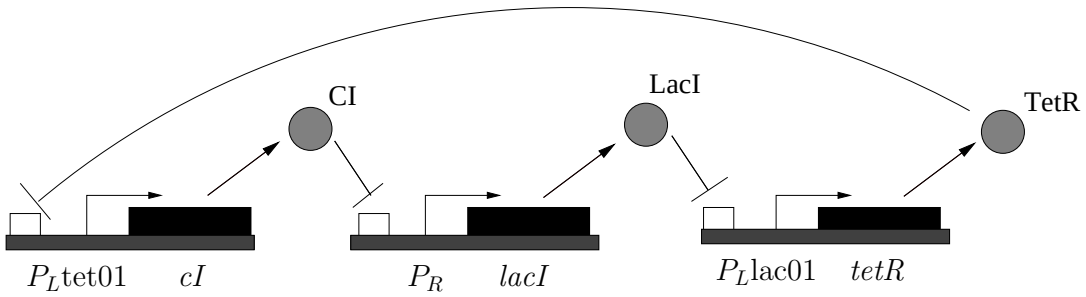



**Figure 6.5**: Genetic circuit model for repressilator. In this circuit, CI represses the production of LacI, LacI represses the production of TetR, and TetR represses the production of CI. This model is expected to produce oscillatory behavior of the CI, LacI, and TetR species.

An example of performing ODE and SSA simulations on the repressilator is shown in Figure 6.6. This figure shows that an individual SSA simulation run captures oscillations of the CI species. However, this figure also shows that the ODE simulation results, as well as, the average of 100 SSA simulations do not capture oscillations. Instead, they predict that the circuit more or less stabilizes at a false intermediate value.

The repressilator provides a good example of why the adaptive iSSA is a more accurate simulation method than its nonadaptive counterpart. An example of comparing the nonadaptive and the adaptive versions of the median path method on the repressilator is shown in Figures 6.7 and 6.8. Figure 6.7(a) and (b) show the result of simulating this circuit using the nonadaptive median path. With different time increment choices, the period of oscillation changes. However, when an adaptive version of the median path method is used as in Figure 6.8(a) and (b), the time increments change to try and capture 25 of the slowest reaction events in each increment which leads to stable results whether the model is simulated with or without reaction-based abstraction. Also, like the circadian rhythm example, the adaptive median path iSSA method has a run-time that is comparable to the SSA as both of these methods have a run-time of under 2 seconds.

After using the iSSA to verify that the repressilator oscillates, stochastic model checking can be used to determine the probability that the circuit oscillates. Figure 6.9 presents the results of applying steady state analysis to the repressilator using 9 levels evenly spaced between 0 and 80 for CI, LacI, and TetR and the CSL property, $\texttt{St}((\text{CI} \geq 30 \wedge \texttt{F}(t \leq \text{limit}, \text{CI} < 30) \geq 0.95) \vee (\text{CI} < 30 \wedge \texttt{F}(t \leq \text{limit}, \text{CI} \geq 30) \geq 0.95))$, which determines the likelihood that the value of the CI species is low and goes high or is high and goes low within a predetermined amount of time. These results show that

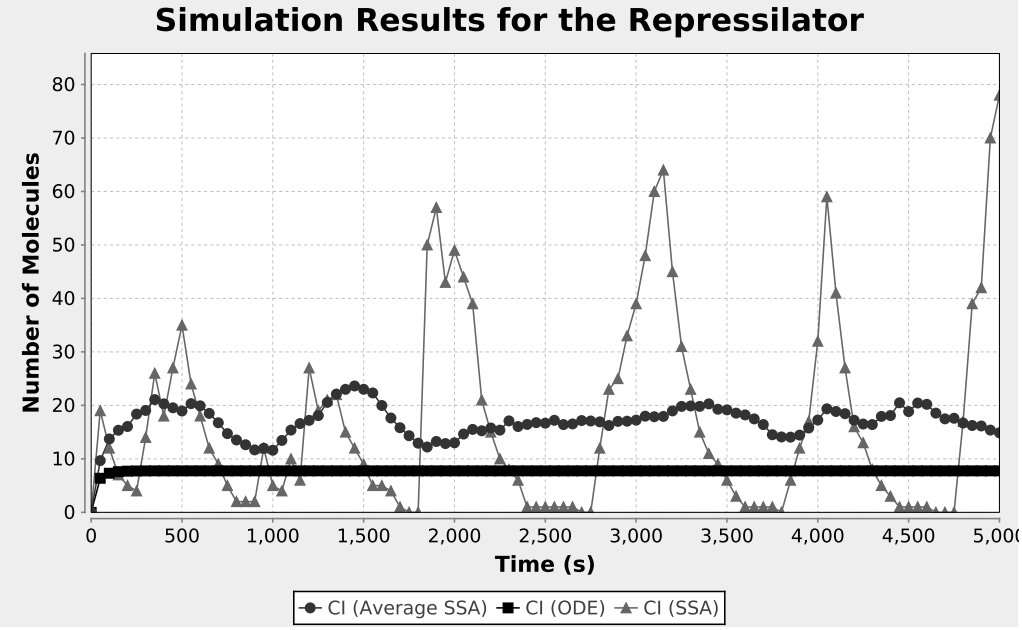
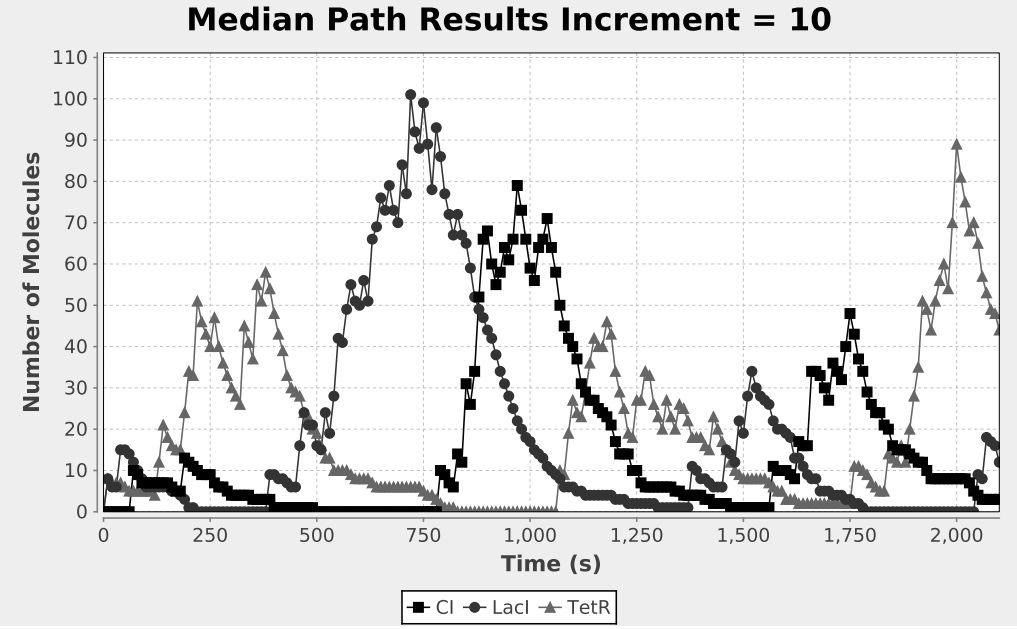
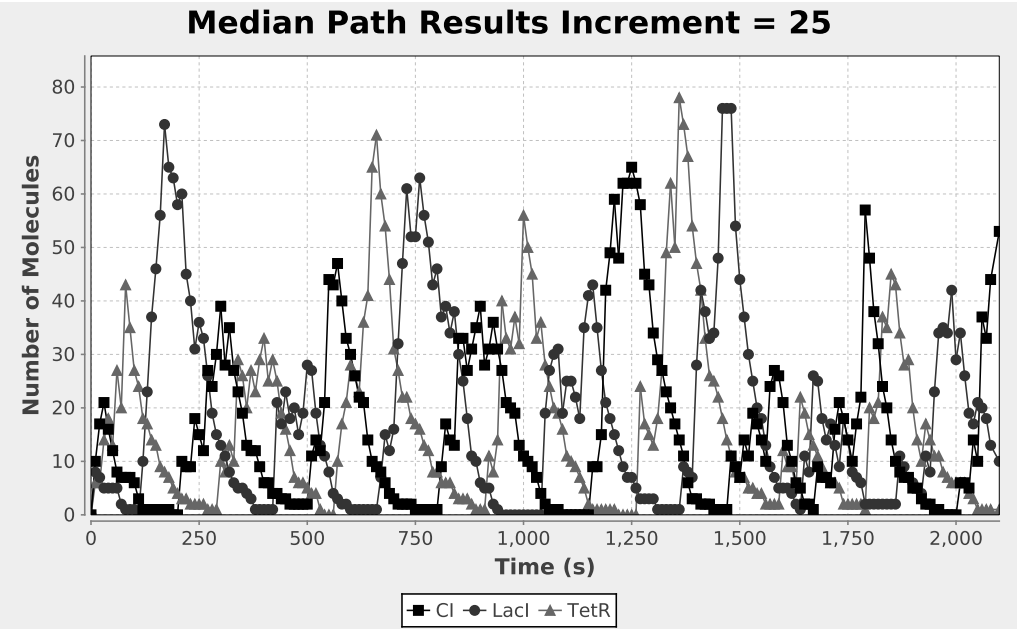



**Figure 6.6**: Simulation results for the repressilator. This plot shows an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs.

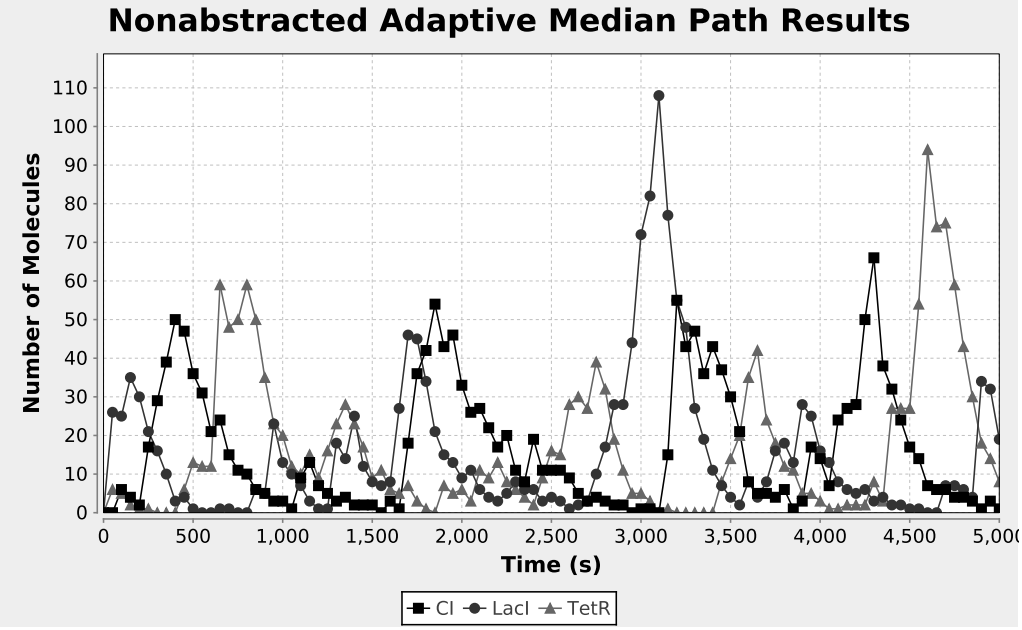



(a)

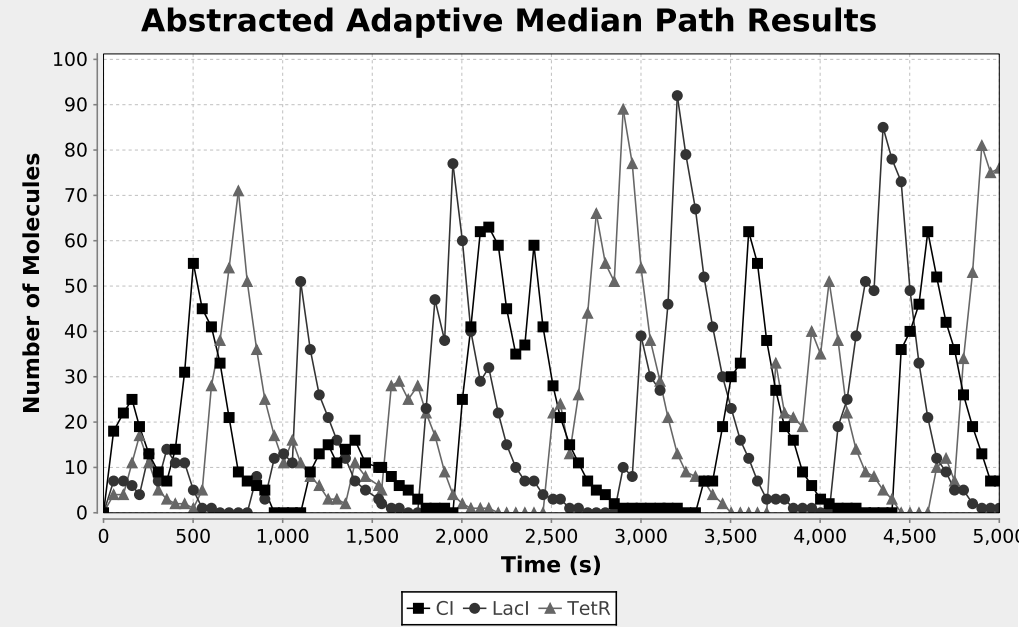



(b)

**Figure 6.7**: Nonadaptive median path simulation results for the repressilator. Each plot was produced with 100 simulation runs. (a) Results produced from the median path method with a time increment of 10. (b) Results produced from the median path method with a time increment of 25. As seen in these plots, the median path method produces different results for a different choice of time increment.




(a)




(b)

**Figure 6.8**: Adaptive median path simulation results for the repressilator with the desired number of events during each time increment equal to 25. (a) Results obtained without reaction-based abstraction. (b) Results obtained with reaction-based abstraction. This method adapts the time increment in order to try and capture 25 of the slowest reaction events in each increment which allows it to obtain good results with or without reaction-based abstraction.

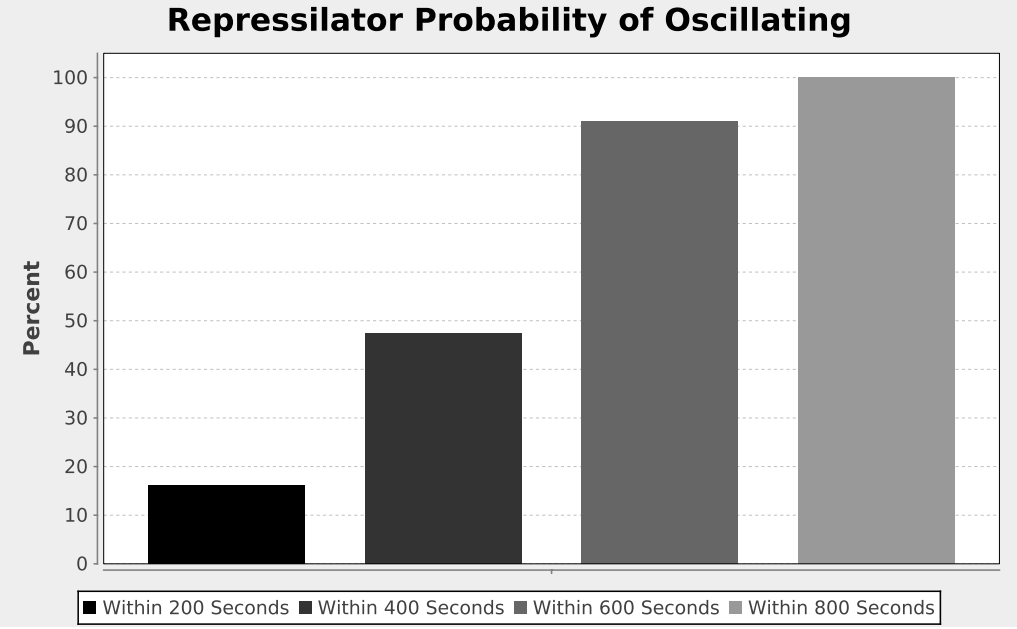



**Figure 6.9**: Stochastic model checking results for the repressilator. This plot presents the results of checking a property on the repressilator that represents states of the circuit switching from low to high or high to low within a specified amount of time. The probabilities represent the likelihood that the circuit oscillates.

as the time limit is extended, the likelihood of the circuit oscillating increases until for 800 seconds, it is almost certain to oscillate. In addition to the probability increasing as the time limit increases, the run-time also increases because it takes longer to perform analysis of nested properties with larger time bounds. The run-time for a time limit of 200 seconds is 3 minutes, for 400 seconds is 5 minutes and 40 seconds, for 600 seconds is 8 minutes and 25 seconds, and for 800 seconds is 11 minutes. This type of analysis can give a designer an idea of how reliable a circuit like the repressilator is as compared to other oscillator circuits so that the best circuit can be selected for a given task.

### 6.1.3 Dual-Feedback Genetic Oscillator

The dual-feedback genetic oscillator model shown in Figure 6.10 is composed of two identical promoters, $P_1$ and $P_2$, which are activated by AraC and repressed by LacI [77]. LacI is produced when transcription is initiated at promoter $P_1$ and the *lacI* gene is transcribed. Similarly, AraC is produced when transcription is initiated at promoter $P_2$ and the *araC* gene is transcribed. AraC builds up in the system causing LacI to build

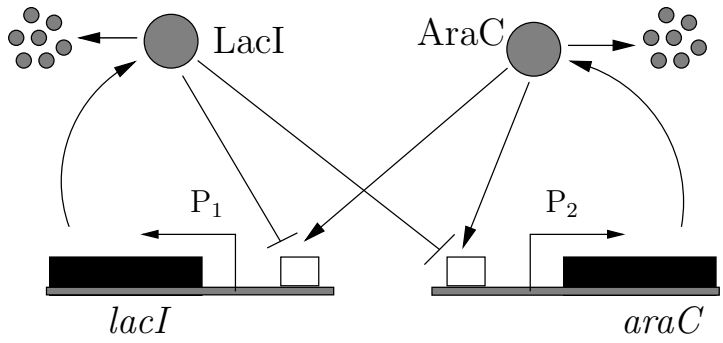



**Figure 6.10**: Genetic circuit model for the dual-feedback genetic oscillator. In this model, LacI represses itself and AraC through promoters $P_1$ and $P_2$. Conversely, AraC activates itself and AraC through the same promoters. This model is expected to produce oscillatory behavior of the LacI and AraC species.

up. LacI is then able to repress promoters $P_1$ and $P_2$ which causes both AraC and LacI concentrations to drop leading to AraC building up again causing the circuit to oscillate.

The model used is assumed to initially include 20 $P_1$ and $P_2$ promoters and 100 RNAP molecules. The ODE results and average of 100 SSA runs shown in Figure 6.11 are poor indicators of the system's behavior because the amount of AraC is never shown to reach its low point of zero like the individual SSA run in this plot does. In this system, the SSA sample-paths move out of phase very quickly, which can wash out the oscillations when conventional SSA averaging is performed. The iSSA results for this system are shown in Figure 6.12 and more clearly indicate the oscillatory behavior of this system. These results are obtained using the adaptive median path iSSA method and use 100 runs and 25 slow events per increment. This analysis provides more support for the fact that the adaptive iSSA results are found to be fairly insensitive to parameter variations. For instance, when the parameters are varied, e.g., 10 to 25 events per increment and 10 to 100 runs, the results do not markedly change. Additionally, the run-time for the ODE is under a second, for the SSA is about 2 seconds, and for the adaptive median path iSSA is about 1 second providing more evidence that the adaptive median path iSSA performs as efficiently as the SSA. This example demonstrates that the adaptive iSSA is robust when used to simulate highly dynamic reaction systems.

Similar to the repressilator, the dual-feedback genetic oscillator can be analyzed using stochastic model checking to determine the probability that it oscillates within a certain time bound. Figure 6.13 presents the results of applying steady state analysis to this model using 8 levels evenly spaced between 0 and 120 for AraC and LacI and the CSL property,

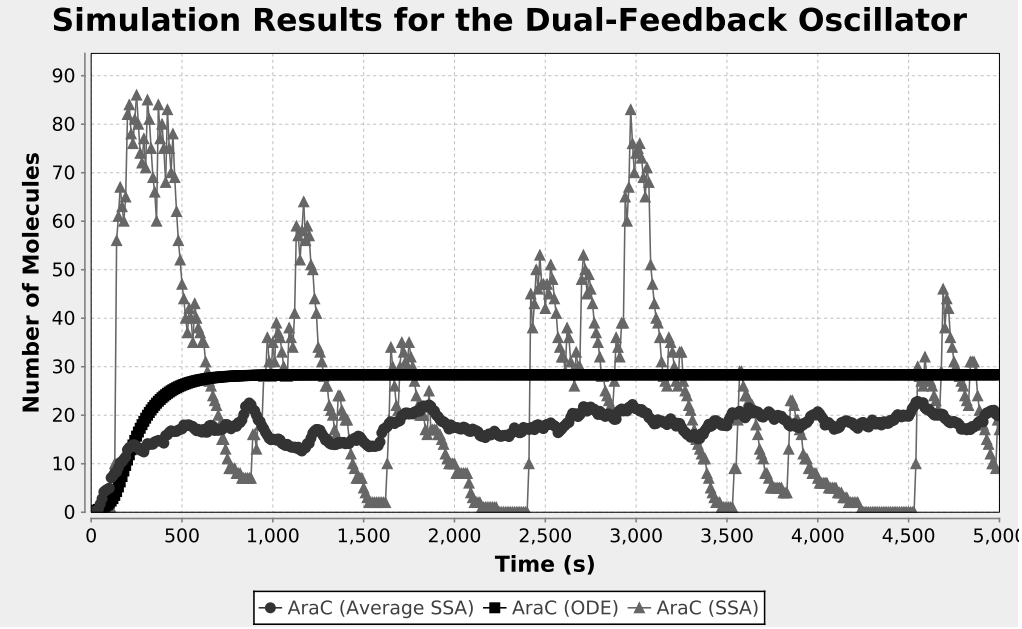



**Figure 6.11**: Simulation results for the dual-feedback genetic oscillator. This plot shows an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs.

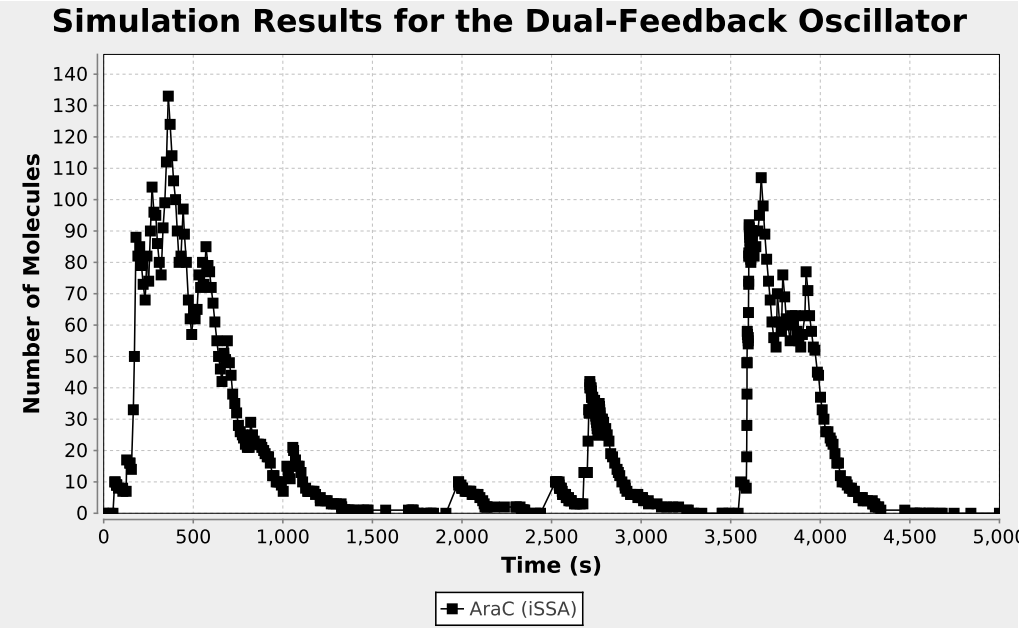



**Figure 6.12**: iSSA simulation results for the dual-feedback genetic oscillator. This plot shows the results of applying the adaptive median path iSSA method using 100 simulation runs and 25 slow reaction events per time increment.

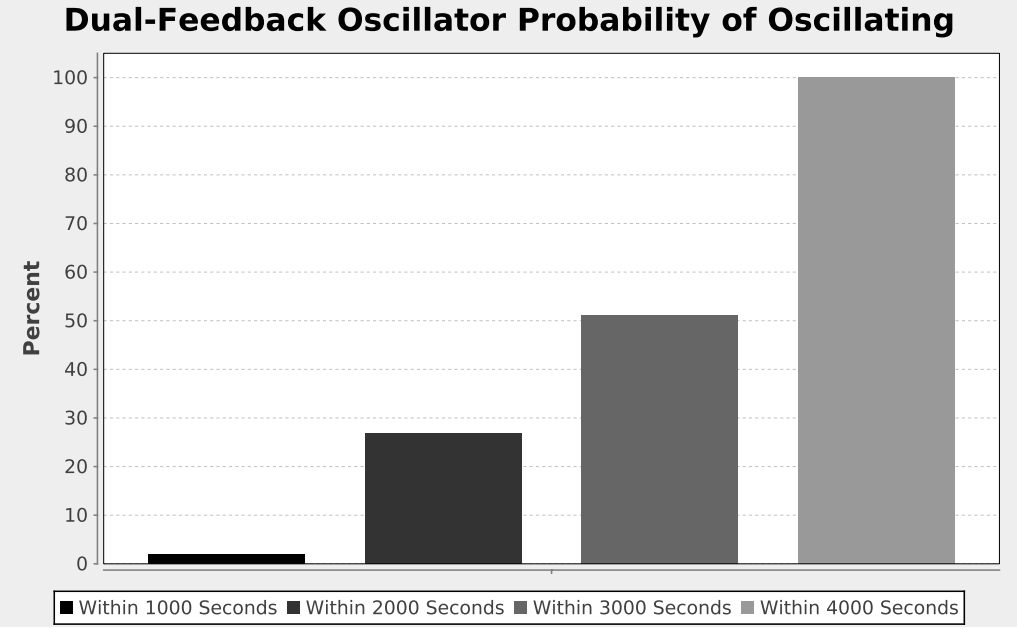



**Figure 6.13**: Stochastic model checking results for the dual-feedback genetic oscillator. This plot presents the results of checking a property on the dual-feedback genetic oscillator that represents states of the circuit switching from low to high or high to low within a specified amount of time. The probabilities represent the likelihood that the circuit oscillates.

$\texttt{St}((\text{AraC} \geq 60 \wedge \texttt{F}(t \leq \text{limit}, \text{AraC} < 60) \geq 0.95) \vee (\text{AraC} < 60 \wedge \texttt{F}(t \leq \text{limit}, \text{AraC} \geq 60) \geq 0.95))$, which captures the probability that AraC is low and goes high or is high and goes low within a predetermined amount of time. Like the repressilator, these results show that as the time limit is extended, the likelihood of the circuit oscillating increases. The run-time for a time limit of 1000 seconds is 40 seconds, for 2000 seconds is 1 minute and 25 seconds, for 3000 seconds is 1 minute and 55 seconds, and for 4000 seconds is 2 minutes and 40 seconds. These results show that the dual-feedback genetic oscillator has a much longer period than the repressilator. A designer could use this information to select the appropriate genetic oscillator based on how fast the oscillations need to be for a particular application.

## 6.2 State Holding Gates

Another type of construct that is probably as important to a cell as a genetic oscillator is a state holding gate. State holding gates serve to set cells into specific modes where they behave a certain way. These gates can then be reset by applying input signals to the circuit that cause the cell to transition into a different mode. An example of how

state holding gates can be beneficial to cells is to control change behavior when they move from a nutrient rich environment to a nutrient deficient environment. With the lack of nutrients as inputs to the cell, it can transition to an energy saving mode. Once the cell moves back into an environment where there are nutrients, the cell can then transition back into a more active mode and carry on with other cellular activities. This section analyzes the robustness of several state holding gates including the genetic toggle switch, three implementations of a genetic Muller C-element, and a quorum trigger model.

### 6.2.1 Toggle Switch

Figure 6.14 shows the results of simulating the genetic toggle switch, the running example throughout this dissertation, using ODEs and the SSA. In this figure, the toggle switch's inputs are varied and the output is supposed to go high at 5,000 seconds and back low at 15,000 seconds. This switching is reflected well in this plot as the ODE result, the single SSA result, and the average SSA result all follow this behavior. However, both the ODE and average SSA results fail to capture the noise that is observed when analyzing the individual SSA runs. On the other hand, the adaptive iSSA simulation shown in Figure 6.15 accurately captures the desired behavior, as well as the stochastic noise in the system. Run-times for each of these simulation methods are about 12 seconds. These results again show that performing simulations using iSSA is often as efficient as simulating with other simulation methods.

As shown in Figure 4.16 from Chapter 4, the iSSA predicts that when the genetic toggle switch is initialized to a state where all of the molecules are set to a low value and the inputs are not applied, the circuit either switches ON or OFF. In order to quantify the probability of each of these events happening, stochastic model checking can be utilized to determine the likelihood that the circuit ends up in the ON or the OFF state. Figure 6.16 presents results of applying both statistical model checking and steady-state Markov chain analysis to determine these probabilities. These results show that switching ON and OFF are roughly equally likely; however, the simulation approach takes 1 minute and 40 seconds to simulate 100,000 runs and obtain results while the Markov chain analysis is able to obtain results in about 1 second for 11 levels selected uniformly between 0 and 100 for both LacI and TetR.

Due to stochasticity and noise, a state holding gate like the toggle switch can fail. A useful experiment for this circuit is to determine the probability that it changes state erroneously within a cell cycle (2,100 seconds) which occurs if some spurious production of

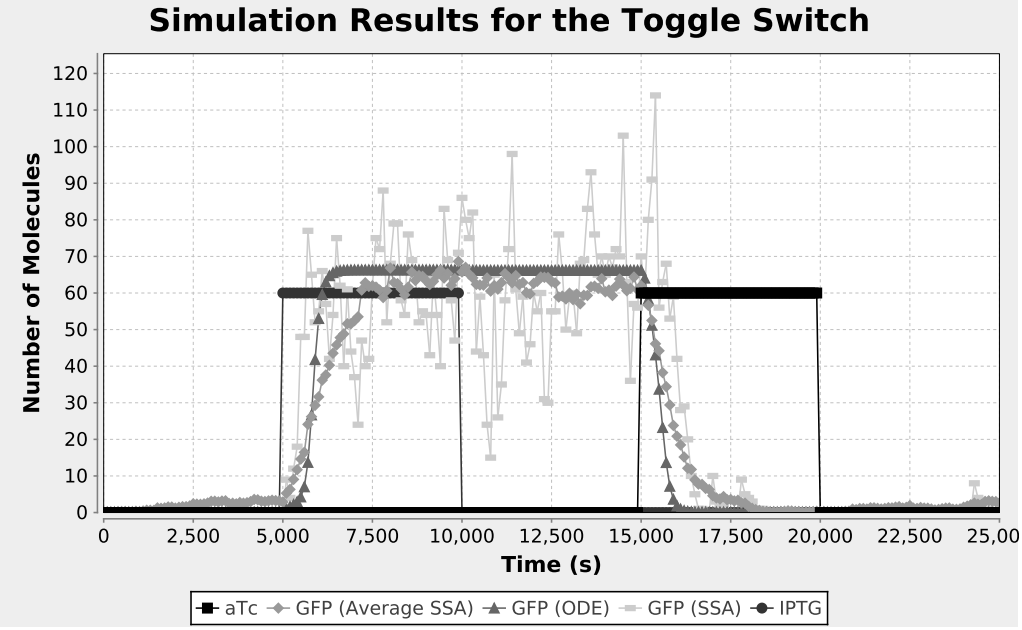



**Figure 6.14**: Simulation results for the genetic toggle switch. This plot shows an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs. IPTG is added to the circuit to set it at 5,000 seconds and taken away at 10,000 seconds. aTc is then added to reset it at 15,000 seconds and taken away at 20,000 seconds.

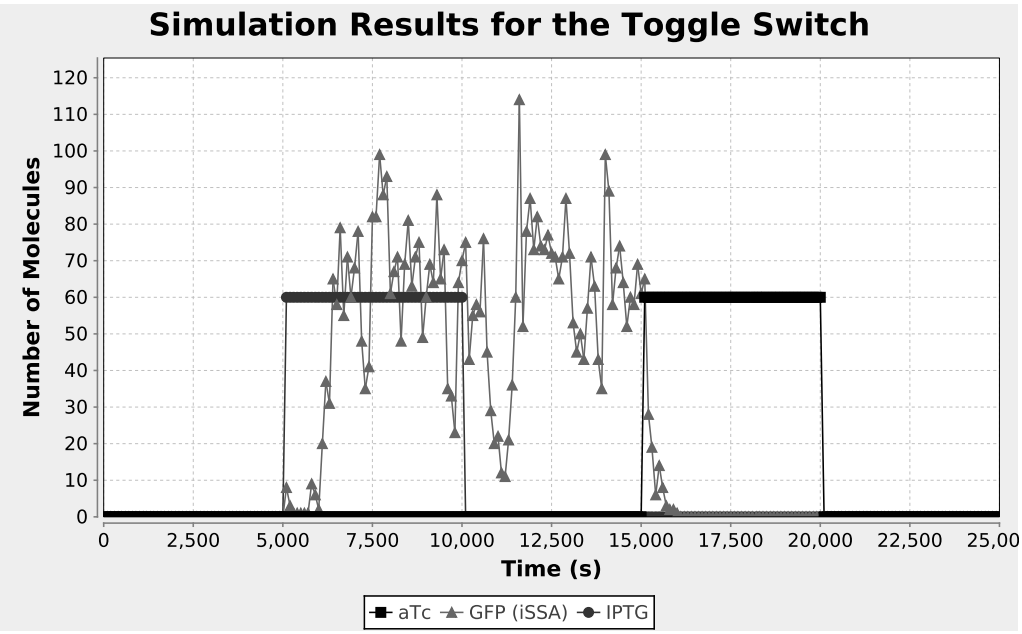
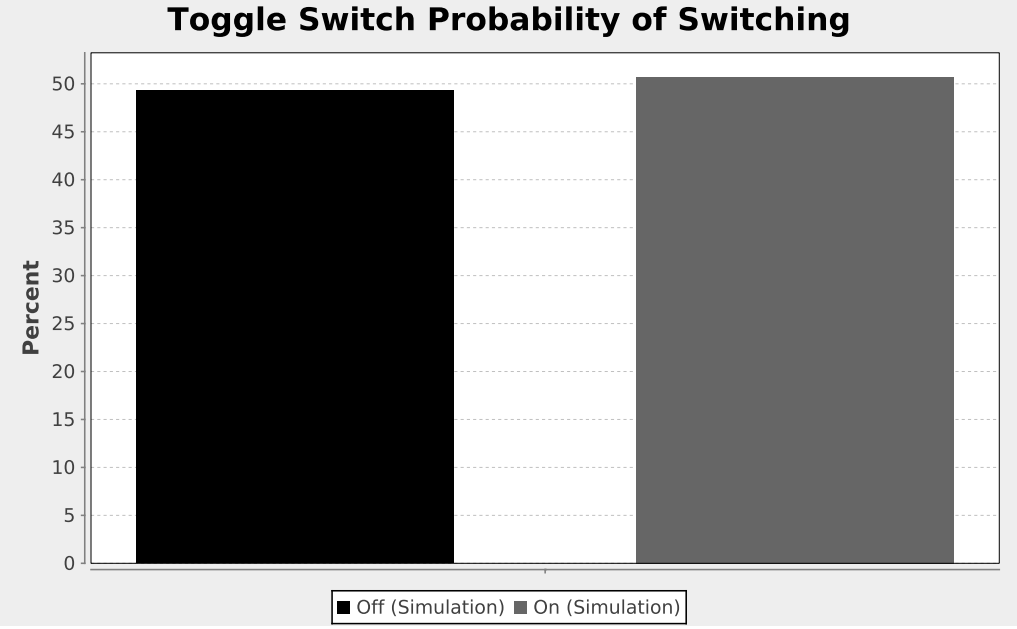
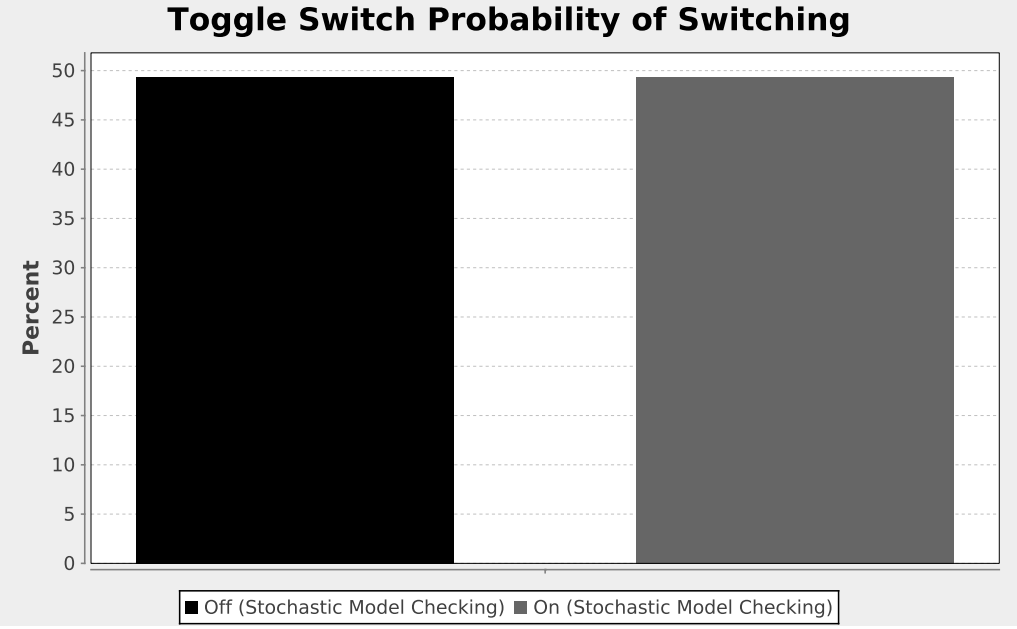



**Figure 6.15**: iSSA simulation results for the genetic toggle switch. This plot shows results of applying the adaptive median path iSSA method using 100 simulation runs and 25 slow reaction events per time increment. Like the Figure 6.14, the circuit is supposed to switch ON at 5,000 seconds and OFF at 15,000 seconds due to varying inputs.




(a)




(b)

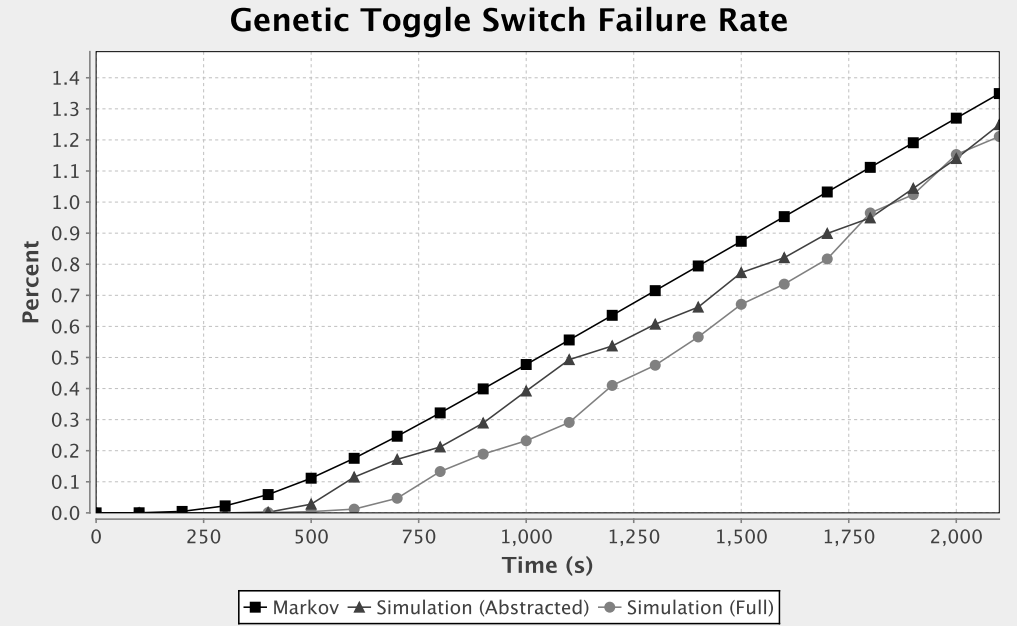
**Figure 6.16**: Stochastic model checking results for the genetic toggle switch with an initial state of both inputs low. (a) Results produced by performing 100,000 stochastic simulation runs of the toggle switch and statistically computing the proportion that switch ON and the proportion that switch OFF. (b) Steady-state Markov chain analysis results computed using the steady-state property $\texttt{St}((\text{TetR} > 40) \wedge (\text{LacI} < 20))$ to capture the circuit switching ON and the steady-state property $\texttt{St}((\text{LacI} > 40) \wedge (\text{TetR} < 20))$ to capture the circuit switching OFF.

the low signal inhibits the high signal enough to allow it to degrade away and switch state. For this experiment, the toggle switch is initialized to a starting state where LacI is set to a high state of 60 molecules and TetR is set to a low state of 0 molecules. In order to test whether or not it changes state, the CSL property, $\mathtt{F}(t \leq 2100, \mathrm{LacI} < 20 \wedge \mathrm{TetR} > 40)$, is checked. This property makes states absorbing in which LacI has dropped below 20 (the low state) and TetR has risen above 40 (the high state). For this analysis, 9 levels are selected for LacI uniformly distributed between 0 and 80, and 11 levels are selected for TetR uniformly distributed between 0 and 50, which produces a CTMC with 99 states. Levels are selected to ensure that one of the levels captures the initial amount for each species and that the levels span over the possible values for each species going slightly above and below the property bounds. It should be emphasized that this selection is a very simple and straightforward choice for the levels.

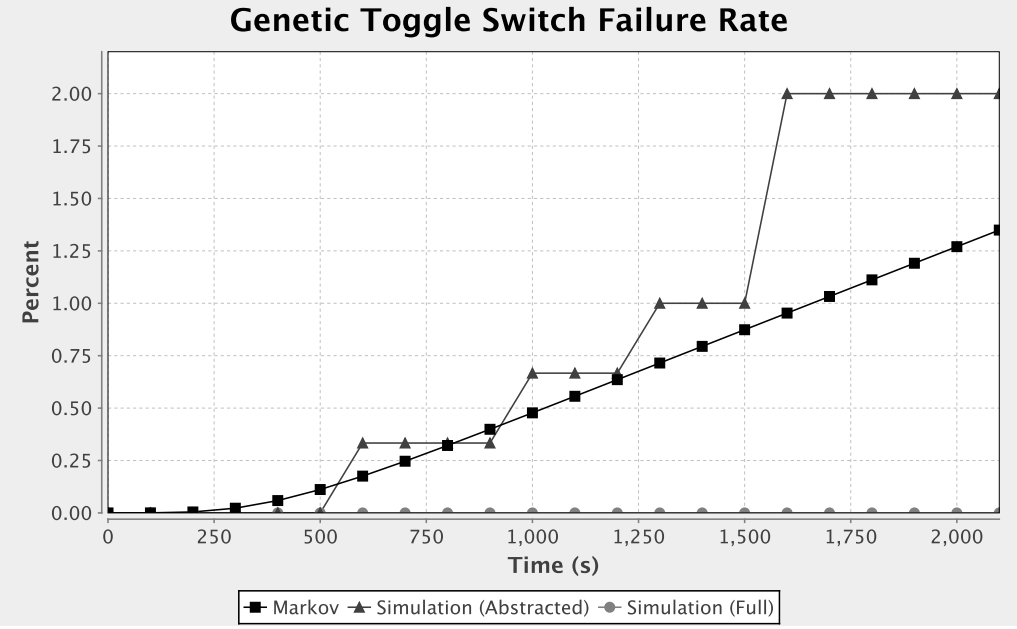
Figure 6.17(a) shows a comparison of results found using simulation both with and without reaction-based abstraction [56] and applying transient Markov chain analysis. This figure shows that the transient Markov chain analysis tracks the simulation results fairly closely and ends up with a final probability of 1.35 percent which is quite close to the 1.2 percent found by simulation of the full model. However, the transient Markov chain analysis method greatly outperforms the simulation based approaches as it takes under 1 second to obtain results while the simulation with abstraction takes about 3 minutes and 15 seconds to perform 32,000 runs and the simulation without abstraction takes about 43 minutes to perform the same number of runs. It is clear from Figure 6.17(a) that the stochastic simulations have not yet converged to the mean. The choice of 32,000 simulation runs is chosen in order to achieve 95 percent confidence that the result is within 10 percent assuming the true failure rate is 1.2 percent, the approximate value after 2100 seconds. This value is determined using the equation below:

$$d = 1.96 \times \sqrt{\frac{1-p}{p \times n}} \tag{6.43}$$

where $d$ is the relative error bound, $p$ is the predicted probability, and $n$ is the number of simulation runs [55, 37]. It should be noted that for earlier time points where the failure rate is lower, the error increases. For example, at 1000 seconds, the full simulation predicts a probability of failure of 0.3 percent, but we are only 95 percent confident that this result is within 20 percent of the true value. Figure 6.17(b) shows another comparison of applying these same analysis methods to the genetic toggle switch but with similar run-times to the Markov analysis. In this plot, the full model could only perform 20 runs in under 1




(a)




(b)

**Figure 6.17**: Time course plots showing the probability of the genetic toggle switch changing state erroneously. (a) Compares the results of performing 32,000 simulation runs both with and without reaction-based abstraction with Markov chain analysis. (b) Compares the same results as in (a) but with 300 simulation runs with reaction-based abstraction and 20 runs without reaction-based abstraction in order to match the run-time of the Markov chain analysis. The CSL property in both cases is $\mathtt{F}(t \leq 2100, \mathrm{LacI} < 20 \wedge \mathrm{TetR} > 40)$.

second and the abstracted model had enough time for 300 runs. Using Equation 6.43, this computation results in errors of at least 382 percent for the full model and 99 percent for the abstracted model.

The next experiment is to determine the response time of the circuit when switching from the OFF state to the ON state, and these results are presented in Figure 6.18. This analysis uses the same CSL property but a slightly different initial condition. As before, LacI is set to 60 and TetR is set to 0, but IPTG is set to 100 representing that it has just been added to set the toggle switch to the high state. For this experiment, 14 levels for LacI are selected uniformly distributed between 0 and 130, since individual simulation results show it reaching a much higher value than in the last experiment. For TetR, only 5 levels are used uniformly selected between 0 to 60 because less resolution is required for catch its change from a low to high state. This level selection results in a CTMC of 70 states. Again, transient Markov chain analysis tracks the simulation results fairly closely ending up with a final probability of 98.7 percent while the simulation of the full model results in 98.9 percent. Also like the previous example, the transient Markov chain

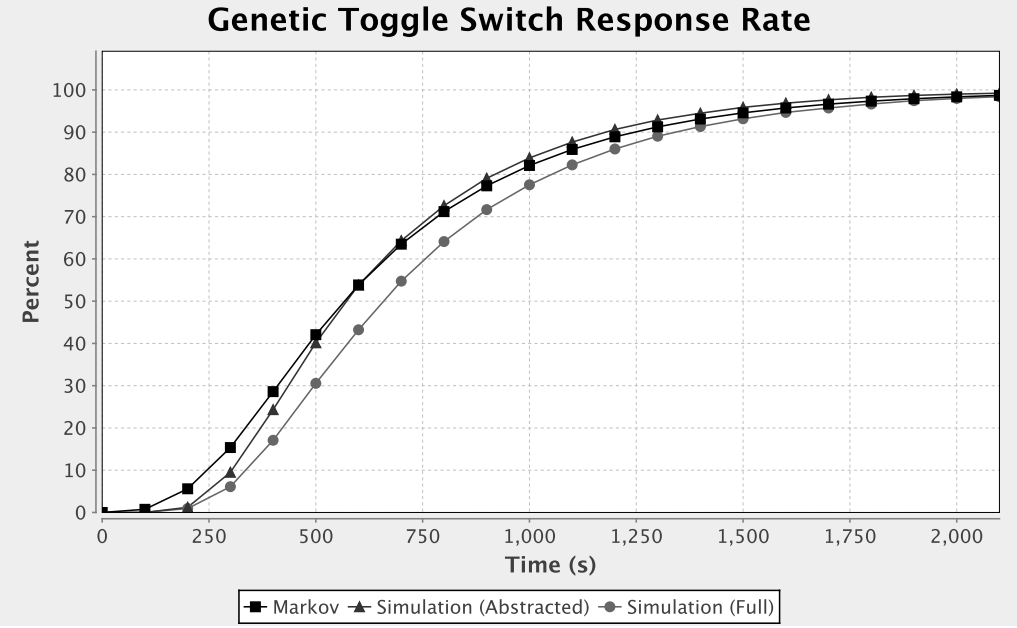



**Figure 6.18**: Time course plot showing the probability of the genetic toggle switch changing state correctly in response to an input change. Like Figure 6.17, this plot compares the results of using simulation both with and without reaction-based abstraction and analysis of the CTMC using Markov chain analysis with the same CSL property, $\mathtt{F}(t \leq 2100, \mathrm{LacI} < 20 \wedge \mathrm{TetR} > 40)$, but with a different initial value of IPTG.

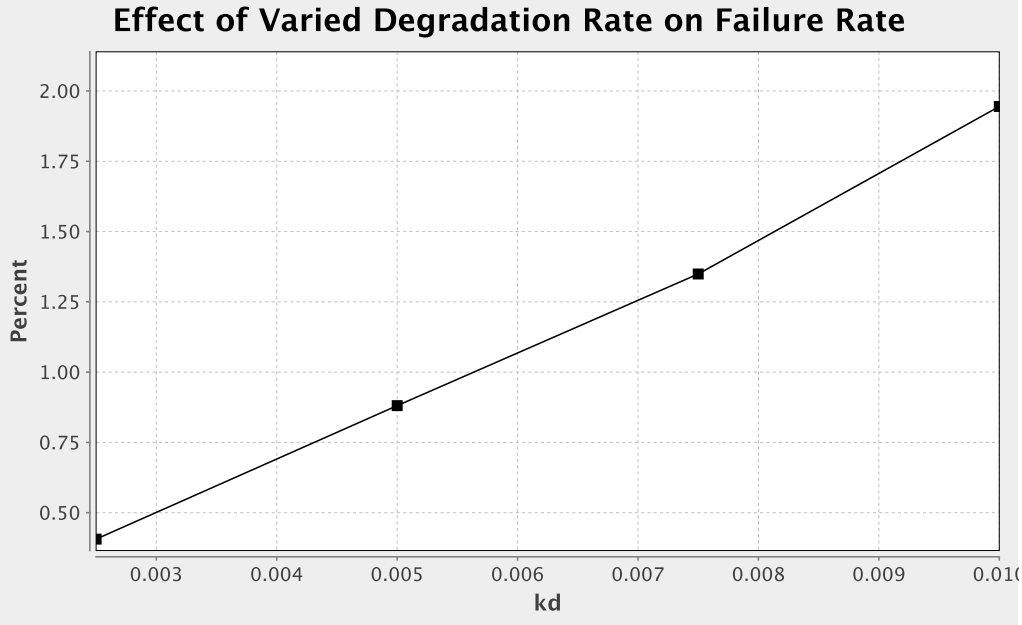
analysis method outperforms the simulation-based approaches as it takes about a half a second to obtain results while 32,000 simulation runs of the reaction-based abstracted model takes about 1 minute and the full model takes about 3 hours and 12 minutes. It should be noted that the reason that the full model takes so much longer to simulate than the abstracted model is that in the presence of IPTG, the full model simulation spends an exorbitant amount of time fire binding and unbinding reactions of LacI and IPTG.

With this analysis method, the design space can be efficiently explored. For example, a genetic designer may consider the effect of parameter variation on robustness and performance. One important parameter for the genetic toggle switch is the degradation rate, $k_d$, and the results of varying this parameter are shown in Figure 6.19. These results indicate that tuning the degradation rate has a significant effect. If it is too high, the circuit is less robust, but if it is too low, it responds too slowly.

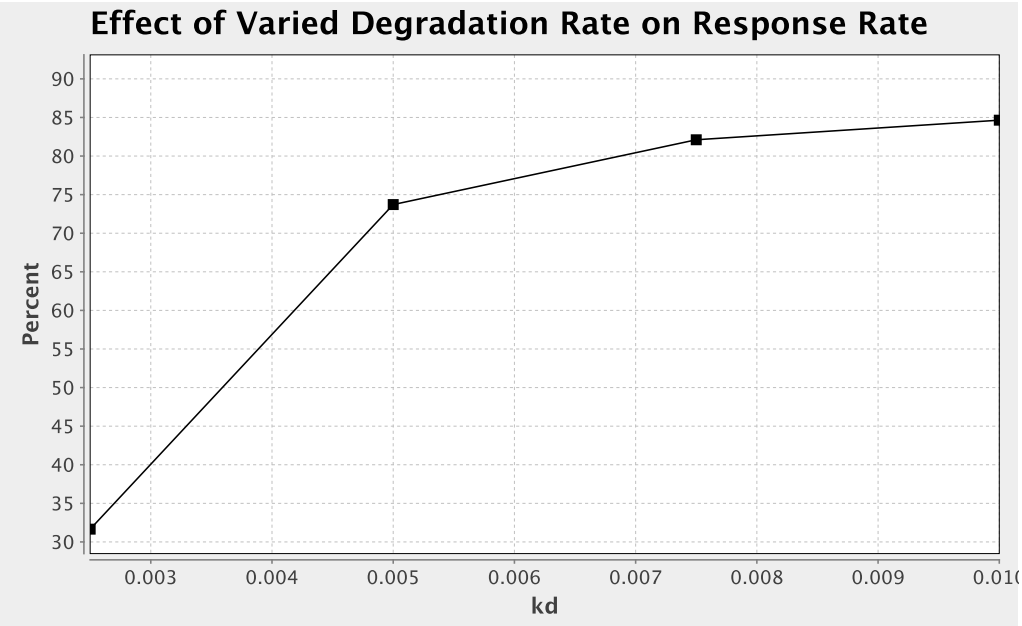
### 6.2.2 C-Element

The next circuit analyzed is a genetic Muller C-element [67, 66]. C-elements are commonly used by asynchronous designers to coordinate parallel processes. Our analysis considers three implementations of this circuit with their logic diagrams shown in Figure 6.20. For each of these circuits, the inputs are IPTG and aTc and the output is GFP similar to the genetic toggle switch example. The first implementation shown in Figure 6.20(a) has a majority gate design where the two inputs to the circuit and a feedback signal from the circuit's output are fed through three NAND gates in varying combinations. The outputs from these NAND gates are then compared with each other and the signal that has the majority of the votes is selected as the new output. The next implementation shown in Figure 6.20(b) has a speed-independent design. The idea behind this circuit is that no matter how fast or slow the gates change, the circuit behaves correctly. The final implementation shown in Figure 6.20(c) uses the genetic toggle switch described earlier with some additional logic. This circuit takes an inverted NAND gate signal of the two inputs as the set part of the circuit and an inverted NAND gate signal of the inverted inputs as the reset part of the circuit.

These circuits have slightly different state holding properties when compared to the genetic toggle switch. Instead of switching when one input is applied or taken away, these circuits only switch when both inputs are applied; however, they maintain whatever state they are in until both of the inputs are simultaneously ON or OFF. Figures 6.21, 6.22, and 6.23 present ODE, SSA, and iSSA simulations of the majority gate, speed-independent,




(a)




(b)

**Figure 6.19**: Results showing the effect of varying the degradation rate, $k_d$, for the genetic toggle switch. (a) Plot depicting the probability of the genetic toggle switch changing state erroneously within 2100 seconds for different values of $k_d$. (b) Plot depicting the probability of the genetic toggle switch changing state correctly within 2100 seconds in response to input change for different values of $k_d$.

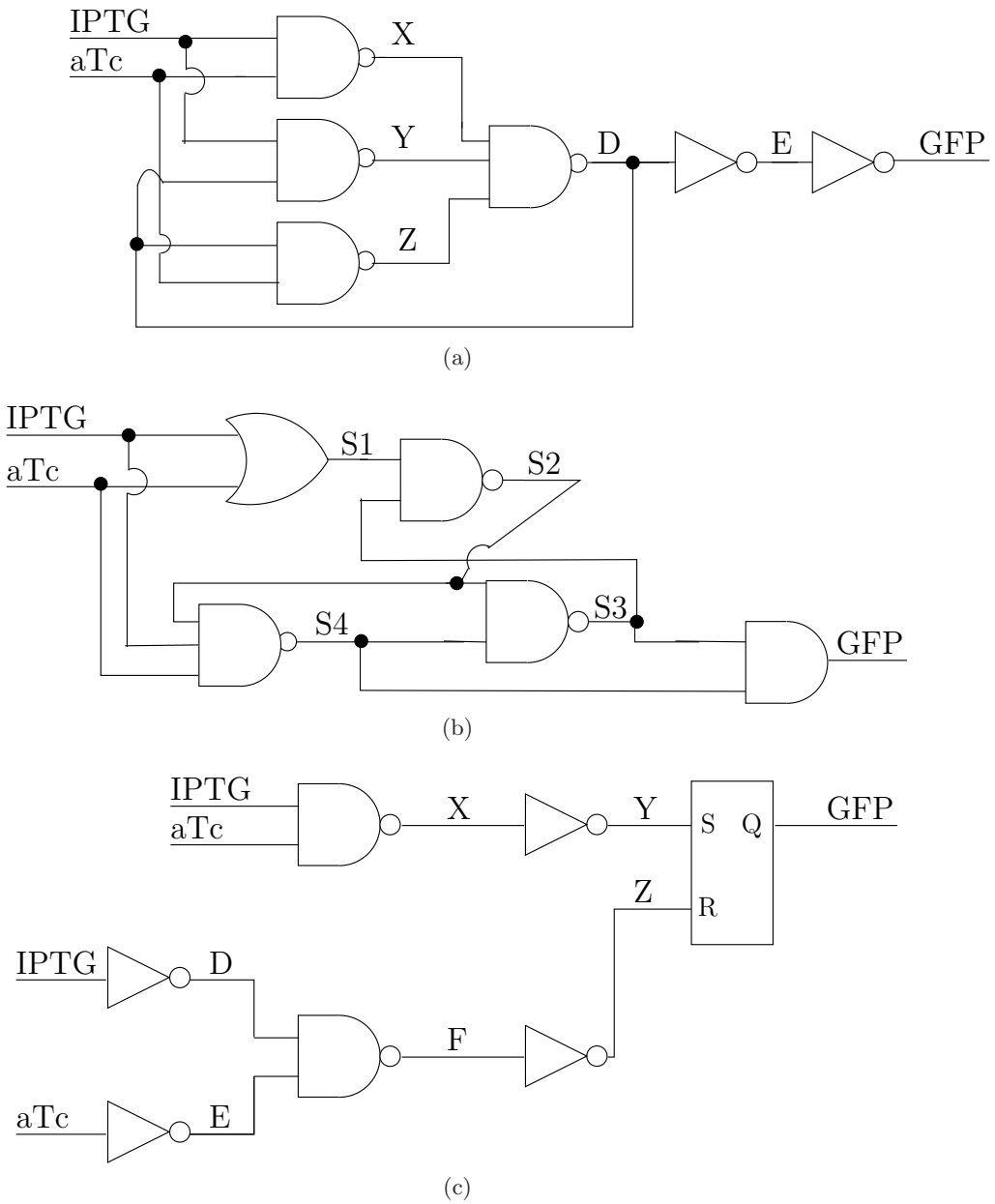
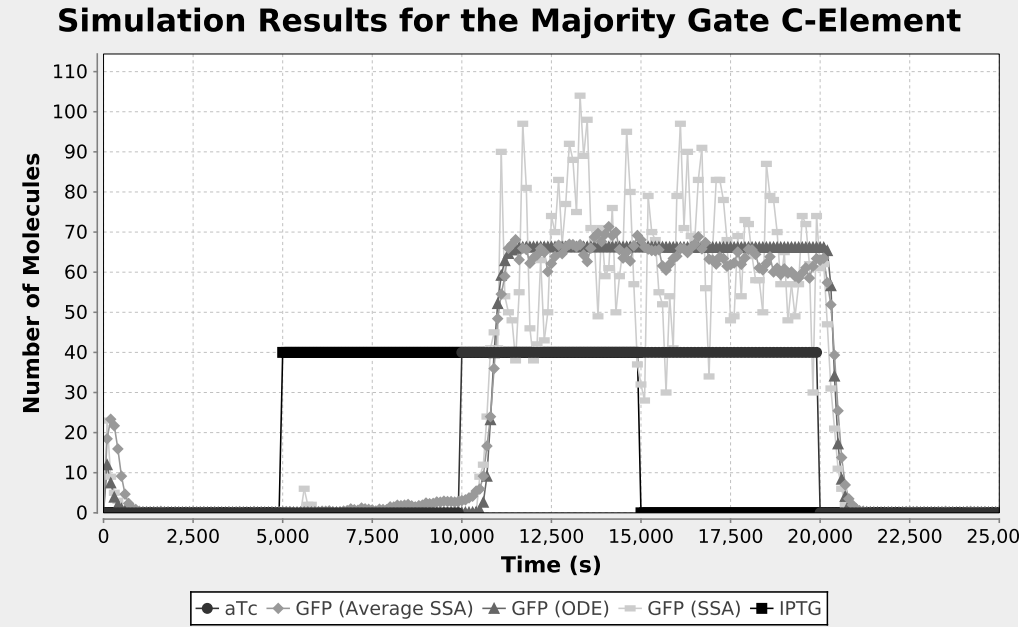
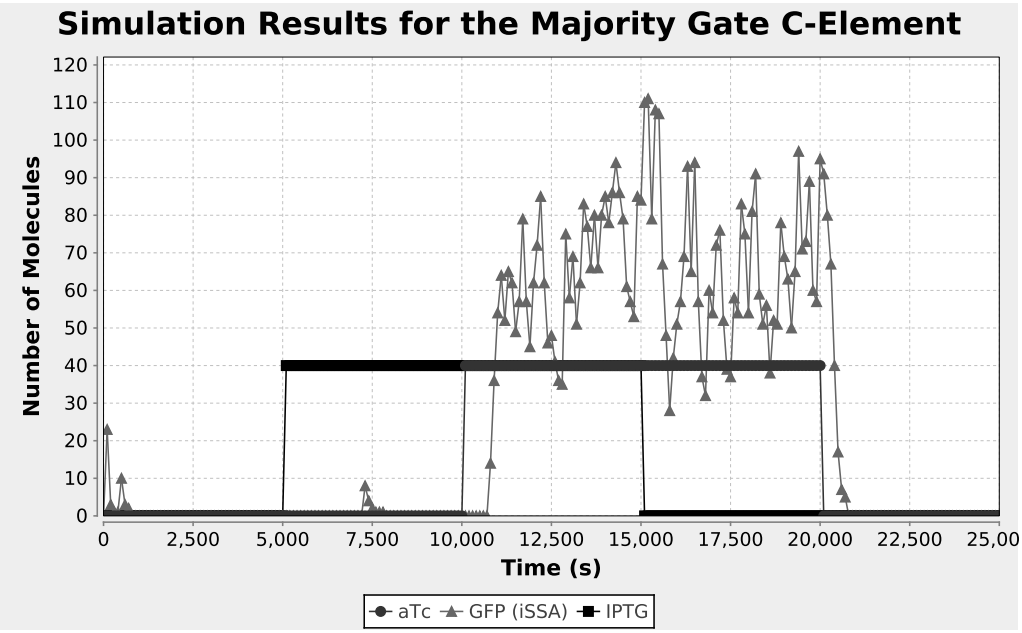



**Figure 6.20**: Logic diagrams for the genetic Muller C-element. (a) Majority gate design. (b) Speed-independent design. (c) Toggle switch design.

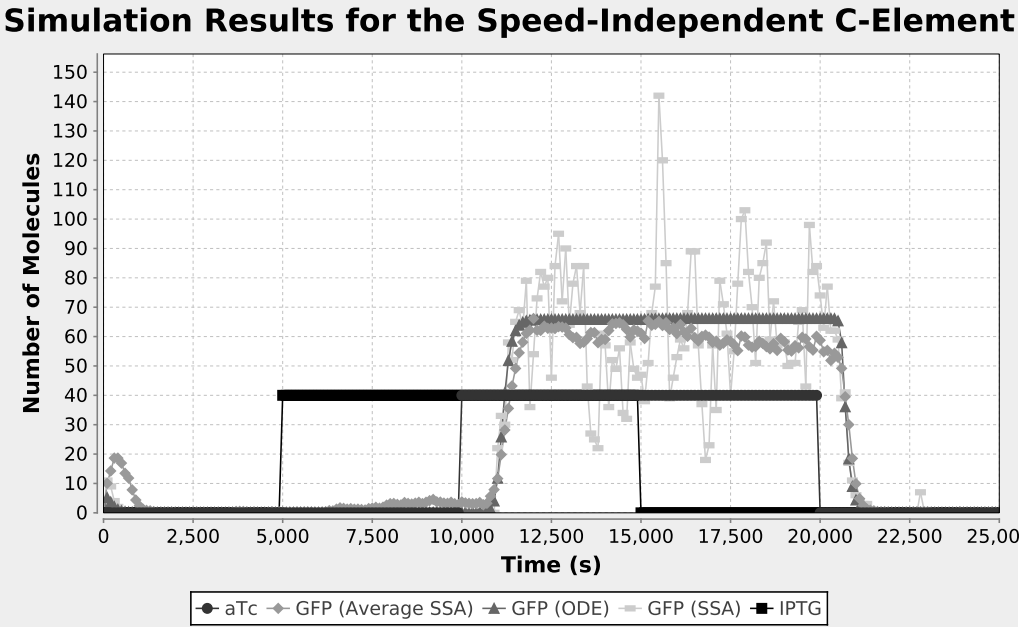



(a)

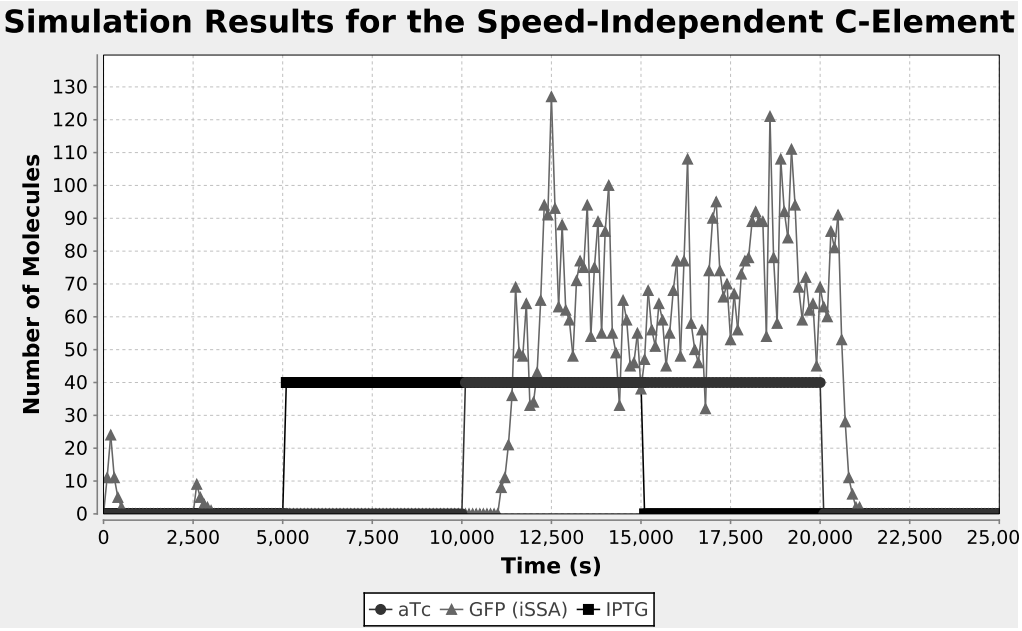



(b)

**Figure 6.21**: Simulation results for the majority gate C-element. (a) Plot showing an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs. (b) Plot depicting results of applying the adaptive median path iSSA method using 100 simulation runs and 25 slow reaction events per time increment.

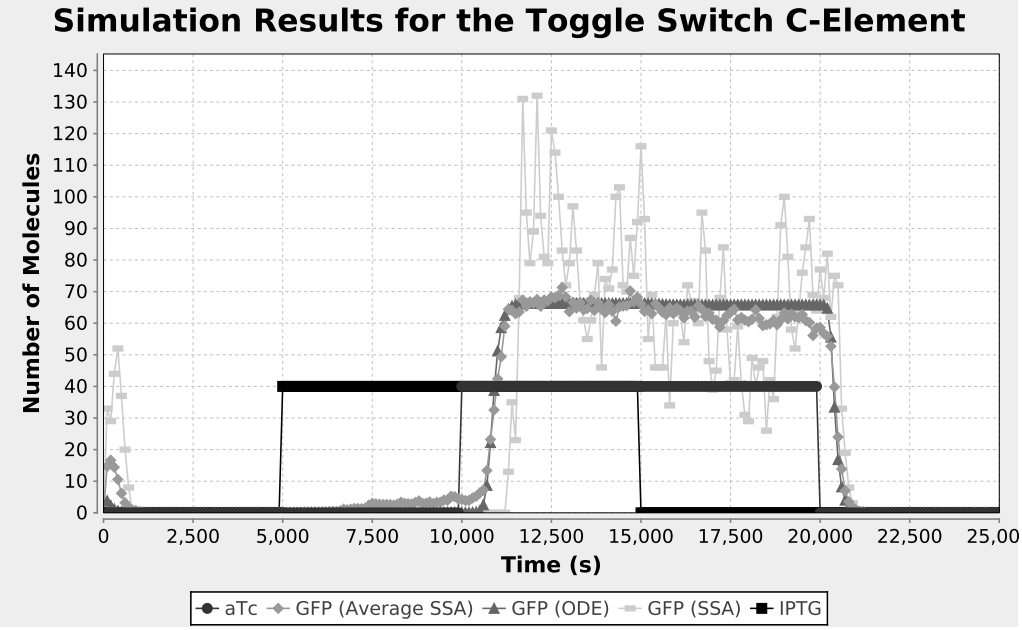



(a)

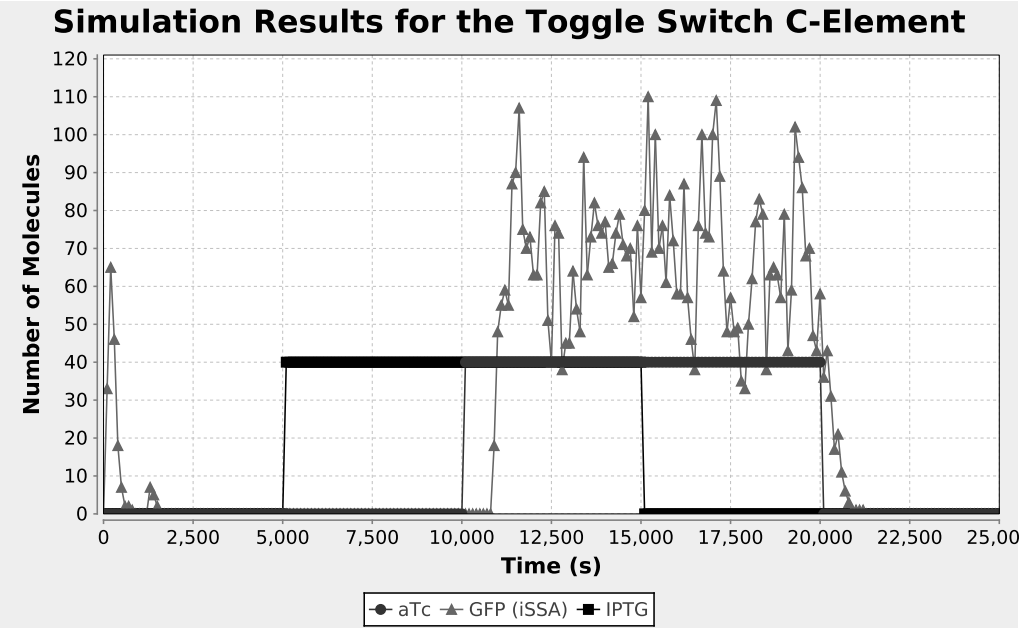



(b)

**Figure 6.22**: Simulation results for the speed-independent C-element. (a) Plot showing an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs. (b) Plot depicting results of applying the adaptive median path iSSA method using 100 simulation runs and 25 slow reaction events per time increment.




(a)



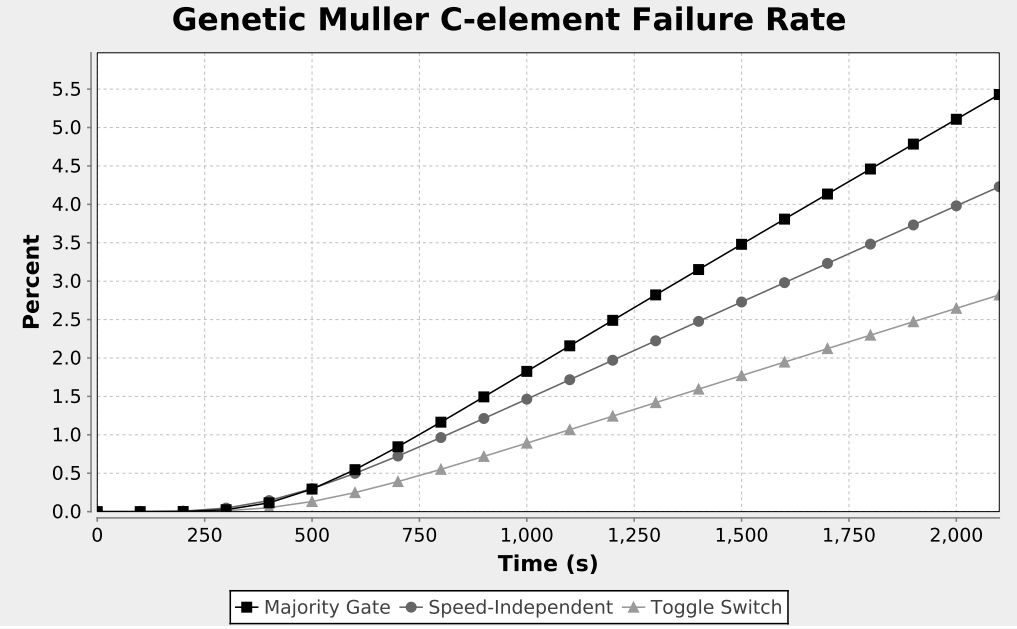

(b)

**Figure 6.23**: Simulation results for the toggle switch C-element. (a) Plot showing an ODE simulation run, a single SSA simulation run, and the average of 100 SSA simulation runs. (b) Plot depicting results of applying the adaptive median path iSSA method using 100 simulation runs and 25 slow reaction events per time increment.

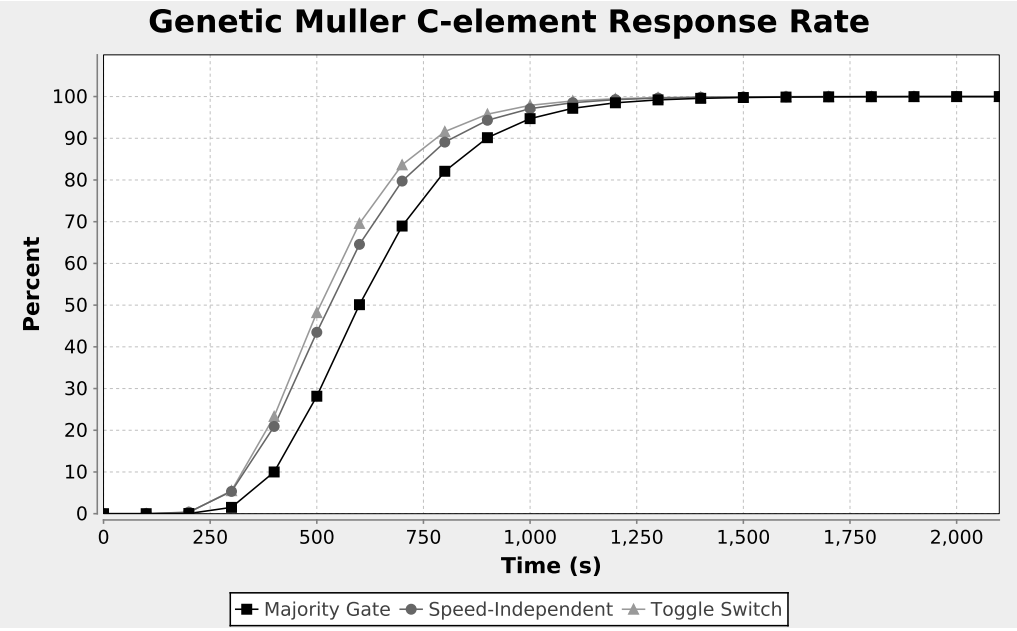
and toggle switch implementations, respectively. In each of these analyses, IPTG is added to the circuit at 5,000 seconds. Each circuit does not switch from OFF to ON, however, until aTc is also added at time 10,000 seconds. When IPTG is removed at 15,000 seconds, the circuits maintain state until aTc is removed at 20,000 seconds. These results show that the ODE and average SSA simulations capture these state changes but fail to capture the stochastic noise visible in the individual SSA run. Conversely, the iSSA results clearly show this stochastic noise. For each of these circuits, run-times for ODE are about 30 seconds, for SSA are about 1 minute, and for iSSA are about 1 minute once again showing that the efficiency of iSSA is comparable to that of SSA.

Since stochastic noise is present in these circuits as it is in the genetic toggle switch, these circuits are analyzed to find their failure and response rates. Instead of analyzing these circuits against a property that only checks the amount of GFP, dual-rail properties are analyzed because they provide a better measure of the circuit changing state. Figure 6.24(a) shows a comparison of the failure rate analysis when each circuit is set in its high mixed state meaning that one input is high, one input is low, the output is high, and the internal species are set appropriately. For this analysis, the CSL property, $\mathtt{F}(t \leq 2100, \mathrm{GFP} < 20 \wedge \mathrm{E} > 40)$, is used to analyze the majority gate implementation with 16 evenly spaced levels for GFP between 0 and 150, 16 evenly spaced levels for E between 0 and 45, 6 evenly spaced levels for D between 0 and 250, and 5 evenly spaced levels for X, Y, and Z between 0 and 120. The CSL property for the speed-independent implementation is $\mathtt{F}(t \leq 2100, \mathrm{S3} < 20 \wedge \mathrm{S2} > 80)$, and the levels used are 11 evenly spaced levels for S2 between 0 and 100, 11 evenly spaced levels for S3 between 0 and 150, 6 evenly spaced levels for S1 between 0 and 250, 4 evenly spaced levels for S4 and GFP between 0 and 120, and 4 evenly spaced levels for X, Y, and Z between 0 and 90. Finally, the CSL property for the toggle switch implementation is $\mathtt{F}(t \leq 2100, \mathrm{Y} < 40 \wedge \mathrm{Z} > 80)$, and the levels used are 16 evenly spaced levels for Y between 0 and 225, 16 evenly spaced levels for Z between 0 and 90, 6 evenly spaced levels for F between 0 and 250, 5 evenly spaced levels for GFP between 0 and 120, and 5 evenly spaced levels for D, E, and X between 0 and 80.

Run-times for these analyses are 13 minutes and 15 seconds for the majority gate, 20 minutes and 15 seconds for the speed-independent, and 21 minutes and 45 seconds for the toggle switch. This increase in run-time over the genetic toggle switch is due to the fact that each of these models contains substantially more species. In addition, most of




(a)




(b)

**Figure 6.24**: Results showing the probability of the C-elements changing state for varying inputs. (a) Time course plot showing the probability of the C-element implementations losing state with mixed inputs. (b) Time course plot showing the probability of the C-element implementations changing state correctly in response to both inputs changing state. These plots compare the results of using Markov chain analysis on the majority gate implementation, the speed-independent implementation, and the toggle switch implementation.

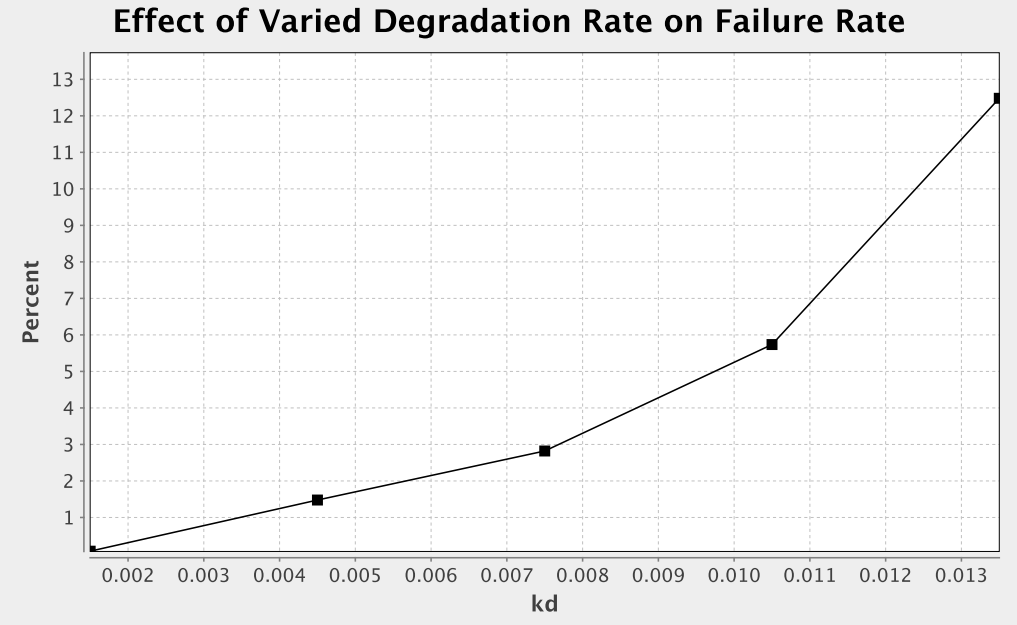
the species in the C-element models have more levels defined for them resulting in larger CTMCs. For instance, compared to the genetic toggle switch's 70 to 99 states, the majority gate implementation has nearly 200,000 states, the speed-independent implementation has about 750,000 states, and the toggle switch implementation has nearly 1,000,000 states. From the plot in Figure 6.24(a), it can be discerned that the toggle switch implementation is the most likely to maintain its state with mixed inputs, followed by the speed-independent implementation, and finally the majority gate implementation.

In order to determine the response time of the C-element circuits, the same initial condition, levels, and properties for each circuit are used with the exception that both inputs are set low indicating that the circuit's output should change from high to low. The results of this analysis are shown in Figure 6.24(b) where it can be seen that the toggle switch implementation again outperforms the speed-independent and majority gate circuits. These analyses have similar run-times to the failure rate experiment with the majority gate taking 11 minutes and 15 seconds, the speed-independent taking 20 minutes, and the toggle switch taking 22 minutes.

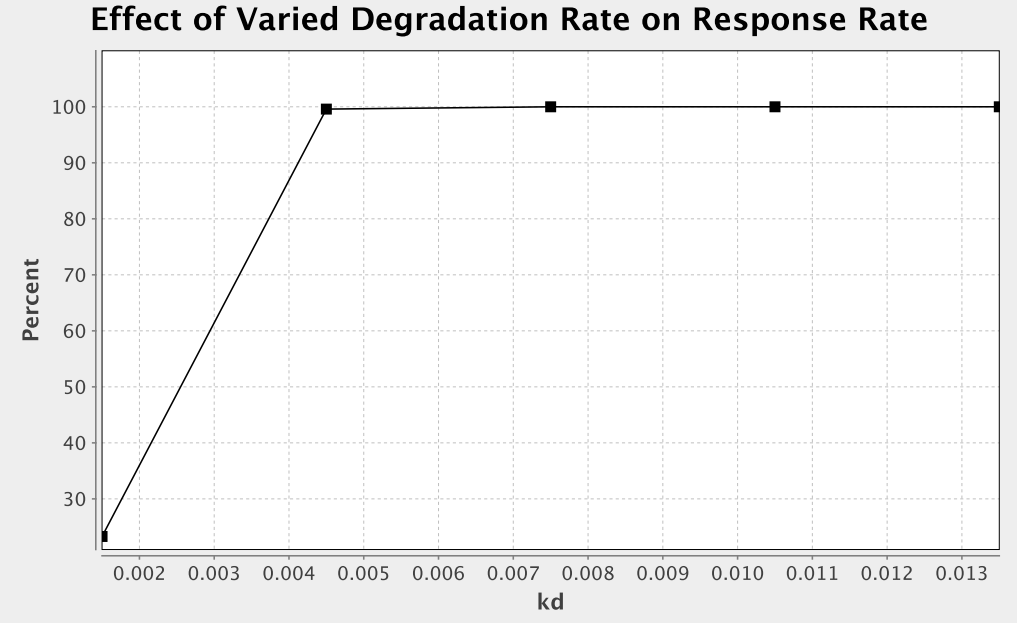
After determining that the genetic toggle switch Muller C-element is the most robust (perhaps a surprising conclusion to some), we can now perform various parameter variation experiments to determine the best parameter choices for the application. Figure 6.25 gives an example of varying the degradation rate of this circuit similar to the experiments performed on the genetic toggle switch in Figure 6.19. These results show comparable behavior to that of the genetic toggle switch indicating that there is a trade-off between robustness and responsiveness when varying this parameter.

### 6.2.3 Quorum Trigger

The final circuit analyzed is a quorum trigger model presented in Figure 6.26. This circuit is designed to be placed in many cells in a population where it is supposed to allow cells to switch into an ON state in the presence of a signal in the environment. It works by allowing the production of LuxR to be activated by an environmental signal which can then bind with a derivative of LuxI known as 3OC6HSL to form a complex. This complex then continues to activate LuxR and LuxI production leading to the cell locking into the ON state. In order to reinforce the switching in all the cells in the quorum, this circuit contains a method to communicate to other cells through 3OC6HSL signal which can diffuse through the cellular membrane and into the medium to be taken up by other cells and affect their quorum trigger circuits.




(a)




(b)

**Figure 6.25**: Results showing the effect of varying the degradation rate, $k_d$, for the toggle switch implementation of the genetic Muller C-element. (a) Plot depicting the probability of the toggle switch implementation of the genetic Muller C-element changing state erroneously for different values of $k_d$. (b) Plot depicting the probability of the toggle switch implementation of the genetic Muller C-element changing state correctly in response to an input change for different values of $k_d$.

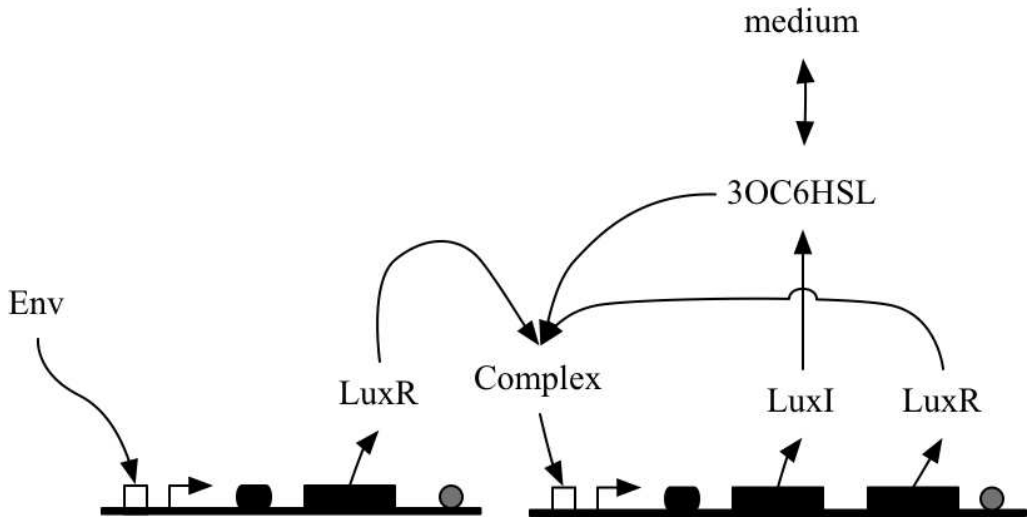



**Figure 6.26**: Genetic circuit model for quorum trigger circuit. In this model, the production of LuxR is activated by an signal in the environment of the circuit. Additionally, production of both it and LuxI are activated by a complex which is formed when LuxR binds with 3OC6HSL, a derivative of LuxI that diffuses through a cell's membrane and into the medium the cell is in. This circuit basically works by detecting the presence of the environmental signal and switching to an ON state where it then communicates with other cells through the 3OC6HSL signal to let them know that they should also switch ON.

Figures 6.27, 6.28, and 6.29 present the results of applying the iSSA to the quorum trigger while varying the basal rate of production. In Figure 6.27, this rate is set to 0 and the iSSA predicts that the circuit does not switch ON when this is the case with a run-time of 25 seconds. When the rate is increased to 0.0001 as in Figure 6.28, the iSSA shows that the circuit only switches when the environmental signal is set to a high value with a run-time of 1 minute and 5 seconds. However, the iSSA always predicts that the circuit switches ON when the basal rate is too high with a run-time of 2 minutes and 10 seconds as shown in Figure 6.29.

Stochastic model checking analysis of the quorum trigger circuit is able to confirm the predictions made by the iSSA and give some insight into the likelihood of the circuit switching ON for various values of $k_b$. Figure 6.30 shows that the probability that the circuit switches when the value of $k_b$ is 0 meaning that without leakage in production, the circuit cannot switch. Figure 6.31, on the other hand, shows that the circuit switches ON after 10,000 seconds with a probability of about 8 percent for a low environmental signal and with a probability of about 70 percent for a high environmental signal. This difference is desirable as there should be a markable difference in how the circuit responds

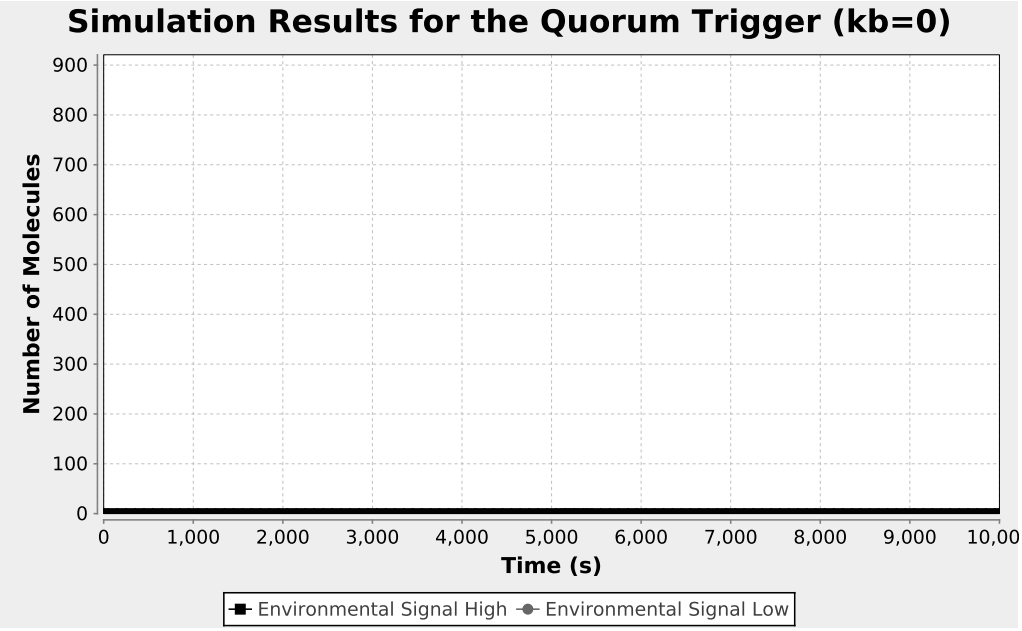



**Figure 6.27**: iSSA simulation results for the quorum trigger when the basal rate of production, $k_b$, is set to 0. This plot shows that the circuit is unable to switch ON without some leakage in production.

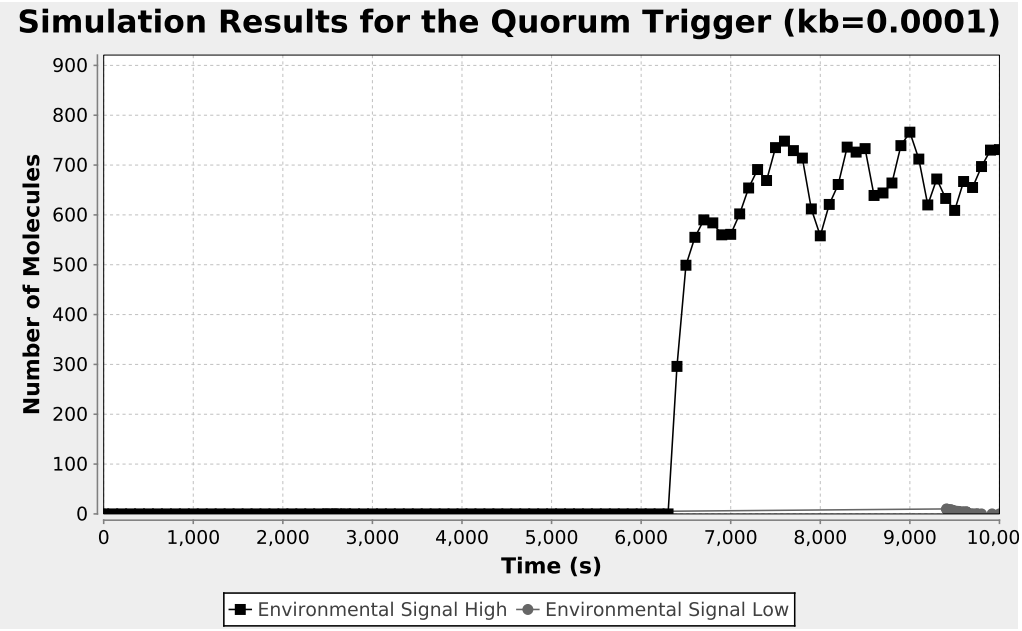



**Figure 6.28**: iSSA simulation results for the quorum trigger when the basal rate of production, $k_b$, is set to 0.0001. This plot shows that the circuit only switches ON when the environmental signal is high for a low basal rate.

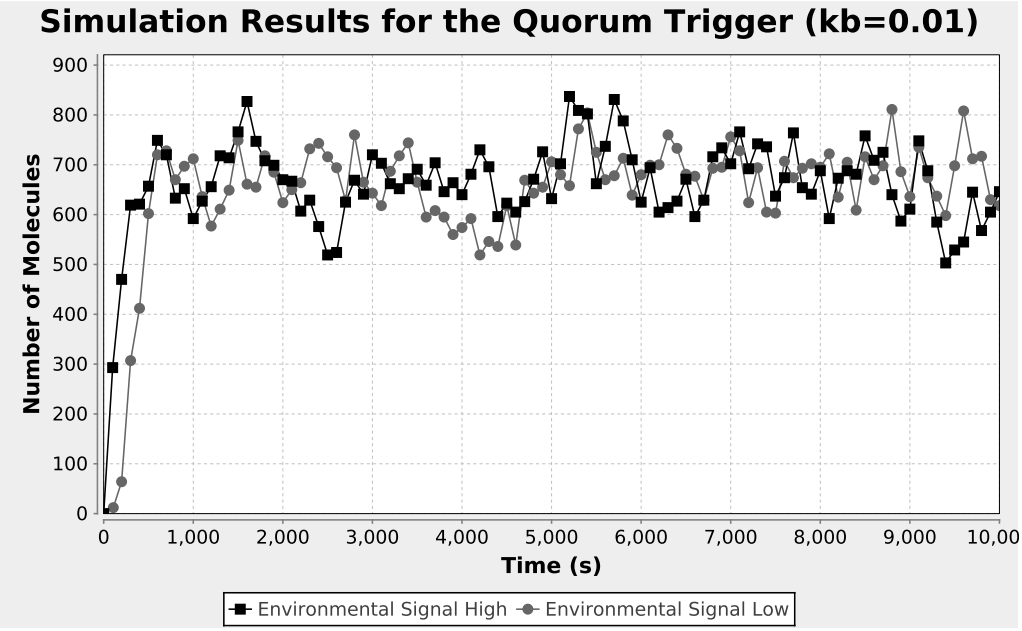



**Figure 6.29**: iSSA simulation results for the quorum trigger when the basal rate of production, $k_b$, is set to 0.01. This plot shows that the circuit always switches ON due to the basal rate being too high.

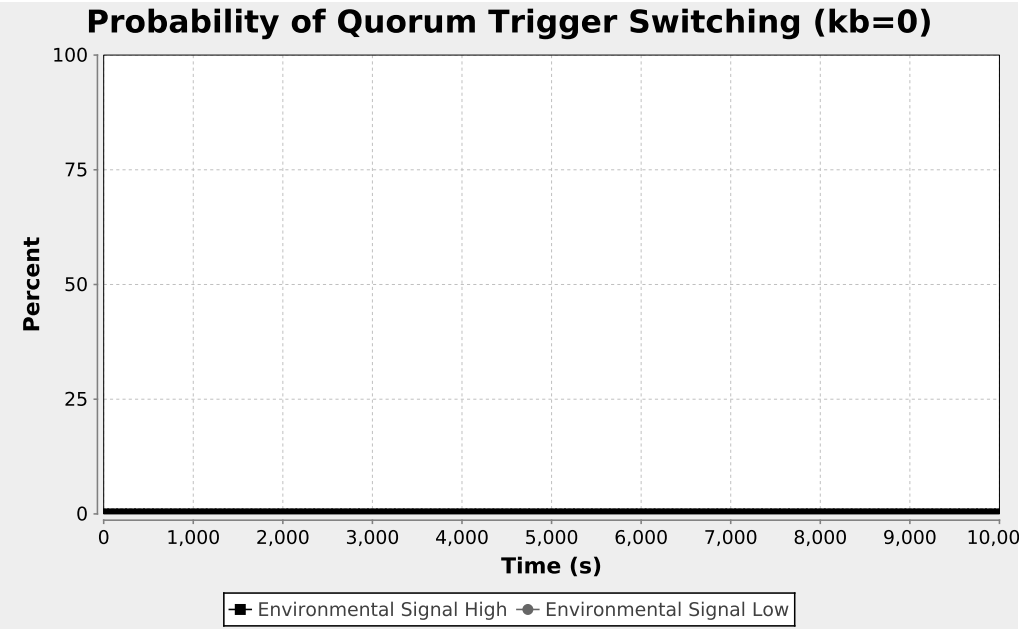



**Figure 6.30**: Stochastic model checking results for the quorum trigger when the basal rate of production, $k_b$, is set to 0. These results confirm that the quorum trigger is unable to switch ON when the basal rate is 0.

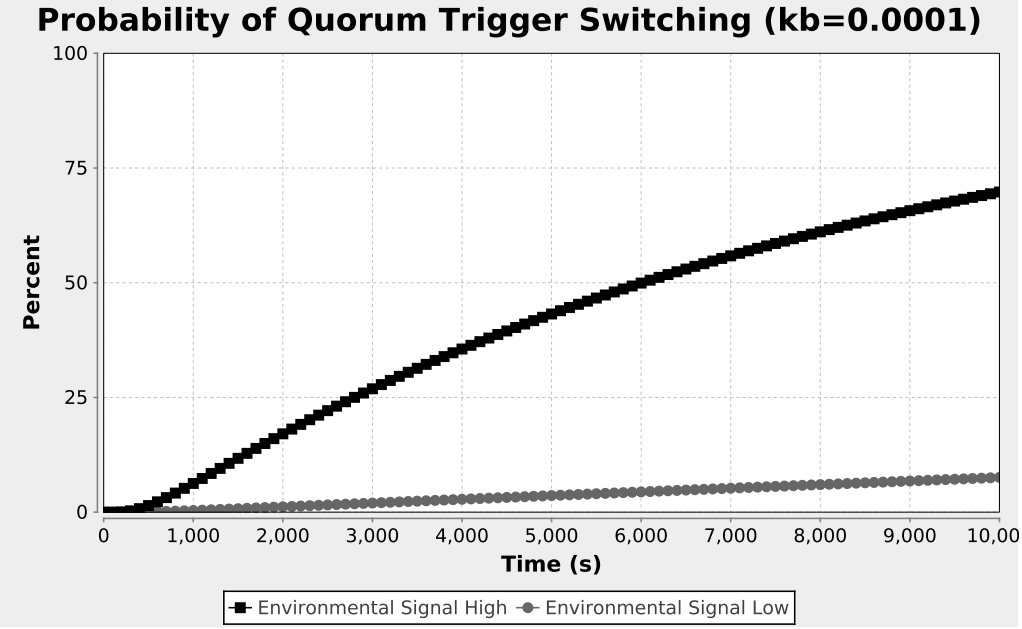



**Figure 6.31**: Stochastic model checking results for the quorum trigger when the basal rate of production, $k_b$, is set to 0.0001. These results confirm that the quorum trigger is only able to switch ON when the environmental signal is high for a median value of the basal rate.

in the presence of the environmental signal. Confirming the iSSA analysis, having a basal rate that is 0.01 causes the circuit to switch ON with 100 percent probability regardless of the value of the environmental signal as is seen in Figure 6.32. The stochastic model checking results are obtained with run-times of 2 seconds when $k_b$ is 0, 20 seconds when $k_b$ is 0.0001, and 20 seconds when $k_b$ is 0.01. These analyses show that a designer of this circuit would need to carefully tune the basal rate of production to be a value somewhere in the neighborhood of 0.0001 in order to ensure that the circuit does not switch ON too early but also does switch ON in a medium where the environmental signal is present.

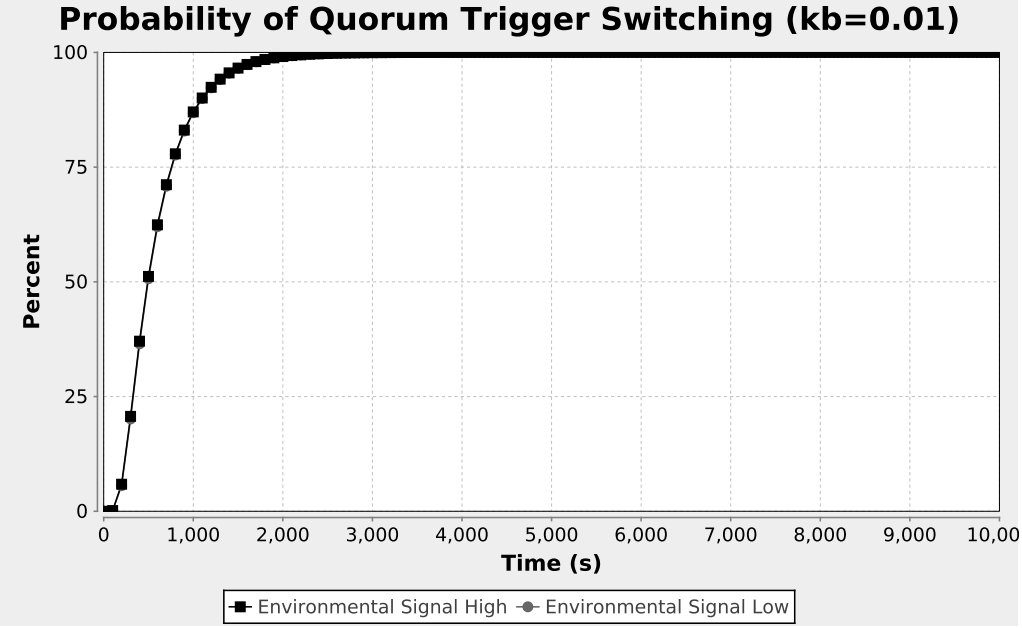



**Figure 6.32**: Stochastic model checking results for the quorum trigger when the basal rate of production, $k_b$, is set to 0.01. These results confirm that the quorum trigger always switches ON if the basal rate is too high.

# CHAPTER 7

# CONCLUSIONS

Synthetic biology has the potential to allow researchers and scientists to design biological systems to solve a variety of problems. These problems include, among other things, the clean up of toxic materials, the production of medicines and biofuels, and the destruction of tumor cells. However, in order to design these systems, efficient methods to perform design space exploration through analysis and verification of computational models are necessary. This dissertation proposes one such methodology that can greatly aid synthetic biologists in the design process. This chapter concludes the dissertation by giving a summery in Section 7.1 and presenting possible future work and extensions in Section 7.2.

## 7.1 Summary

This dissertation presents a methodology for performing design space exploration of synthetic genetic circuits in order to allow researchers to construct circuits that behave as intended and that are more robust to external noise and varying environments. This methodology involves utilizing iSSA and stochastic model checking to analyze models of genetic circuits. By performing these analyses in tandem, different design and parameter choices of a circuit can be efficiently analyzed and considered.

The iSSA delivers information about a genetic circuit that can often be hidden when using other simulation methods. By performing simulations in small time increments, it is capable of capturing important stochastic events that lead to the circuit exhibiting its "typical" behavior. The method itself can also be tuned to produce slightly different results by changing how the starting states in each increment are selected, how the time increment is calculated, and what information is stored and ultimately returned by the simulation algorithm. Additionally, although the iSSA may provide a smoothed simulation trace, it is still a stochastic simulation algorithm and has the potential to return different results from repeat executions, particularly when simulating oscillating circuits or circuits

with bifurcating sample paths. Due to these considerations, the iSSA by itself may not allow a researcher to obtain a full understanding of the behavior of a genetic circuit.

A method that can be used in conjunction with iSSA to improve the genetic circuit design process is stochastic model checking. This method translates the infinite state space of a genetic circuit into a finite state logical representation. This translation requires that a user supply a level set for each interesting species in the model which is used to discretize each species' state space. The resulting state space is then translated into a CTMC using both an operator site reduction and an amplified degradation rate abstraction to compute the rates of moving from one state to the next. This CTMC can be analyzed using stochastic model checking via steady state or transient Markov chain analysis. However, in order to determine the likelihood of a certain event or condition occurring in the system, CSL properties of the genetic circuit can be specified and checked while the stochastic model checking is performed.

When utilized together in the proposed methodology, these methods can be used to simulate a genetic circuit with the iSSA, to generate a CSL property that captures the observed "typical" behavior or a rare behavior, and then to apply stochastic model checking to the circuit yielding the likelihood of observing typical and rare behaviors as shown in Figure 7.1. If the desired behavior is not observed with a large enough probability, then the designer can alter elements of the circuit or parameters in the system to refine the genetic circuit. This dissertation has applied the design space exploration methodology to several examples of genetic oscillators and state holding gates allowing different design and parameter choices to be efficiently analyzed and considered for each model.

## 7.2 Future Work

Despite the utility of the proposed design space exploration methodology, there are still many ways that it can be improved. This section enumerates several possible areas of this work that could use improvement and describes some potential approaches to undertaking these research investigations.

### 7.2.1 Level Selection

Currently, in order to select good levels, a user performs a small number of simulation runs to determine a range of values of interest. The user then applies this information to select a number of levels in which to divide this range uniformly. Adding more levels can greatly improve the accuracy of the algorithm, but it increases the size of the state

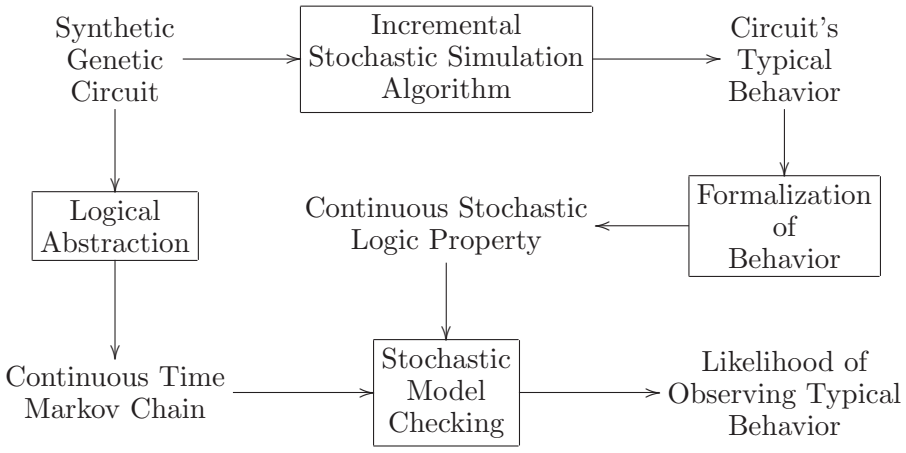



**Figure 7.1**: An example work flow of using iSSA, logical abstraction, and stochastic model checking to determine the likelihood of a system exhibiting its typical behavior.

space leading to longer analysis time. Conversely, fewer levels lead to less accurate results but speed up analysis time. One way to improve the level selection process could be to use a nonuniform choice of levels. For example, a user could leverage the iSSA to help by showing a typical evolution of the interesting species over time allowing a user to select more representative levels nonuniformly. However, automating the level selection process may prove itself to be more fruitful as it may still be difficult for a user to know where the levels should be placed. One idea for automating this process would be to select levels that approximate the transition rate function by analyzing it to determine inflection points. This way, more levels would be placed near species values that cause the evolution of a species count to slow down requiring more resolution and fewer levels would be placed at points where species values are changing rapidly and less resolution is needed.

### 7.2.2 Automatic Property Generation

Currently, the proposed methodology requires a user to analyze the resulting trace produced by the iSSA by hand in order to generate a property that captures the circuits typical behavior. A big improvement to this methodology would be the development of a process that could automatically analyze a trace and produce a CSL property that represents the behavior of the trace. Additionally, this method could possibly be tuned to produce a property that captures behaviors not observed in the analyzed trace. This way, the process may also be able to automatically produce a property that captures rare

behavior or at least would be able to be used to compute the probability that the circuit does not behave as expected.

### 7.2.3 Timed Events

Models of genetic circuits often allow for timed events where a species value can be changed at a predetermined time in simulation. Indeed, some of the simulation analyses presented in this dissertation utilize timed events to change the inputs of genetic circuits. These events are useful in modeling systems where the environment changes due to the introduction of some new chemical species. The CTMC that is generated from the conversion process could be extended to include decision transitions for these environment changes using a *Markov decision process* (MDP) [27]. The decision transitions in this model would not have a rate associated with them because the environment is not governed by the circuit and changes could come at anytime with unknown probabilities.

### 7.2.4 Partial Order Reduction

The efficiency of the stochastic model checking is also greatly impacted by the size of the model's state space. One way to deal with this problem is to apply partial order reduction to the CTMC to try to eliminate uninteresting intermediate states in the state graph. Previous work of applying partial order reduction to MDPs has proven useful; however, extending it to CTMCs in general may also yield good results [38]. Partial order reduction has the potential to cause the Markov analysis to perform its computations more quickly but could lead to a less accurate calculation of the probability when some states are collapsed that alter the rate that probability moves to absorbing states.

### 7.2.5 More Case Studies

This dissertation primarily focuses on stochastic analysis of synthetic genetic circuits, particularly genetic oscillators and state holding circuits. However, this methodology can be applied to any biological system that can be represented as a chemical reaction network because at its core, it deals with species and chemical reactions. Other systems that can lend themselves to this type of analysis include signal transduction pathways and metabolic networks. Such case studies could provide more insight into the behavior of these types of systems and serve designers in the construction of more robust systems.

# REFERENCES


[1] Abdulla, P., Baier, C., Iyer, P., and Jonsson, B. Reasoning about probabilistic lossy channel systems. *Lecture Notes in Computer Science 1877* (2000).

[2] Abdulla, P. A., Bertrand, N., Rabinovich, A., and Schnoebelen, P. Verification of probabilistic systems with faulty communication. *Inf. Comput. 202* (November 2005), 141–165.

[3] Abdulla, P. A., Henda, N. B., and Mayr, R. Verifying infinite Markov chains with a finite attractor or the global coarseness property. In *Proc. LICS '05-21th IEEE Int. Symp. on Logic in Computer Science* (2005), pp. 127–136.

[4] Anderson, J. C., Clarke, E. J., and Arkin, A. P. Environmentally controlled invasion of cancer cells by engineering bacteria. *J. Mol. Biol. 355* (2006), 619–627.

[5] Antoniotti, M., Policriti, A., Ugel, N., and Mishra, B. Model building and model checking for biochemical processes. *Cell Biochemistry and Biophysics 38* (2003), 2003.

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

[7] Atsumi, S., and Liao, J. C. Metabolic engineering for advanced biofuels production from escherichia coli. *Current Opinion in Biotechnology 19*, 5 (2008), 414 – 419. Tissue, cell and pathway engineering.

[8] Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. Model-checking continuous-time Markov chains. *ACM Trans. Comput. Logic 1* (July 2000), 162–170.

[9] Bobbio, A., and Trivedi, K. An aggregation technique for the transient analysis of stiff Markov chains. *IEEE Transactions on Computers 35* (1986), 803–814.

[10] Burrage, K., Hegland, M., Macnamara, S., and Sidje, R. A krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In *Mam 2006 : Markov Anniversary Meeting: an international conference to celebrate the 150th anniversary of the birth of A.A. Markov* (Charleston, South Carolina, 2006), A. N. Langville and W. J. Stewart, Eds., Boston Books, pp. 21–38.

[11] Calder, M., Vyshemirsky, V., Gilbert, D., and Orton, R. Analysis of signalling pathways using continuous time Markov chains. *Transactions on Computational Systems Biology 4220* (2006), 44–67.

[12] Canton, B., Labno, A., and Endy, D. Refinement and standardization of synthetic biological parts and devices. *Nature Biotechnology 26* (2008), 787–793.

[13] Cao, Y., Gillespie, D. T., and Petzold, L. R. The slow-scale stochastic simulation algorithm. *The Journal of Chemical Physics 122*, 1 (January 2005).

[14] Cao, Y., Gillespie, D. T., and Petzold, L. R. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics 124* (2006).

[15] Cases, I., and de Lorenzo, V. Genetically modified organisms for the environment: stories of success and failure and what we have learned from them. *International Microbiology 8* (2005), 213–222.

[16] Ciocchetta, F., Degasperi, A., Hillston, J., and Calder, M. Some investigations concerning the ctmc and the ode model derived from bio-pepa. *Electronic Notes in Theoretical Computer Science 229*, 1 (2009), 145 – 163. Proceedings of the Second Workshop From Biology to Concurrency and Back (FBTC 2008).

[17] Clarke, E. M., Emerson, E. A., and Sistla, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems 8* (1986), 244–263.

[18] Crick, F. H. Central Dogma of Molecular Biology. *Nature 227*, 5258 (Aug. 1970), 561–563.

[19] D'argenio, P. R., Jensen, H. E., and Larsen, K. G. Reachability analysis of probabilistic systems by successive refinements. In *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Veri (PAPM/PROBMIV'01), volume 2165 of LNCS* (2001), Springer, pp. 39–56.

[20] de Alfaro, L., and Roy, P. Magnifying-lens abstraction for Markov decision processes. In *Proceedings of the 19th International Conference on Computer Aided Verification* (Berlin, Heidelberg, 2007), CAV'07, Springer-Verlag, pp. 325–338.

[21] Didier, F., Henzinger, T. A., Mateescu, M., and Wolf, V. Approximation of event probabilities in noisy cellular processes. In *Proc. of CMSB* (2009).

[22] Elowitz, M., and Leibler, S. A synthetic oscillatory network of transcriptional regulators. *Nature 403*, 6767 (2000), 335–338.

[23] Endy, D. Foundations for engineering biology. *Nature 438* (2005), 449–453.

[24] Esparza, J., and Etessami, K. Verifying probabilistic procedural programs. In *Foundations of Software Technology and Theoretical Computer Science* (2004), pp. 16–31.

[25] Esparza, J., Kucera, A., and Mayr, R. Model checking probabilistic pushdown automata. In *Proceedings of LICS 2004* (2004), IEEE, pp. 12–21.

[26] Etessami, K., and Yannakakis, M. Algorithmic verification of recursive probabilistic state machines. In *Proc. 11th TACAS* (2005), Springer, pp. 253–270.

[27] Feinberg, E., and Shwartz, A., Eds. *Handbook of Markov Decision Processes - Methods and Applications*. Kluwer International Series, 2002.

[28] Friedman, N., Linial, M., Nachman, I., and Pe'er, D. Using bayesian networks to analyze expression data. *Journal of Computational Biology 7*, 3–4 (2000), 601–620.

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

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

[31] Gillespie, D. The chemical langevin equation. *The Journal of Chemical Physics 113*, 1 (2000).

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

[33] Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry 81*, 25 (1977), 2340–2361.

[34] Gillespie, D. T. *Markov Processes: An Introduction for Physical Scientists*. Academic Press, 1992.

[35] Gillespie, D. T. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics 115*, 4 (2001), 1716–1733.

[36] Gillespie, D. T., and Petzold, L. R. Tau leaping. *The Journal of Chemical Physics 119* (2003), 8229–8234.

[37] Gillespie, D. T., Roh, M., and Petzold, L. R. Refining the weighted stochastic simulation algorithm. *J Chem Phys 130*, 17 (2009).

[38] Grösser, M., and Baier, C. Partial order reduction for Markov decision processes: A survey. In *FMCO* (2005), pp. 408–427.

[39] Hahn, E. M., Hermanns, H., Wachter, B., and Zhang, L. Time-bounded model checking of infinite-state continuous-time Markov chains. *Fundam. Inf. 95* (January 2009), 129–155.

[40] Hansson, H., and Jonsson, B. A framework for reasoning about time and reliability. In *Real Time Systems Symposium, 1989., Proceedings.* (1989), IEEE, pp. 102–111.

[41] Hansson, H., and Jonsson, B. A logic for reasoning about time and reliability. *Formal Aspects of Computing 6*, 5 (1994), 512–535.

[42] Heath, J., Kwiatkowska, M., Norman, G., Parker, D., and Tymchyshyn, O. Probabilistic model checking of complex biological pathways. *Theor. Comput. Sci. 391*, 3 (Feb. 2008), 239–257.

[43] Henzinger, T., Mateescu, M., and Wolf, V. Sliding window abstraction for infinite Markov chains. In *Computer Aided Verification*, A. Bouajjani and O. Maler, Eds., vol. 5643 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, pp. 337–352.

[44] Henzinger, T. A., Mikeev, L., Mateescu, M., and Wolf, V. Hybrid numerical solution of the chemical master equation. In *Proceedings of the 8th International Conference on Computational Methods in Systems Biology* (New York, NY, USA, 2010), CMSB '10, ACM, pp. 55–65.

[45] Hinton, A., Kwiatkowska, M., Norman, G., and Parker, D. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)* (2006), H. Hermanns and J. Palsberg, Eds., vol. 3920 of *LNCS*, Springer, pp. 441–444.

[46] Hucka, M., et al. The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinform. 19*, 4 (2003), 524–531.

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

[48] Katoen, J., Klink, D., Leucker, M., and Wolf, V. Three-valued abstraction for continuous-time Markov chains. In *Proc. 19th Int. Conf. on Comput. Aided Verification (CAV'07)* (2007).

[49] Kauffman, S. A. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol. 22* (1969), 437–467.

[50] Kauffman, S. A. *The Origins of Order: Self-Organization and Selection in Evolution*, 1 ed. Oxford University Press, USA, June 1993.

[51] Kauffman, S. A., Shymko, R. M., and Trabert, K. Control of sequential compartment formation in Drosophila. *Science 199* (1978), 259–270.

[52] Kucera, A. Methods for quantitative analysis of probabilistic pushdown automata. *Electron. Notes Theor. Comput. Sci. 149* (February 2006), 3–15.

[53] Kuwahara, H. *Model Abstraction and Temporal Behavior Analysis of Genetic Regulatory Networks*. PhD thesis, U. of Utah, 2007.

[54] Kuwahara, H., Madsen, C., Mura, I., Myers, C., Tejada, A., and Winstead, C. Efficient stochastic simulation to analyze targeted properties of biological systems. In *Stochastic Control*, C. Myers, Ed. Sciyo, 2010, pp. 505–532.

[55] Kuwahara, H., and Mura, I. An efficient and exact stochastic simulation method to analyze rare events in biochemical systems. *The Journal of Chemical Physics 129*, 16 (2008).

[56] Kuwahara, H., Myers, C., Barker, N., Samoilov, M., and Arkin, A. Automated abstraction methodology for genetic regulatory networks. *Trans. Comp. Syst. Biol. VI* (2006), 150–175.

[57] Kwiatkowska, M., Norman, G., and Parker, D. Game-based abstraction for Markov decision processes. In *Proc. of QEST: Quantitative Evaluation of Systems* (2006), IEEE Computer Society, pp. 157–166.

[58] Kwiatkowska, M., Norman, G., and Parker, D. Stochastic model checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)* (2007), M. Bernardo and J. Hillston, Eds., vol. 4486 of *LNCS (Tutorial Volume)*, Springer, pp. 220–270.

[59] Liu, B., Hsu, D., and Thiagarajan, P. S. Probabilistic approximations of odes based bio-pathway dynamics. *Theor. Comput. Sci. 412*, 21 (May 2011), 2188–2206.

[60] MacQueen, J. B. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley 1* (1967), 281–297.

[61] McAdams, H. H., and Arkin, A. Genetic regulation at the nanomolar scale: it's a noisy business. *Trends Genet. 15*, 2 (1999), 65–69.

[62] Mikeev, L., Sandmann, W., and Wolf, V. Efficient calculation of rare event probabilities in Markovian queueing networks. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools* (2011), VALUETOOLS '11.

[63] Munsky, B., and Khammash, M. The finite state projection algorithm for the solution of the chemical master equation. *The Journal of Chemical Physics 124*, 4 (Jan 2006).

[64] Myers, C. J. *Engineering Genetic Circuits*. Chapman and Hall/CRC, 2009.

[65] Myers, C. J., Barker, N., Jones, K., Kuwahara, H., Madsen, C., and Nguyen, N.-P. D. iBioSim: a tool for the analysis and design of genetic circuits. *Bioinform. 25*, 21 (2009), 2848–2849.

[66] Nguyen, N. Design and analysis of genetic circuits. Master's thesis, U. of Utah, 2008.

[67] Nguyen, N., Kuwahara, H., Myers, C., and Keener, J. The design of a genetic muller c-element. In *The 13th IEEE International Symposium on Asynchronous Circuits and Systems* (Mar. 2007).

[68] Ognjanovic, Z. Discrete linear-time probabilistic logics: Completeness, decidability and complexity. *J. Log. Comput. 16*, 2 (2006), 257–285.

[69] Pearl, J. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine* (Aug. 1985), pp. 329–334.

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

[71] Rabinovich, A. Quantitative analysis of probabilistic lossy channel systems. *Inf. Comput. 204* (May 2006), 713–740.

[72] Remke, A. K. I. *Model Checking Structured Infinite Markov Chains*. PhD thesis, Univ. of Twente, Zutphen, June 2008.

[73] Ro, D.-K., Paradise, E. M., Ouellet, M., Fisher, K. J., Newman, K. L., Ndungu, J. M., Ho, K. A., Eachus, R. A., Ham, T. S., Kirby, J., Chang, M. C. Y., Withers, S. T., Shiba, Y., Sarpong, R., and Keasling, J. D. Production of the antimalarial drug percursor artemisinic acid in engineered yeast. *Nature 440* (2006).

[74] Segel, Lee A., and Slemrod, Marshall. The quasi-steady-state assumption: A case study in perturbation. *SIAM Review 31*, 3 (sep 1989), 446–477.

[75] Slepoy, A., Thompson, A. P., and Plimpton, S. J. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics 128*, 20 (2008), 205101.

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

[77] Stricker, J., Cookson, S., Bennett, M., Mather, W., Tsimring, L., and Hasty, J. A fast, robust and tunable synthetic gene oscillator. *Nature 456* (Oct 2008), 516–519.

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

[79] Thomas, R. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology 42*, 3 (Dec. 1973), 563–585.

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

[81] Vardi, M. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *5th Int'l Workshop on Formal Methods for Real-Time and Probabilistic Systems* (1999), J. Katoen, Ed., Lecture Notes in Computer Science 1601, Springer-Verlag, pp. 265–276.

[82] Vilar, J. M. G., Kueh, H. Y., Barkai, N., and Leibler, S. Mechanisms of noise-resistance in genetic oscillators. *Proc. of the National Academy of Sciences 99*, 9 (2002), 5988–5992.

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

[84] Winstead, C., Madsen, C., and Myers, C. J. iSSA: An incremental stochastic simulation algorithm for genetic circuits. In *International Symposium on Circuits and Systems (ISCAS)* (2010), IEEE, pp. 553–556.

[85] Younes, H., Kwiatkowska, M., Norman, G., and Parker, D. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT) 8* (2006), 216–228.