DESIGN AND ANALYSIS OF GENETIC CIRCUITS

by

Nam-phuong D. Nguyen

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

Master of Science

in

Computer Science

School of Computing

The University of Utah

May 2009

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Nam-phuong D. Nguyen

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Chris J. Myers

James P. Keener

Konrad Slind

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Nam-phuong D. Nguyen in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Chris J. Myers
Chair: Supervisory Committee

Approved for the Major Department

Martin Berzins
Chair/Director

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

Electronic Design Automation (EDA) tools have facilitated the design of ever more complex integrated circuits each year. Synthetic biology would also benefit from the development of *Genetic Design Automation* (GDA) tools. Existing GDA tools require biologists to design genetic circuits at the molecular level, roughly equivalent to designing electronic circuits at the layout level. Analysis of these circuits is also performed at this very low level. This thesis presents a first step at developing a GDA tool that supports higher levels of abstraction. In particular, this thesis describes the *Genetic Circuit Model* (GCM), a graphical specification language from which molecular descriptions can be synthesized. The GCM has several advantages. The input is tightly controlled through the use of an editor, limiting the possibility of user error. The representation of the genetic circuit is much more compact than using the *System Biology Markup Language* (SBML), the standard form for representing genetics circuits. The GCM can be automatically translated into SBML, allowing GCM's to be easily simulated across multiple different simulators. The GCM to SBML translation process is targeted in such a way that the resulting output can be easily abstracted to allow for efficient simulation.

To evaluate and test the GCM, this thesis presents a case study of the design of a genetic Muller C-element, a gate often used in asynchronous design. Three different genetic Muller C-elements are designed and analyzed. The utility of the GCM is demonstrated as it allows for efficient analysis of the Muller C-elements. The results of the simulations show that logically equivalent circuits can have different behaviors. In particular, a speed independent Muller C-element does not necessary imply that the gate is more robust than a non-speed independent gate. Design priciples gathered from the simulations are that dual-rail outputs are essential, high gene count increases robustness, cooperativity greater than one is necessary, repression needs to be strong, and decay rates must be balanced for high robustness and low switching time. One potential application of the genetic Muller C-element is determining when to start the invasion of cancer cells. The two input signals are an environmental signal, and a communication signal. Using these signals, the bacteria colony can correctly reach consensus on when to begin the invasion.

One interesting result is that noise is necessary in correctly switching into the invasion state.

To my family who provided me with cheap housing, my professors who helped me find my path, and my friends who kept me sane.

CONTENTS

ABSTRACT . . . . . **iv**

LIST OF FIGURES . . . . . **ix**

LIST OF TABLES . . . . . **xii**

CHAPTERS

1. INTRODUCTION **1**

1.1 Related Work 2
1.2 Contributions 4
1.3 Organization 5

2. GENETIC CIRCUITS MODELS **7**

2.1 Genetic Circuits 7
2.2 SBML 10
2.3 Genetic Circuit Model 12
2.4 SBML Generation 18

3. GENETIC MULLER C-ELEMENT **28**

3.1 A Genetic Muller C-element 28
3.1.1 Majority Gate Design 29
3.1.2 Toggle Switch Design 33
3.1.3 Speed Independent Design 40
3.2 Mathematical Analysis 44
3.2.1 Majority Gate Design 45
3.2.2 Toggle Switch Design 49
3.2.3 Speed Independent Design 50
3.3 Failure Rate Analysis Using Stochastic Simulation 54
3.4 Parameter Variation Analysis 58
3.4.1 Changing Gene Copies 64
3.4.2 Changing Cooperativity 68
3.4.3 Changing Repression Strength 68
3.4.4 Changing Decay Rates 70
3.4.5 Changing Production and Degradation Rates 80
3.5 Design Principles 81
3.6 Application 85

4. CONCLUSION **88**
4.1 Summary of Work 88
4.2 Future Work 89
4.2.1 Hierarchy 90
4.2.2 Technology Mapping 90
4.2.3 Synthesizing the Genetic Muller C-element 91
APPENDIX: GCM SPECIFICATION LANGUAGE **93**
REFERENCES **98**

LIST OF FIGURES

1.1 The iBioSim tool flow. 5
2.1 Phage λ circuit. 8
2.2 Phage λ (a) simulation result and (b) digital representation. 9
2.3 Species and reactions from the SBML model 11
2.4 Graphical model of circuit. 12
2.5 Example using same promoter 15
2.6 Example using different promoter. 15
2.7 Example with no biochemical influences. 16
2.8 Example with biochemical influences. 17
2.9 Degradation of CI and CII. 19
2.10 CI and CII's open complex formation reactions. 22
2.11 Dimerization reaction for CI. 23
2.12 Complex formation of A and B for Figure 2.8. 24
2.13 Reactions for CI dimer repression of CII. 25
2.14 Reactions for CII activation of production of CI. 27
3.1 Schematic symbol for a Muller C-element. 29
3.2 Genetic majority gate designs. 31
3.3 SBML (a) before abstraction and (b) after abstraction for the genetic majority C-element. 32
3.4 Results from simulation of the majority genetic Muller C-element. 33
3.5 Genetic toggle switch's (a) genetic design and (b) experimental results 34
3.6 Genetic toggle gate designs. 36
3.7 Genetic circuit diagram for the genetic toggle C-element. 37
3.8 SBML for the genetic toggle C-element. 38
3.9 Simulation results for the genetic toggle Muller C-element. 39
3.10 Genetic SI C-element designs. 41
3.11 SBML for the genetic SI. 42
3.12 Simulation results for the SI genetic Muller C-element. 43

3.13 Phase plane diagram for the genetic majority C-element when inputs are low. 46
3.14 Phase plane diagram for the genetic majority C-element when inputs are mixed. 47
3.15 Phase plane diagram for the genetic majority C-element when inputs are high. 48
3.16 Phase plane diagram for the genetic toggle C-element when inputs are low. 50
3.17 Phase plane diagram for the genetic toggle C-element when inputs are mixed. 51
3.18 Phase plane diagram for the genetic toggle C-element when inputs are high. 52
3.19 Phase plane diagram for the genetic SI C-element when inputs are low. . . . 54
3.20 Phase plane diagram for the genetic SI C-element when inputs are mixed. . 55
3.21 Phase plane diagram for the genetic SI C-element when inputs are high. . . . 56
3.22 Probability of failure for single-rail versus dual-rail for the majority gate . . 59
3.23 Probability of failure for single-rail versus dual-rail for the toggle gate 60
3.24 Probability of failure for single-rail versus dual-rail for the SI gate 61
3.25 Probability of the C-element state to change from low to high 62
3.26 Probability of the C-element state to change from high to low 63
3.27 The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the number of gene copies in a circuit being varied between 1 to 5. 65
3.28 The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the number of gene copies in a circuit being varied between 1 to 5. 65
3.29 The nullclines of the toggle gate when both inputs are mixed, and the gene copy is (a) 1, and (b) 2 66
3.30 Switching time of the C-element, varying the gene count. 67
3.31 The effects of changing cooperativity on the production rate versus the number of repressor molecules 69
3.32 The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the cooperativity in a circuit being varied between 1 to 5. 69
3.33 The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the the cooperativity in a circuit being varied between 1 to 5. 70
3.34 The nullclines of the toggle gate when both inputs are mixed, and the cooperativity is (a) 1, and (b) 2 71
3.35 Switching time of the C-element, varying the cooperativity constant. 72

3.36 The effects of changing repression strength on the production rate versus the number of repressor molecules 73
3.37 The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the repression constant being varied between .1 to 1. . 73
3.38 The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the repression constant being varied between .1 to 1. 74
3.39 The nullclines of the (a) SI gate and (b) toggle gate when both inputs are mixed, and K_R is 0.1. 75
3.40 Switching time of the C-element, varying the repression strength. 76
3.41 The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the decay rate being varied between 1.875e-3 to 0.03. 77
3.42 The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the decay rate being varied between 1.875e-3 to 0.03. 77
3.43 The nullclines of the toggle gate when both inputs are mixed, and k_d is (a) 0.005, (b) 0.0075 78
3.44 Switching time of the C-element, varying the decay rate. 79
3.45 A schematic for a digital Muller C-element with weak feedback. 80
3.46 The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the ratio of production to decay being varied between 0.5 to 4.0. 81
3.47 The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, the ratio of production to decay being varied between 0.5 to 4.0. 82
3.48 Switching time of the C-element, varying the production to decay rate. 83
3.49 The nullclines of the SI gate when both inputs are mixed, and the constant multiplier is (a) 0.5, (b) 2.0. 84
3.50 Bacteria consensus with an error rate of switching of 0.005. 86
3.51 Bacteria consensus with an error rate of switching of 0.05. 87
3.52 Bacteria consensus with no error rate of switching. 87
4.1 A possible realization of a genetic toggle Muller C-element. 92
A.1 Part of the phage λ circuit 93
A.2 Simple example with different promoters. 94
A.3 Simple example with same promoters. 94
A.4 Simple example that behaves as a NOR. 95
A.5 Simple example that behaves as a NAND. 96
A.6 Simple example that behaves as an AND. 96
A.7 Simple example that contains global parameters. 97

LIST OF TABLES

2.1 Parameter List. 14
3.1 Truth Table for a Muller C-element. 29
A.1 Mapping of Parameter List. 97

CHAPTER 1

INTRODUCTION

Synthetic biology is the study of genetically engineering new biological pathways that can potentially change the behavior of organisms in useful ways. Synthetic biology can help us better understand how microorganisms function by allowing us to examine how a synthetic pathway behaves *in vivo* as compared with *in silico* [34]. By studying the differences, scientists can understand how designing in DNA technology differs from designing in a digital technology. There are also numerous exciting potential applications. The Gates foundation is funding research on the design of pathways for production of artemensin, an antimalarial precursor protein [30]. This may lead to cheaper antimalarial drugs. Scientists have also been working on modifying bacteria to metabolize toxic chemicals [8, 9]. This may result in a cost effective method to clean up hazardous waste sites. Finally, scientists are designing bacteria to detect and kill tumor cells [4]. Crucial to the success of synthetic biology is the construction of new *genetic circuits*.

Much like digital circuits drive electronic processes, genetic circuits drive the biological processes of the cell. The goal of synthetic biology is to engineer genetic circuits to perform specificed tasks [33]. This approach differs from traditional biology as synthetic biology focuses on the application of engineering techniques to build an artificial biological system in organisms rather than study natural existing systems in organisms. This process includes modeling, abstraction, simulation, verification, and building modular components. Mathematical models are created to understand the underlying design principles. The models are abstracted to make the modeling more tractable. The models are simulated to test predictability and reliability. The models can be conceptualized into modular components. The components are then fabricated and tested to verify the correctness of both the models and the components. Once the components behave reliably, components can be combined with other components to build more complex systems.

One can draw many parallels between digital circuit design and genetic circuit design. Until the advent of *Electronic Design Automation* (EDA), digital circuits were designed by hand and had to be manually laid out. Once EDA tools were developed, they facilitated the design of ever more complex integrated circuits each year. Synthetic biology would also benefit from the development of *Genetic Design Automation* (GDA) tools. Existing GDA tools require biologists to design genetic circuits at the molecular level, roughly equivalent to designing electronic circuits at the layout level. Analysis of these circuits is also performed at this very low level. This thesis presents a first step at developing a GDA tool that supports higher levels of abstraction. In particular, this thesis describes a graphical specification language from which molecular descriptions can be synthesized. These descriptions can then be abstracted to facilitate efficient analysis.

1.1 Related Work

One of the first steps in the development of synthetic biology began with the understanding of the of the *lac operon*. Monod and Jacob showed that there was a mathematical logic in the regulation of genes [22]. This resulted in a plausible mathematical model for gene regulation and protein production. Further experiments discovered that a biological phenomenon known as cooperativity plays an important role in gene regulation. Cooperativity can be modeled with a nonlinear transfer function. A non-linear transfer function allows for switch-like behavior and is necessary in the development of *sequential logic gates*. Sequential logic gates are logic gates with memory, i.e. the output depends on both the input and the history of the input. With sequential logic gates, one can build state machines. One sequential logic gate developed is the genetic toggle switch by Gardner et al. [14]. The genetic toggle switch can hold two different states, depending on the initial input. Once a stable state has been established and the input signal is removed, the toggle switch maintains the state until it receives a different input signal to switch state. Adler and Cherry analyzed the toggle switch and determined that cooperativity allows the switch to achieve bistablility [11], which is necessary for building circuits with memory.

There has been significant research in constructing *combinational logic gates* using genetic circuits. Combinational logic gates are logic gates where the output is a function of the input. Arkin and Ross showed how to build NAND, NOT, and AND gates using enzymatic reactions [5]. Elowitz and Leibler combined digital design theory with

synthetic biology to show that a three-ring inverter built from genetic technology oscillates [13]. The results shows that synthetic biology can draw upon ideas from digital design. Guet et al. analyzed randomly generated genetic networks and determined that diverse computational functions can arise by changing the connectivity of a network [20]. New advances in RNA technology has allowed scientists to create different biological gates [12, 35]. To aid scientists in the development of new genetic circuits, MIT has even created a registry of standard biological parts [2]. The goal is to have a database of interchangeable parts, a key feature in engineering. By using parts from MIT's database, and the motifs presented in previous research [23], it is possible to build a diverse library of logic gates.

With many different ways to build genetic gates, new tools and methods in simulation have become vital in the advancement of synthetic biology. Simulation allows biologists to eliminate faulty designs without having to test the design in vivo, saving both time and money. This is analogous to verification in digital design. It is much cheaper to design and simulate a chip than to fabricate and test a chip. Unlike digital circuits, however, the molecule counts in genetic circuits are extremely small, making the continuous-deterministic assumption inadequate. Stochastic effects can accumulate, and this behavior is not reflected in continuous-deterministic models. Simulation techniques in digital design do not consider the stochastic flucations that occur in genetic design. In order to capture this behavior, stochastic simulation techniques such as Gillespie's *Stochastic Simulation Algorithm* (SSA) [17, 18] are used. The SSA method, however, is very inefficient, and simulation techniques such as the Gibson and Bruck *next reaction method* [15] and *tau leaping* [16] have been developed that have much better performance.

To improve efficiency, model abstraction has also been leveraged. *Reaction based abstraction* finds patterns in the model and applies transformations to reduce the model's size by techniques such as merging reactions, removing irrelevant species and reactions, propagating constants, etc [25, 26]. Using these abstractions requires the model to be in a format recognizable by the abstraction engine.

To aid in the simulation of biological systems, a standard representation, the *Systems Biology Markup Language* (SBML) [3] is being developed. Many simulation tools have been developed to accept models in the SBML format (see for example [7, 1, 10, 32, 21]). SBML represents biological systems at the molecular level. A typical SBML model is composed of a number of chemical *species* (i.e., proteins, genes, etc.) and *reactions*

that transform these species. SBML is a very low level representation which is roughly equivalent to the layout level for electronic circuits. Designing and simulating genetic circuits at this level of detail is extremely tedious and time-consuming. Other tools, such as `BioJADE` [19], provide schemetic capture, but do not integrate with SBML nor apply abstraction. Therefore, there is a need for a higher-level representation of genetic circuits that can easily be abstracted.

1.2 Contributions

This thesis presents a first step at developing a GDA tool that supports higher levels of abstraction. One main contribution is the development of a *Genetic Circuit Model* (GCM), which is a graphical specification method that describes a genetic circuit at a level of abstraction above the molecular level and an algorithm to translate the GCM into SBML. A GCM does not include every species and reaction explicitly rather it only includes the *influences* between important *species*, so the model is more compact than an SBML representation. This makes models much more tractable for analysis and speeds up the development of models. The algorithm automatically translates the GCM into SBML using design patterns that control the form of the SBML output. This allows for automated abstraction methods, accelerating simulation time.

As a case study, the GCM is used to design several different genetic *Muller C-elements*. A robust Muller C-element would theoretically allow the design of any asynchronous finite state machine in genetic technology. This gate is vital to our long-term goal to adapt asynchronous synthesis and technology mapping tools to target a genetic circuit technology. This work is greatly aided by the GCM language. The case study shows that logically equivalent designs can have different behaviors, as one design outperforms the other gates. The case study also shows the effects of parameter variation on the robustness of the gates. The results show the design principles of building a robust Muller c-element.

The GCM language and SBML translation algorithm are integrated in our tool, `iBioSim`. This tool can be utilized to either analyze an existing natural genetic circuit or assist in the design of a new synthetic genetic circuit. Our design flow is shown in Fig. 1.1. For an existing genetic circuit, time-series experimental data must first be collected using, for example, *microarrays* to gather gene expression data. Our tool can then analyze this data to learn the connectivity in the genetic circuit using the method described in [6] and output the GCM. For the construction of a new circuit, `iBioSim` contains an editor

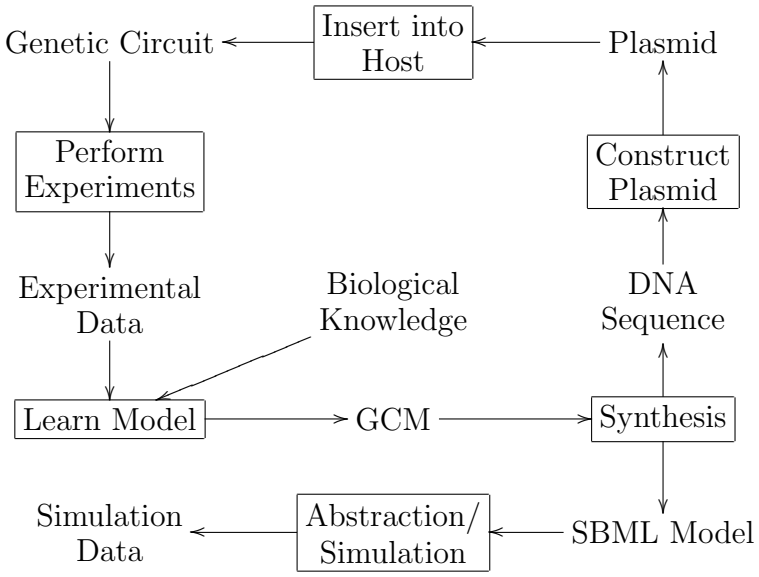


Figure 1.1. The iBioSim tool flow.

for the creation of a new GCM. The editor reduces the error rate by ensuring the user inputs only valid GCM formats. The GCM can then be translated into an SBML model that can be analyzed using any SBML compliant simulation tool. In particular, `iBioSim` incorporates an efficient temporal behavioral simulator that, whenever possible, abstracts out unnecessary details from the SBML model to substantially improve the efficiency of simulation [25]. The GCM controls the SBML output such that a maximum amount of abstraction can be applied, ensuring accelerated simulation. After simulation, the user can modify the GCM until it produces the desired simulation results.

1.3 Organization

The rest of this thesis is organized as follows:

Chapter 2 presents the GCM structure. This chapter includes a brief overview of genetic circuits, a formal description of the GCM language, and an algorithm to the translate a GCM to SBML.

Chapter 3 presents a case study using the GCM to design and analyze three different genetic Muller C-elements. While the three designs are logically equivalent, each design behaves differently. This chapter analyzes the source of these differences through null-cline analysis as well as through stochastic simulation. Parameter variation simulations

are presented to determine design principles in synthetic biology. Finally, a potential application of the genetic Muller C-element is presented in this section.

Chapter 4 summarizes our work and describes how our work can aid biologists. This chapter also presents our future goals.

CHAPTER 2

GENETIC CIRCUITS MODELS

The standard language most often used for the simulation of biochemical reaction networks is SBML, a machine readable language based on XML. SBML can be analyzed using a variety of simulators. However, one drawback of SBML is that it requires a lot of structure to model the behavior of a simple genetic circuit. Representing the same genetic circuit abstracting out low level details would be much more compact. Therefore, it is advantageous to be able to build such abstract graphical models of genetic circuits and have the model be automatically translated into SBML. This is the driving idea behind the *Genetic Circuit Model* (GCM). The GCM is a graphical specification method that describes a genetic circuit at a level of abstraction above the molecular level. A GCM does not include every species and reaction explicitly rather it only includes the *influences* between the important species. This allows more efficient design and analysis of genetic circuits.

This chapter is organized as follows. Section 2.1 gives a brief overview of genetic circuits using a small part of the phage λ decision circuit. Section 2.2 provides a description of the SBML language. Section 2.3 explains the GCM format and gives the GCM representation of our small example. Section 2.4 explains the translation from GCM to SBML.

2.1 Genetic Circuits

Genetic circuits are biological circuits constructed from DNA. Fig. 2.1 depicts a simple genetic circuit found in the phage λ virus [29]. A coding sequence of DNA includes several key regions: *genes*, *promoters*, *operator sites*, *terminators*, and *ribosome binding sites (RBS)*. Genes are regions that code for a protein. In Fig. 2.1, the genes are *cI* and *cII* which code for proteins CI and CII, respectively. Promoters are regions on the DNA to which *RNA polymerase (RNAP)* binds to start transcribing the gene. In Fig. 2.1, the promoters are P_{RE} and P_R. RNAP binds to the promoter and starts *transcription*.

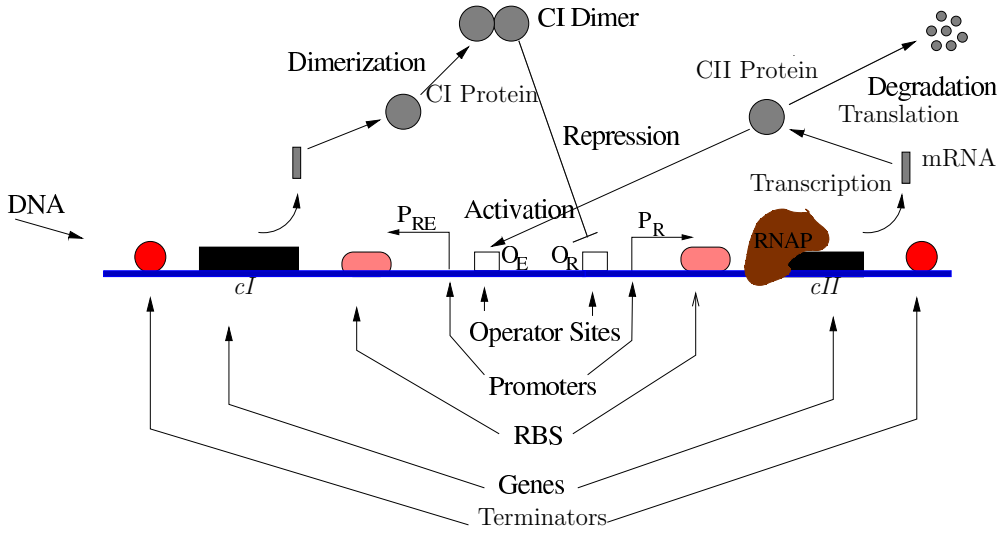
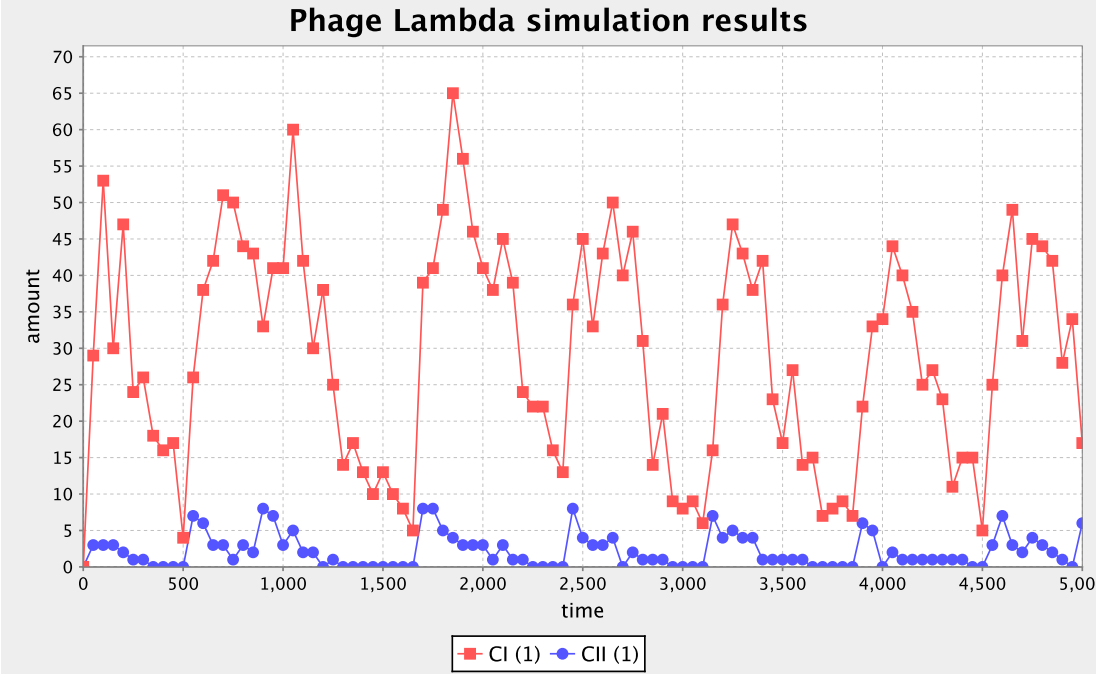


Figure 2.1. Phage λ circuit.

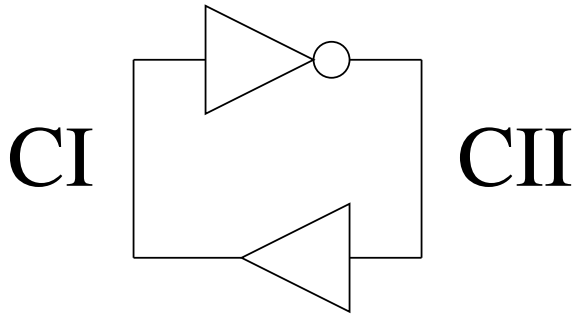
Transcription is the process in which a strand of *messenger RNA (mRNA)* is created from the DNA. Terminator sites signal the RNAP to stop transcription and unbind from the DNA strand. Operator sites are regions to which proteins known as *transcription factors* bind to increase or decrease the affinity of the promoter. The operator sites for the circuit are O_R and O_E. In Fig. 2.1, the transcription of *cI* is *activated* by CII binding to the O_R operator site, and transcription of *cII* is *repressed* by the CI *dimer* binding to the O_E operator site. The RBS is the location on the mRNA to which the ribosome binds to start *translation*. Translation is the process in which a protein is created from mRNA.

The behavior of the genetic circuit shown in Fig. 2.1 is as follows. Initially, there are no molecules of CI or CII in the host cell. In this state, transcription of *cII* is very active while transcription of *cI* is not very active as its transcription must be facilitated by a CII molecule bound to the O_E operator site. As CII molecules begin to build up in the cell, CII activates the transcription of *cI*. As CI molecules are produced, pairs of them bind together (i.e., *dimerize*) to form the CI dimer. A CI dimer can bind to the O_R operator site which represses the transcription of *cII*. Without further production of CII, the level of CII decreases due to degradation. This lowers the amount of CII available to activate CI production, so CI level also begins to decrease due to degradation. Eventually, only

small amounts of CII and CI remain, returning the circuit to its initial state and the process repeats. The simulation result is shown in Fig. 2.2(a). This process is familiar to any digital designer as a simple oscillator as shown in Fig. 2.2(b). The simulation result shows that the circuit does, indeed, oscillate. As the level of CI goes high, the level of CII starts to go high. As the level of CII rises, then the level of CI drops. This causes the level of CI to go back to high again and produces the oscillations. The simulation result is generated by simulating an SBML model of Fig. 2.1.



(a)



(b)

Figure 2.2. Phage λ (a) simulation result and (b) digital representation.

2.2 SBML

SBML has become a standard representation to model biological systems such as genetic circuits, and it is the input to many simulation tools. An SBML model may contain *function definitions*, *unit definitions*, *compartment types*, *species types*, *compartments*, *species*, *parameters*, *initial assignments*, *rules*, *constraints*, *reactions*, and *events*. This thesis focuses on an abstraction of the SBML model.

Our abstracted SBML model can be defined as a tuple $\langle S, R, St, K, V_s, A_s, O \rangle$ where:

- S is a finite set of species;
- R is a finite set of reactions;
- $St : R \mapsto 2^{S \times \mathcal{N}} \times 2^{S \times \mathcal{N}}$ maps reactions to a set of reactants and products;
- $K : R \mapsto eqn$ maps reactions to an equation;
- V_s is a finite set of variables;
- $A_s \subseteq (V_s \times \Re)$ is the assignment to the variables; and
- O are the remaining components of the SBML model.

Species in S are the genes, proteins, RNAP, and complex in the model. Reactions in R describe how the species are produced and consumed. The function St maps a reaction to the *stoichiometry* of the reactants and products. Stoichiometry is the number of reactants and products consumed and produced in a reaction. The function K maps each reaction to a *kinetic law* equation. Kinetic laws are mathematical formulas that describe the rate or probability at which the reaction fires. The set V_s includes the parameter variables. Assignments in A_s assigns values to the variables. O is an abstraction representing the remaining components of the SBML model.

Our abstracted model can also be represented graphically. The 10 species and 10 reactions for the SBML model of the genetic circuit in Fig. 2.2 are shown in Fig. 2.3. In this figure, species are represented with circles, such as CII. Reactions are represented with boxes, such as the "Open Complex PRE" reaction. The reactions are also labeled with a kinetic law. The kinetic law for the "Open Complex PRE" reaction is $K_o * RNAP * PRE - S1$. An edge labeled by an "r" means that a species is a *reactant* for a reaction (i.e., it is consumed by the reaction). An edge labeled by a "p" means that a species is a *product* for a reaction (i.e., it is produced by the reaction). In the case of

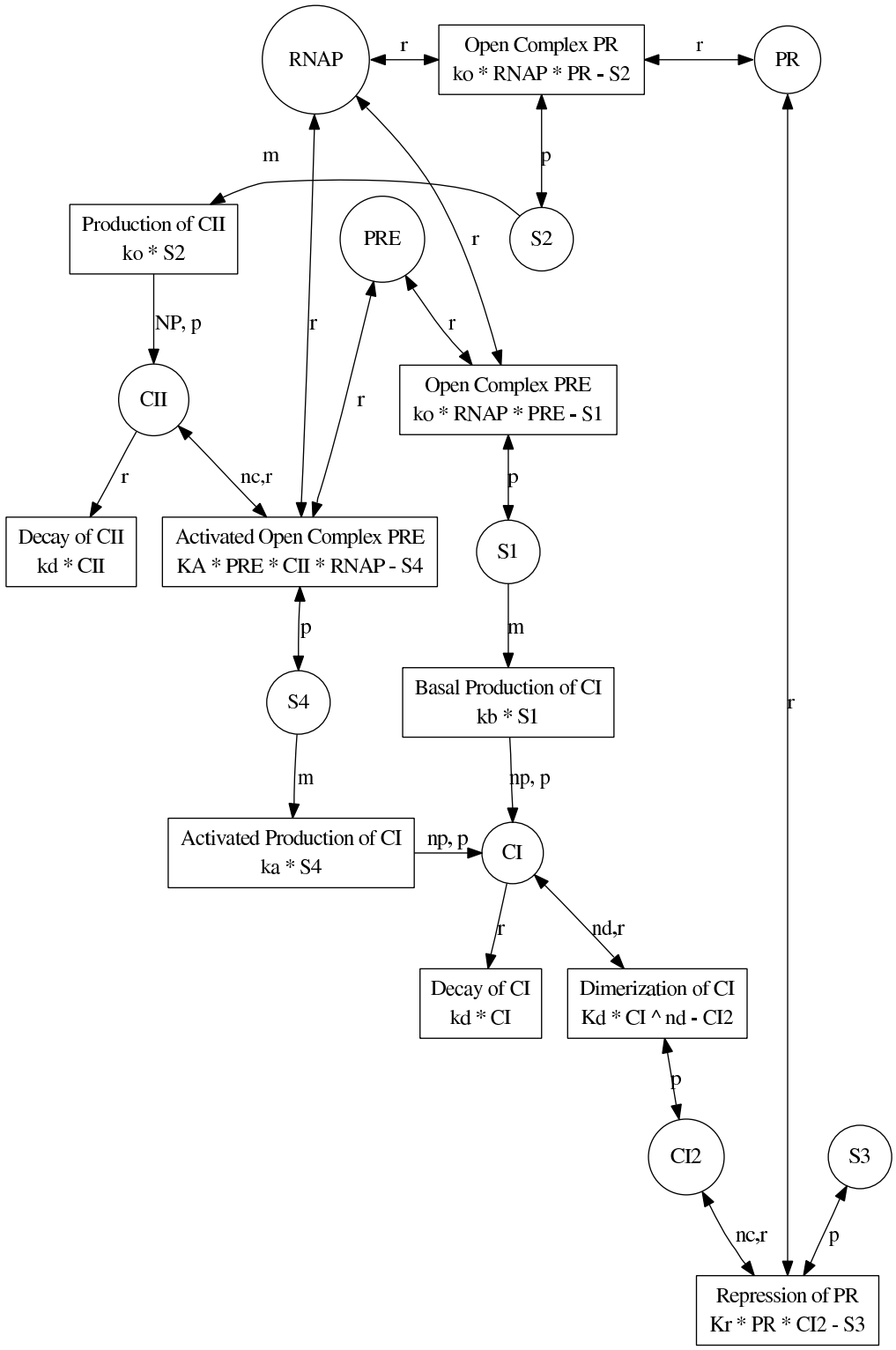


Figure 2.3. Species and reactions from the SBML model for the simple genetic circuit from phage λ.

the "Open Complex PRE" reaction, the reactants are RNAP and PRE, and the product is S1. An edge labeled by an "m" means that a species is a *modifier* for a reaction (i.e., it is neither produced nor consumed by the reaction). Species $S1$ is a modifier to the "Basal Production of CI" reaction. Edges are also annotated with a number to represent the stoichiometry for this species in the reaction. When the value is 1, it is omitted from the figure. The "Basal Production of CI" reaction has a stoichiometry of np, (i.e. it is dependent on the value of the parameter np). When an edge is bidirectional, this is a shorthand to indicate that the reaction is *reversible*. In other words, there are actually two reactions represented. One reaction converts the reactants to products while the second reaction converts the products back to reactants. The "Open Complex PRE" reaction is an example of a reversible reaction.

The model shown in Fig. 2.3 requires 10 species and 10 reactions to represent a simple oscillator. A more compact representation would be to only include the species and the influences of the species upon each other. This representation is shown in Fig. 2.4. The nodes of the graph are the species. The edges are the influences between the species. Activation is denoted by an edge with a normal arrowhead while repression is denoted by an edge with a flat arrowhead. This type of representation is the motivation for the GCM representation described in the next section.

2.3 Genetic Circuit Model

The GCM is a graphical specification language developed to represent genetic circuits. The goal of the language is to aid designers in creating SBML models of genetic circuits

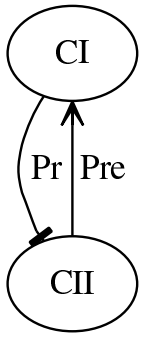


Figure 2.4. Graphical model of circuit.

by providing a method of describing genetic circuits at a higher level of abstraction. In particular, in a GCM, only important species and their influence upon each other are represented which greatly reduces the size of the model.

A GCM is a tuple $\langle S, P, G, I, I_B, S_D, V_g, A_g, E \rangle$ where:

- S is a finite set of species;
- P is a finite set of promoters;
- $G : P \mapsto 2^S$ maps promoters to sets of species;
- $I \subseteq S \times P \times \{a, r\}$ is a finite set of influences;
- $I_B \subseteq I$ is the set of *biochemical influences*;
- $S_D \subseteq S$ is a set of species that influence as dimers;
- V_g is a finite set of variables;
- $A_g \subseteq (V_g \times \Re)$ is the assignment to the variables; and
- E is an SBML model.

Species in S are the proteins in the genetic circuit, and the promoters in P are the locations on the DNA that initiate transcription of genes that produce these proteins. The set of species that are produced from a promoter is represented by the function G. Species can have influences on promoters. These influences can either be of type activation (denoted "a") or repression (denoted "r"). If the type is activation, then as the amount of the species increases, the rate of production of the species associated with the influenced promoter also increases. If the type is repression, then as the amount of the species increases, the rate of production of the species associated with the promoter decreases. The influences in I_B are influences that require species involved to undergo a biochemical reaction in order to bind to the promoter. The species in S_D only influence other species after forming a dimer. The variables in V_g and the assignment of the variables in A_g are the default values of the parameters to be used in the SBML model, shown in Table 2.1. The model, species, promoters, and influences each have a set of associated parameters that can be customized. The default values can either be changed globally to affect the entire model or individually to affect only particular elements. The SBML model E is used as a starting point when outputting SBML. This allows the GCM to use the rules,

Table 2.1. Parameter List.

Parameter	Symbol	Structure	Value	Units
Initial RNAP count	n_r	model	30	$molecule$
Initial species count	n_s	species	0	$molecule$
Dimerization equilibrium	K_d	species	.05	$\frac{1}{molecule}$
Degradation rate	k_d	species	.0075	$\frac{1}{sec}$
Biochemical equilibrium	K_b	influence	.05	$\frac{1}{molecules}$
Degree of cooperativity	n_c	influence	2	$molecule$
N-mer as transcription factor	nd	influence	1	$molecule$
Repression binding equilibrium	K_r	influence	.5	$\frac{1}{molecule^{n_c}}$
Activation binding equilibrium	K_a	influence	.0033	$\frac{1}{molecule^{(n_c+1)}}$
Initial promoter count	n_g	promoter	2	$molecule$
RNAP binding equilibrium	K_o	promoter	.033	$\frac{1}{molecule}$
Basal production rate	k_b	promoter	.0001	$\frac{1}{sec}$
Open complex production rate	k_o	promoter	.05	$\frac{1}{sec}$
Activated open complex production rate	k_a	promoter	.25	$\frac{1}{sec}$
Stoichiometry of production	np	promoter	10	$molecule$

events, constraints, and assignments to create a testing SBML model that can be used by multiple GCM's.

A graphical representation of a GCM is a bipartite graph with species and promoters as the two types of nodes. Species are connected to promoters using the influences in I, and promoters are connected to species using the function G. To simplify presentation, the graphs shown in this thesis use only species as nodes, edges are inferred using I and G, and edges are labeled with the promoter that links the species. The graphical representation of the GCM for the simple genetic circuit is shown in Fig. 2.4.

Promoters group influences together that are associated with the same gene(s). As an example, consider the GCM shown in Fig. 2.5(a) which represents the genetic circuit shown in Fig. 2.5(b). With the same promoter associated with both influences, if either A or B is present, then C is repressed. In other words, this behaves like the NOR gate schematically shown in Fig 2.5(c). Another example is shown in Fig. 2.6(a) which represents the genetic circuit shown in Fig. 2.6(b). With a different promoter for each influence, both A and B must be present to repress C. In this case, the behavior is like the NAND gate schematically shown in Fig 2.6(c).

Promoters can also group biochemical influences together so that all input species of the influences must be present in order to bind to the promoter. As an example, consider

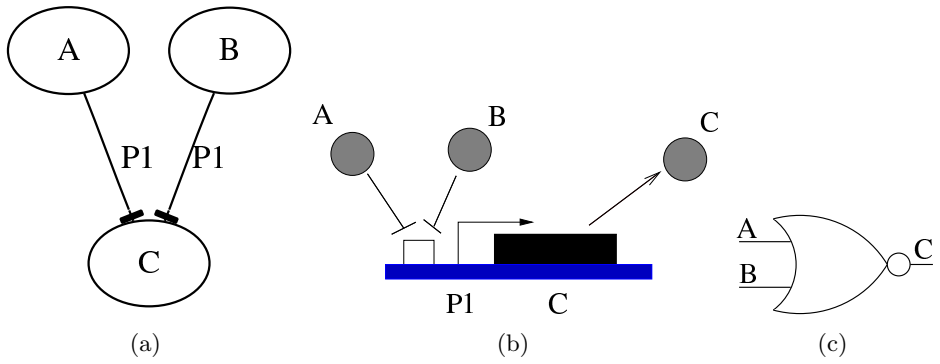


Figure 2.5. Example using same promoter. (a) GCM, (b) genetic circuit, and (c) logical behavior with same promoter.

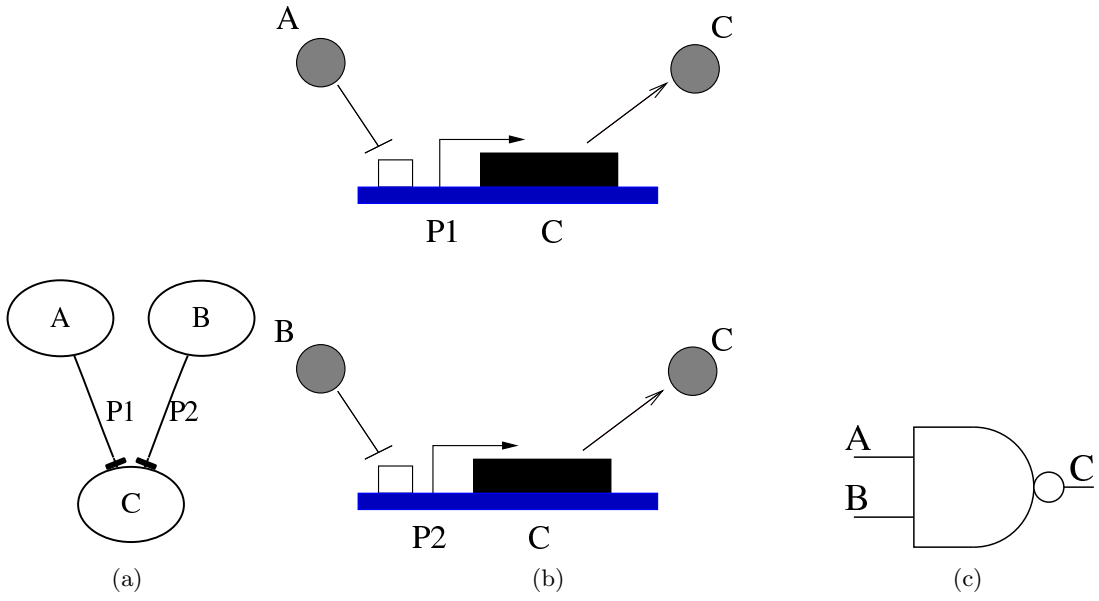


Figure 2.6. Example using different promoter. (a) GCM, (b) genetic circuit, and (d) logical behavior with different promoters.

the GCM shown in Fig. 2.7(a) which represents the genetic circuit shown in Fig. 2.7(b). With no biochemical influence associated with the promoter, if either A or B is present, then C is produced. In other words, this behaves like the OR gate schematically shown in Fig 2.7(c). Another example is shown in Fig. 2.8(a) which represents the genetic circuit shown in Fig. 2.8(b). With a biochemical influence for the promoter, both A and B must bind to form a complex which activates C production. In this case, the behavior is like the AND gate schematically shown in Fig 2.8(c).

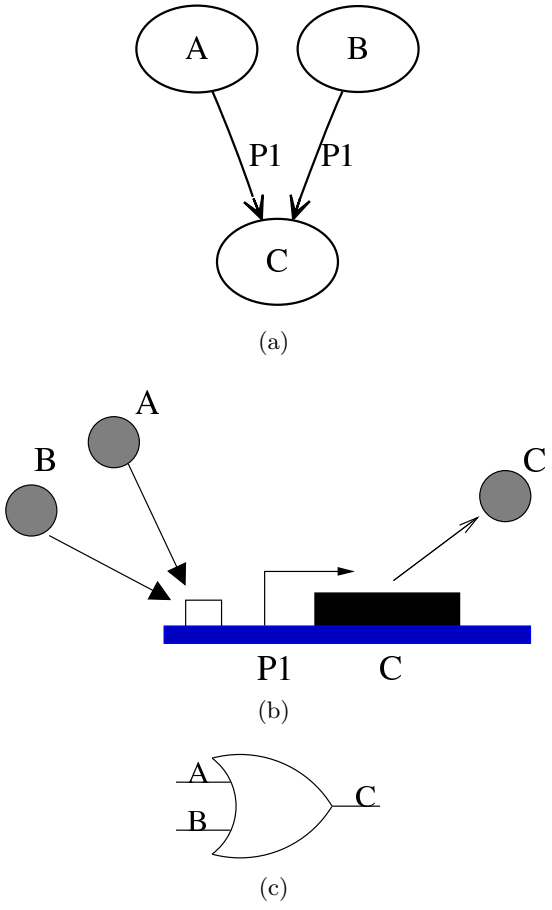


Figure 2.7. Example with no biochemical influences. (a) GCM, (b) genetic circuit, and (c) logical behavior with no biochemical influence.

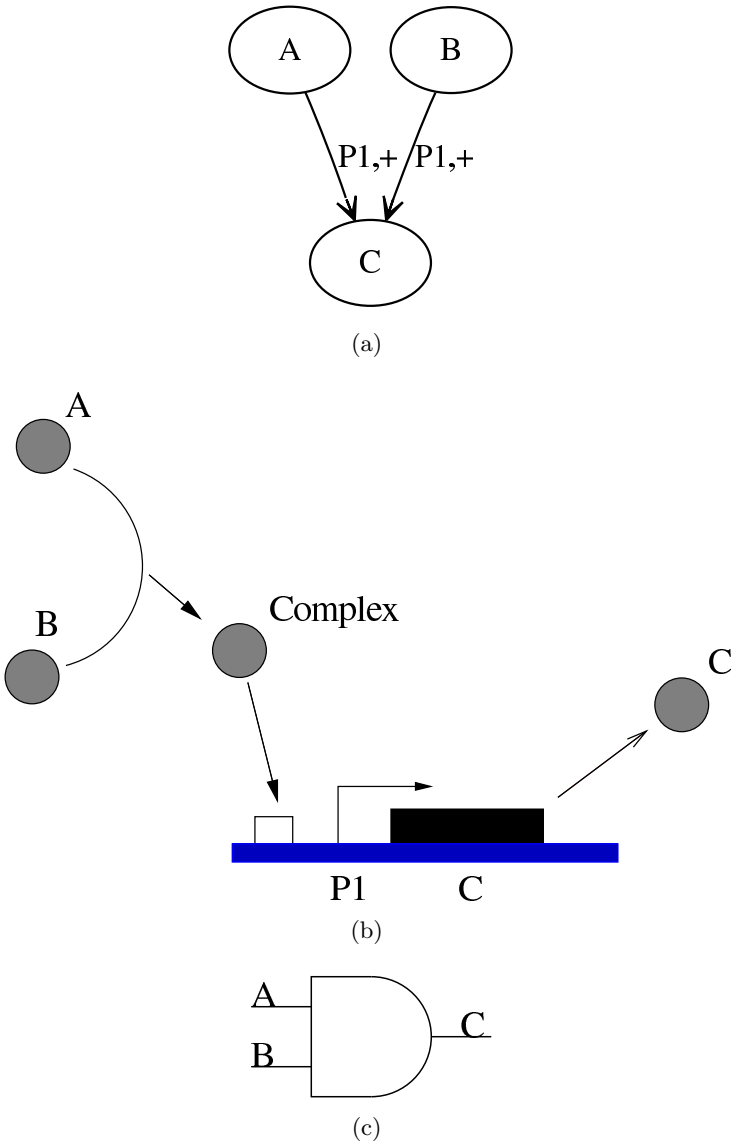


Figure 2.8. Example with biochemical influences. (a) GCM, (b) genetic circuit, and (c) logical behavior with biochemical influence.

Using the GCM language, the simple genetic circuit shown in Fig. 2.2(b) requires only two species and two influences. To describe this same circuit using SBML would require 10 nodes (i.e., species) and 10 edges (i.e., reactions) as shown in Fig. 2.3. Creating a GCM is more efficient than creating the SBML model. This SBML model, however, can now be automatically synthesized from the GCM representation as described in the next section.

2.4 SBML Generation

While `iBioSim` and other tools allow a user to create SBML models, these models are extremely detailed making them tedious and time-consuming to construct. Clearly, there is a need for a higher level of abstraction in the design of genetic circuits. As described above, GCM provides this higher level abstraction by allowing a genetic circuit to be specified using only the important species and their influences upon each others production rates.

Algorithm 1 synthesizes SBML from a GCM. This algorithm gives an overview, as the individual functions are discussed in detail later. Initially, the SBML model is set to the same as the SBML in the GCM and added to the SBML model (line 1). This allows the user to create an SBML model containing rules, events, constraints, and initial conditions, and those SBML features are merged with the GCM's SBML output. The utility of this feature is multiple GCM's can be tested using the same environment. Next, parameters are generated from the GCM (line 2). Next, a degradation reaction is created for each species in S (line 3). Next, open complex reactions are formed for each promoter (line 4). Next, dimer reactions are formed (line 5). Next, biochemical reactions are formed (lines 6-7). Next, repression reactions are formed (line 8). Finally, activation reactions are formed (line 9).

Algorithm 2 shows how degradation reactions are formed. For each species, a degradation reaction is created added to the model (lines 2-3). The result for our simple genetic circuit is shown in Fig. 2.9.

Algorithm 3 shows a helper function that creates reversible reactions that is used in later algorithms. This function takes a reaction, and creates a copy of the reaction (line 1). Next, the reactants and products of the reaction are exchanged (lines 6-4). Finally, a kinetic law representing the destruction of the complex is created. Since parameters used in kinetic laws represent the equilibrium constants, the kinetic law of the reversible

Algorithm 1: GCM2SBML(S,P,G,I,I_B,S_D,V_g,A_g,E)

$M = E$;
$(V_s, A_s) = (V_g, A_g)$;
$M = CreateDegradation(M)$;
$M = CreateOpenComplex(M, P)$;
$(M, I) = CreateDimer(M, S_D, I)$;
$(M, I) = CreateBiochemical(M, I, I_B, P, a)$;
$(M, I) = CreateBiochemical(M, I, I_B, P, r)$;
$M = CreateRepression(M)$;
$M = CreateActivation(M)$;
return M;

Algorithm 2: CreateDegradation(M)

forall $s \in S$ **do**
    $r =$ newReaction();
    $R = R \cup r$;
    $K(r) = kd * s$;
    $St(r) = (\{s\}, \emptyset)$;
return M;

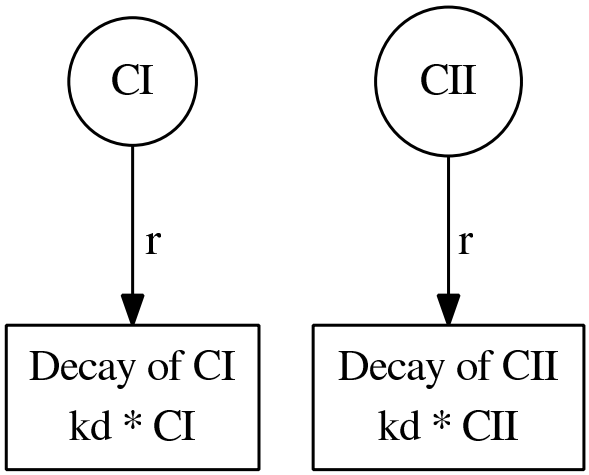


Figure 2.9. Degradation of CI and CII.

Algorithm 3: reverseReaction(r)

```
1 r' = copyReaction(r);
2 (St_r, St_p) = St(r);
3 St(r') = (St_p, St_r);
4 K(r') = 1;
5 forall s ∈ S_p do
6     K(r') = K(r') * s;
7 return r';
```

reaction is directly proportional to the concentration of the products (lines 4-6). Finally, the new reaction is returned (line 7).

Algorithm 4 shows how the open complex reactions are formed. First, a species is created for RNAP and each promoter in P (line 1). Next, a new complex species is created for each promoter in P combined with RNAP, and reactions are added to create and destroy this complex (lines 3-8). Next, if the promoter does not need an activator, then a open complex reaction is created (lines 11-15). If the promoter does require an activator, then a basal reaction is created (lines 17-20). The reactions created by this step for our example are shown in Fig. 2.10.

Next, Algorithm 5 shows how dimerization reactions are formed. For each species that influences in dimer form, a dimer species is created and reactions to create and destroy them are generated (line 1-7). For our example, this results in the reversible reaction shown in Fig. 2.11. All influences that use this species are updated to use the dimer species instead (lines 8-9).

Algorithm 6 shows how biochemical reactions are formed. Fig. 2.2 did not include a biochemical reaction so to illustrate this process, the AND gate shown in Fig. 2.8(a) is used as an example. The first step in Algorithm 6 is for each promoter, find the set of species that have a biochemical influence on the promoter (line 2). If this set is non-empty, then a new complex is created and added to the set of species (lines 3-5). Next, reactions are created for formation and destruction of the biochemical complex (line 9). Finally, all influences are updated to use the new complex (line 11). The result for the new biochemical reaction is shown in Fig. 2.12.

Next, Algorithm 7 shows how repression reactions are formed. For each repression influence, a complex species is created that is composed of the promoter and repressor species (lines 1-3), and reactions are added to create and destroy this complex (lines 4-7). For our example, CI dimers repress CII production using the reaction shown in Fig. 2.13.

Algorithm 4: CreateOpenComplex(M,P)

$S_M = S_M \cup \{RNAP\} \cup P$;
forall $p \in P$ **do**
    $c =$ newComplex();
    $S_M = S_M \cup \{c\}$;
    $r =$ newReaction();
    $R = R \cup r \cup reverseReaction(r)$;
    $K(r) = Ko * RNAP * p$;
    $St(r) = (\{p, RNAP\}, \{c\})$;
    $I_a = \{i \in I | i \in (*, p, a)\}$;
    if $I_a = \emptyset$ **then**
        forall $s \in G(p)$ **do**
            $r =$ newReaction();
            $R = R \cup r$;
            $K(r) = ko * c$;
            $St(r) = (\{c\}, \{(s, n_p), c\})$;
    else
        forall $s \in G(p)$ **do**
            $r =$ newReaction();
            $R = R \cup r$; $K(r) = kb * c$;
            $St(r) = (\{c\}, \{(s, n_p), c\})$;
return M;

Algorithm 5: CreateDimer(M,S_D,I)

forall $s \in S_D$ **do**
    $c =$ newComplex();
    $S = S \cup \{c\}$;
    $r =$ newReaction();
    $R = R \cup r \cup reverseReaction(r)$;
    $K(r) = K_d * n_d * s^{n_d}$;
    $St(r) = (\{(s, N_D)\}, \{c\})$;
    forall $(s, p, *) \in I$ **do**
        $I = (I - \{(s, p, *)\}) \cup \{(c, p, *)\}$;
return (M,I);

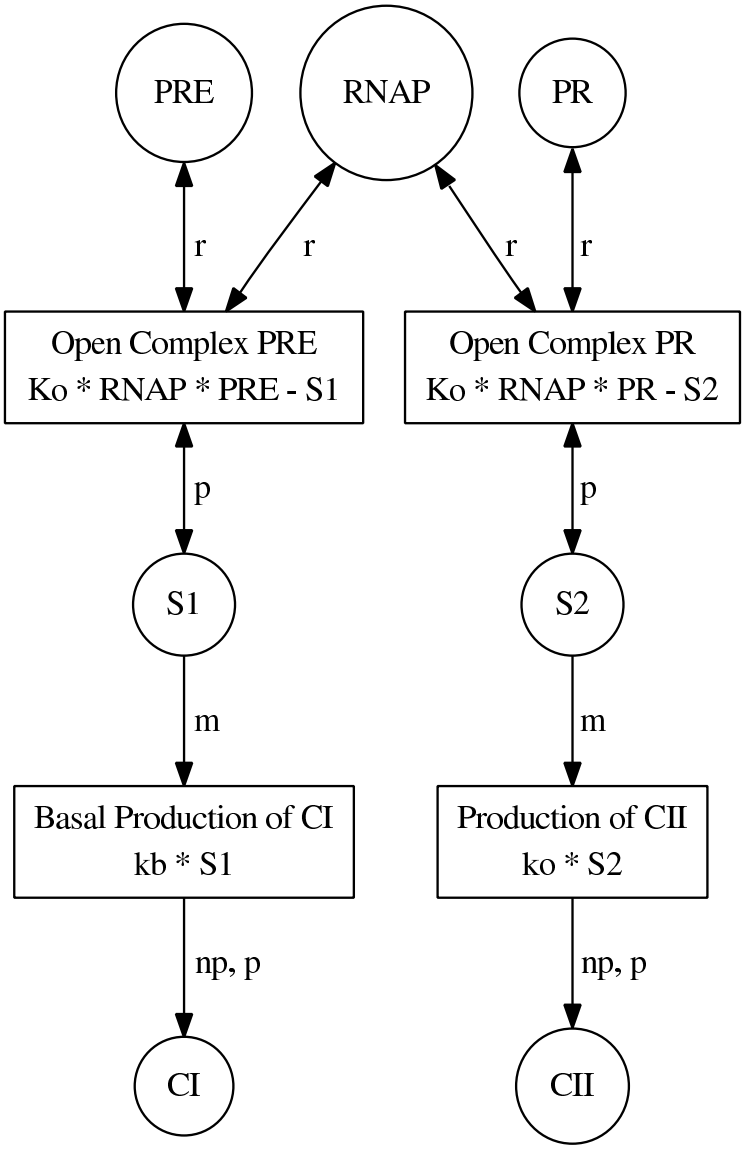


Figure 2.10. CI and CII's open complex formation reactions.

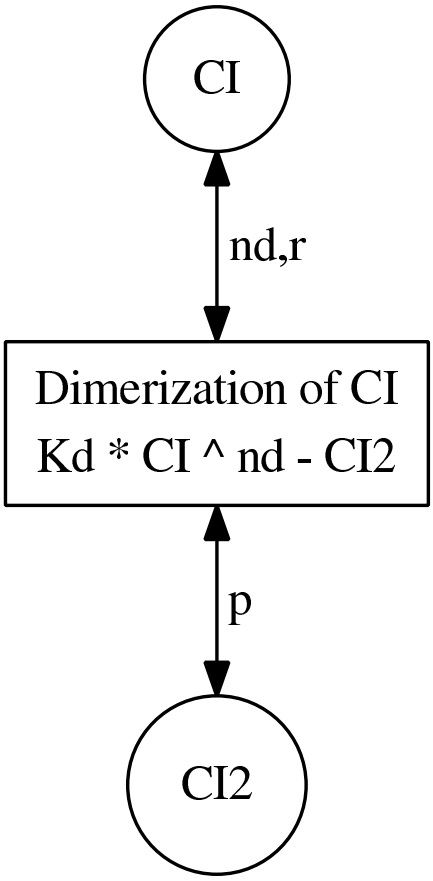


Figure 2.11. Dimerization reaction for CI.

Algorithm 6: CreateBiochemical(M,I,I_B,P,t)

forall $p \in P$ **do**
  $B_t = \{s \in S | (s,p,t) \in I_B\}$;
  if $|B_t| > 0$ **then**
    c = newComplex();
    $S = S \cup \{c\}$;
    r = newReaction();
    $R = R \cup r \cup reverseReaction(r)$;
    $K(r) = K_b * B_t$;
    $St(r) = (\{B_t\}, \{c\})$;
    forall $s \in B_t$ **do**
      $I = I - \{(s,p,t)\} \cup \{(c,p,t)\}$;
return (M,I);

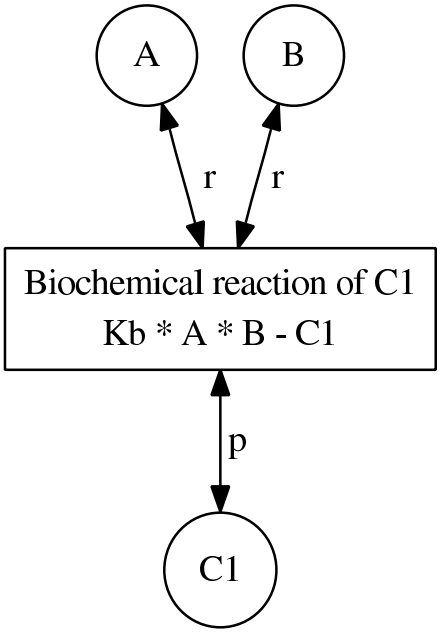


Figure 2.12. Complex formation of A and B for Figure 2.8.

Algorithm 7: CreateRepression(M)

forall $(s, p, r) \in I$ **do**
    $c = \text{newComplex}()$;
    $S = S \cup \{c\}$;
    $r = \text{newReaction}()$;
    $R = R \cup r \cup reverseReaction(r)$;
    $K(r) = Kr * s^{n_c} * p$;
    $St(r) = (\{(s, n_c), p\}, \{c\})$;
return M;

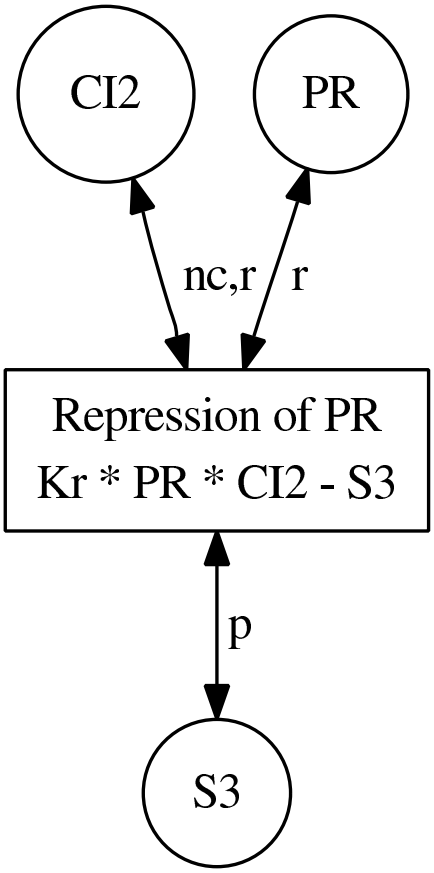


Figure 2.13. Reactions for CI dimer repression of CII.

Algorithm 8: CreateActivation(M)

```
1  forall (s, p, a) ∈ I do
2      c = newComplex();
3      S = S ∪ {c};
4      r = newReaction();
5      R = R ∪ r ∪ reverseReaction(r);
6      K(r) = Ka * s^{n_c} * RNAP * p;
7      St(r) = ({(s, n_c), p, RNAP}, {c});
8      forall s ∈ G(p) do
9          r = newReaction();
10         R = R ∪ r;
11         K(r) = ka * c;
12         St(r) = ({c}, {(s, n_p), c});
13 return M;
```

Next, Algorithm 8 shows how activation reactions are formed. For each activation influence, a complex species is created that is composed of the promoter, the activator, and RNAP, and reactions are added to create and destroy this species (lines 1-7). A reaction must also be added that uses this complex species to produce the proteins associated with this promoter (lines 8-12). This reaction uses the activated production rate. In our example, CII activates CI production using the reactions shown in Fig. 2.14. The final step is to merge the environment file with the new SBML model. The final result is the SBML model shown in Fig 2.3.

`iBioSim` includes a GCM editor to aid users in creating a GCM. Using this interface, species, promoters, and influences can be added, and their parameters can be globally or individually modified. The editor ensures that the user enters a valid GCM.

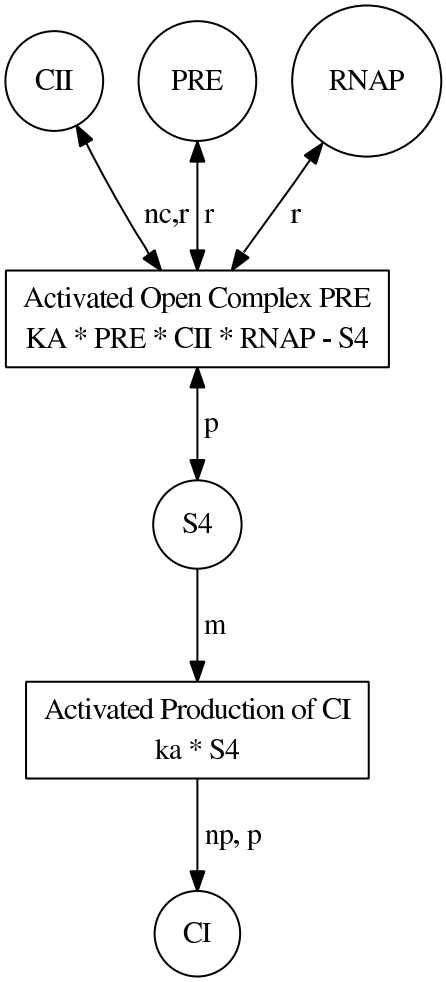


Figure 2.14. Reactions for CII activation of production of CI.

CHAPTER 3

GENETIC MULLER C-ELEMENT

Genetic circuits are inherently asynchronous in nature because there is no global clock inside of a cell. Therefore, it may be possible to adapt asynchronous analysis and synthesis methods to the design of genetic circuits. Our long-term goal is to adapt asynchronous synthesis and technology mapping tools to target a genetic circuit technology. To achieve this goal, it is essential to first develop a library of gates sufficient to design interesting genetic circuits. While synthetic combinational logic gates have been constructed, there are many open issues in the design of sequential logic gates. One such gate common in most asynchronous circuits is the Muller C-element, which is used to synchronize multiple independent processes. The design of a genetic Muller C-element enables the construction of virtually any asynchronous circuit from genetic material.

In previous research [28], we proposed two potential designs. Models and analysis of these designs were done at a low level using SBML. Errors were easy to introduce because each reaction had to be entered by hand. Using the GCM language, we are able to greatly speed up our design and analysis time as well as reduce errors during modeling. This chapter presents a case study of the design of three different genetic Muller C-elements using the GCM language. Section 3.1 presents the three different GCM structures that represent a Muller C-element. In order to determine the most robust design, each GCM is analyzed using nullcline analysis in Section 3.2 and stochastic simulation analysis in Section 3.3. Section 3.4 shows the effects of parameter variation on the robustness of the circuits. Finally, Section 3.5 outlines the design principles learned from the analysis of the genetic Muller C-element. Section 3.6 presents a potential application of the genetic Muller C-element.

3.1 A Genetic Muller C-element

The Muller C-element with the schematic symbol shown in Fig. 3.1 is a standard component used in asynchronous circuit design. The Muller C-element has two inputs

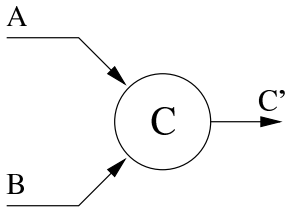


Figure 3.1. Schematic symbol for a Muller C-element.

and one latched output. The output is driven high when both the inputs are high, and it is driven low when both the inputs are low. If the two inputs are not equal, the latched output does not change. The behavior is described by the truth table shown in Table 3.1. This gate is vital in the creation of asynchronous circuits.

A key challenge in designing a robust genetic Muller C-element is ensuring that the gate functions correctly when the the inputs are mixed. During this state, the output can either be low or high, depending on the previous output state. This implies that there exists two stable states when the inputs are mixed, and the C-element must be able to stay in the correct stable state when the inputs are mixed.

This section presents three different designs for the Muller C-element. Each design is logically equivalent to a Muller C-element, but differ in the genetic connectivity. This section presents various results for the average of 1,000 stochastic simulations for each design. The abstractibility of the GCM output proves to be invaluable, as the simulation time is improved substantially for each simulation.

3.1.1 Majority Gate Design

Our first design is based on the simplest implementation of a Muller C-element, a *majority gate*. The logical design is shown in Fig. 3.2(a) and the GCM is shown in Fig. 3.2(b). The GCM contains 8 species and 11 influences. The design is constructed

Table 3.1. Truth Table for a Muller C-element.

A	B	C'
0	0	0
0	1	C
1	0	C
1	1	1

from three 2-input NAND gates, 2 inverters acting as a buffer, and one 3-input NAND gate. The inputs to the three 2-input NAND gates are A, B, and D. D is used to hold state in this design. The design is called a majority gate design because when the majority of the inputs are low, the output is low, and when the majority of the inputs are high, the output is high. This can be seen from Fig. 3.2(a). Each input is fed into two different 2-input NAND gates. When only one of the input signals is high, then signals X, Y, and Z are still high. In order for the output of any of the X, Y, and Z NAND gates to be low, two of the inputs must be high. Once two of the inputs are high, then one of the values of X, Y, or Z should go low, resulting in a high output value.

The genetic circuit schematic is shown in Fig. 3.2(c). The three inputs are A, B, and D. Each input is fed into two different gates. A is fed into the X and Y 2-input NAND gate, B is fed into the X and Z 2-input NAND gate, and D is fed into the Y and Z 2-input NAND gate. If only one or less of the inputs is high, then X, Y, and Z have a source of production. For example, if only A is high, then the X and Y 2-input NAND will still output high because B and D are still low. There must be at least two high inputs in order to repress production of either X, Y, and Z. Once X, Y, or Z is low, then D can be produced, repressing E, and allowing C to be produced. The generated SBML before and after abstraction is shown in Fig. 3.3. While the labels have been removed, Fig. 3.3 shows the reduction in the SBML model. Before abstraction, the SBML contains 33 species and 30 reactions. After abstraction, the SBML contains 8 species and 14 reactions.

While this is the simplest implementation, the average simulation results for 1000 stochastic simulation runs in Fig. 3.4 show that this implementation does behave correctly. Initially, both input signals are low. At 2500 seconds, A goes high. The level of C continues to remain low. At 5000 seconds, B also goes high, causing the level of C to rise to a high level. At 7500 seconds, A goes to low, and the level of C continues to hold high. This shows that the majority gate design does hold state. Finally, at 10000 seconds, B is removed and C falls back to the low level. The simulation time for abstracted versus unabstracted is 19 minutes versus 2 hours and 21 minutes. The simulation results also show that the abstracted and unabstracted results agree quite well.

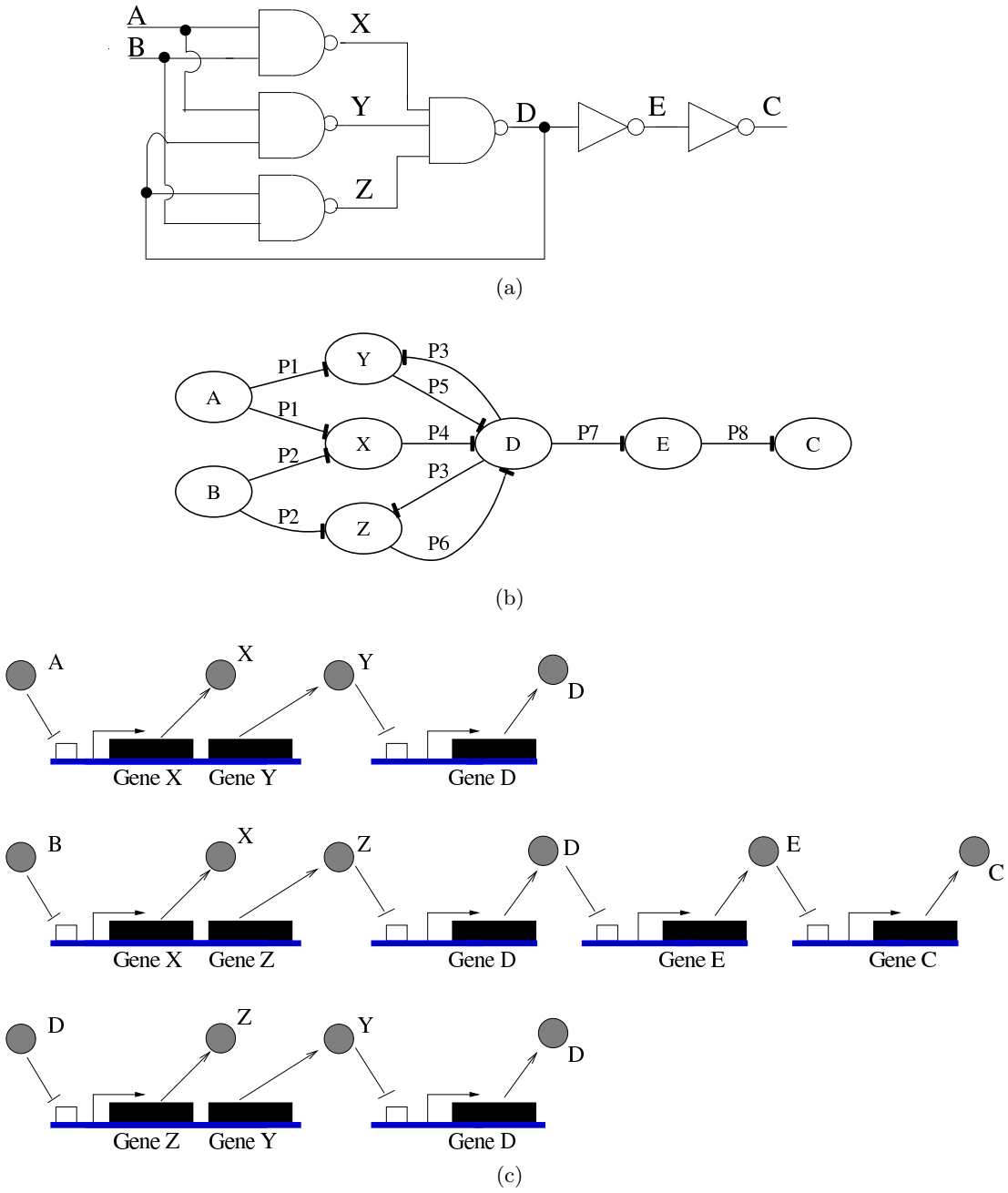
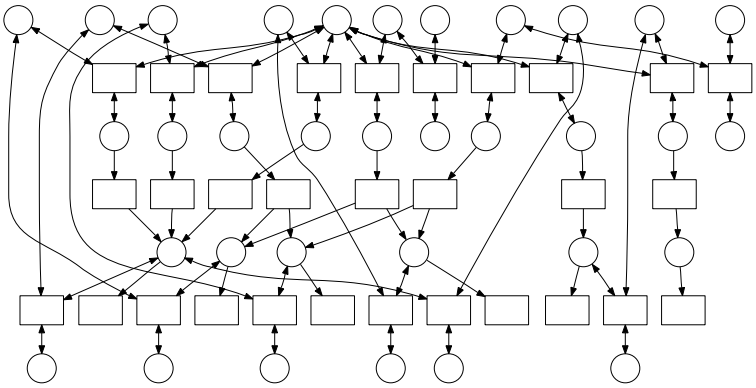
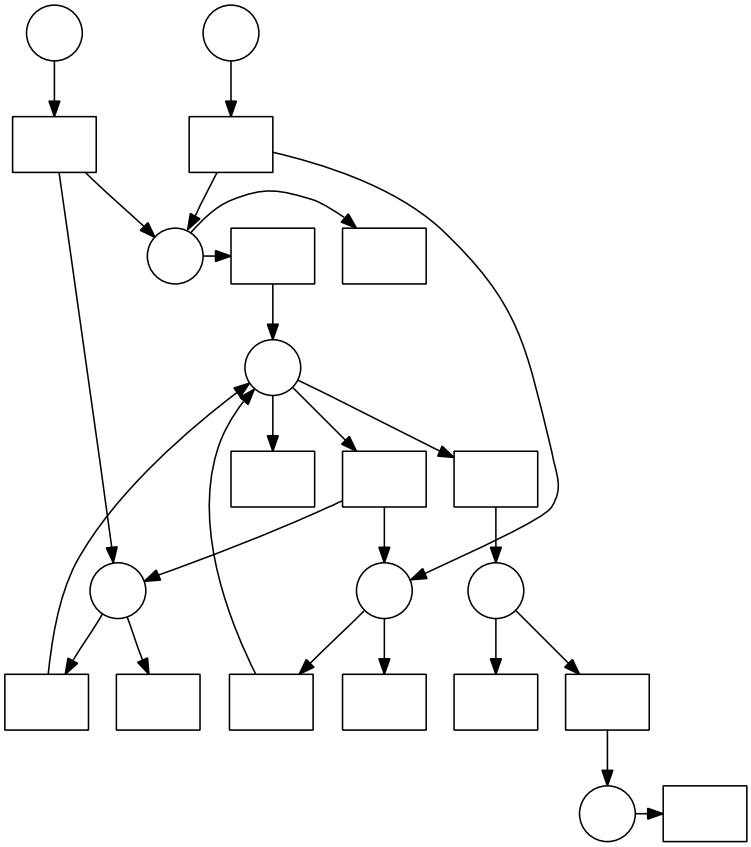


Figure 3.2. Genetic majority gate designs. (a) Logical model, (b) GCM and, (c) genetic circuit for the genetic majority C-element.



(a)



(b)

Figure 3.3. SBML (a) before abstraction and (b) after abstraction for the genetic majority C-element.

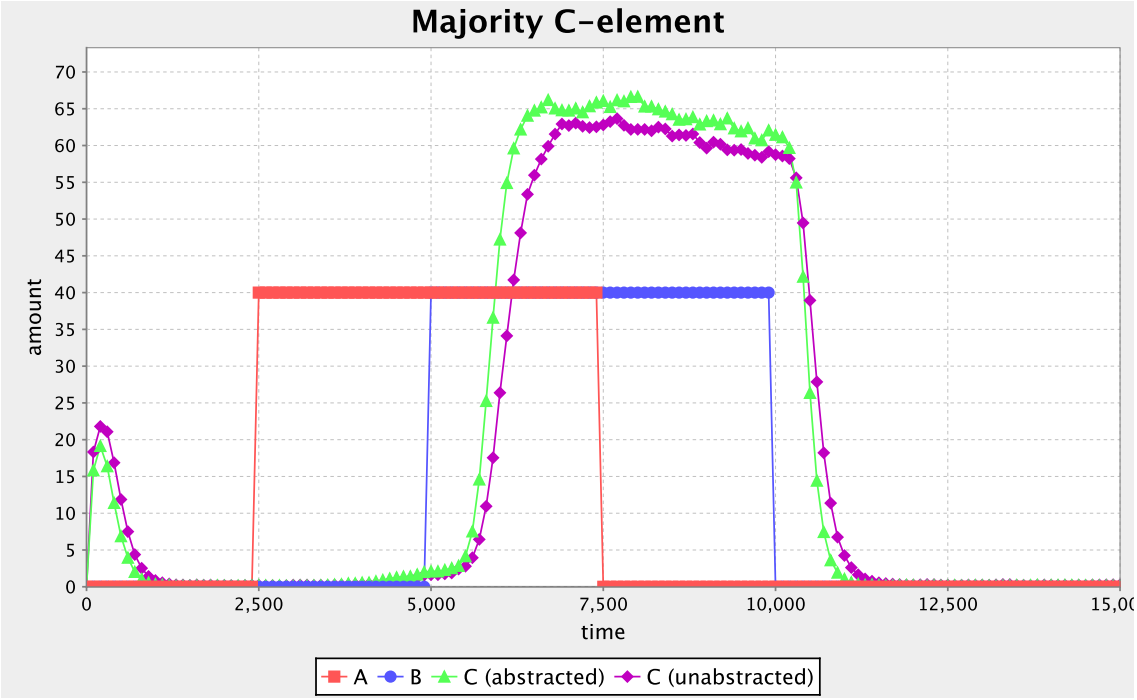
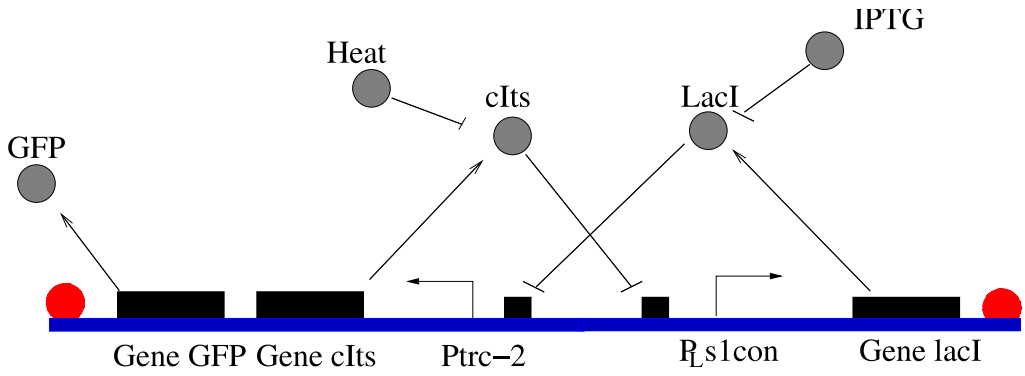


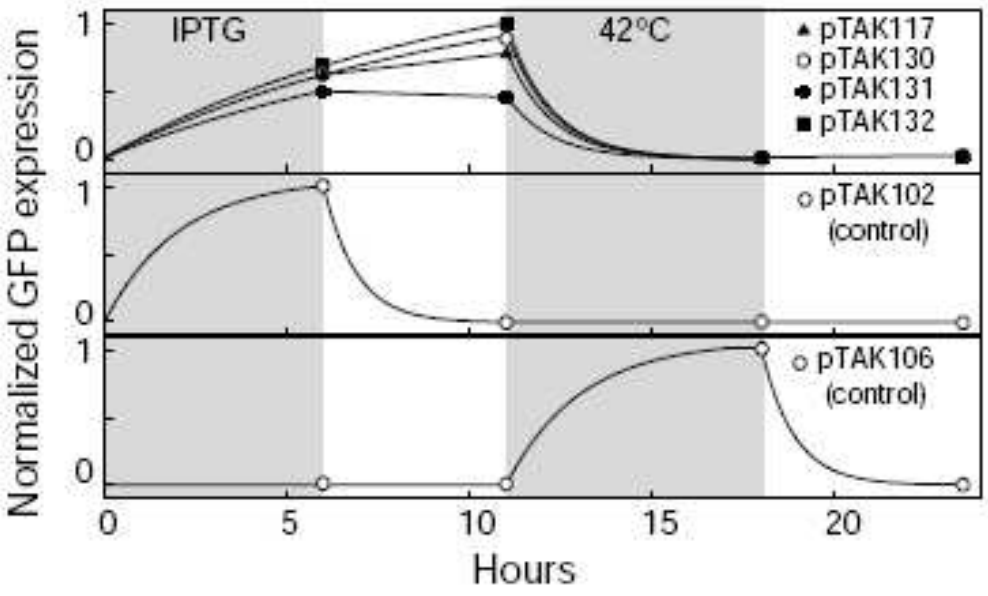
Figure 3.4. Results from simulation of the majority genetic Muller C-element.

3.1.2 Toggle Switch Design

Our second design uses the genetic toggle element developed by Gardner *et al.* [14]. The genetic toggle element uses two genes that mutually repress each other. Fig. 3.5(a) shows the model for this toggle switch, and Fig. 3.5(b) shows their experimental results. First, the inducer, IPTG, is added to the system which prevents LacI from repressing the production of CIts (i.e, temperature sensitive CI). This causes the levels of CIts and GFP to rise to their high levels setting the switch into its high state. When CIts is high, CIts blocks the P_Ls1con promoter, which prevents LacI from being created. Therefore, even after IPTG is removed, the switch stays in its high state. When heat is introduced into the system, it inhibits CIts, which allows LacI to be produced. The LacI produced represses the production of CIts and GFP changing the switch to its low state. When heat is removed from the system, LacI remains high so it holds this state.



(a)



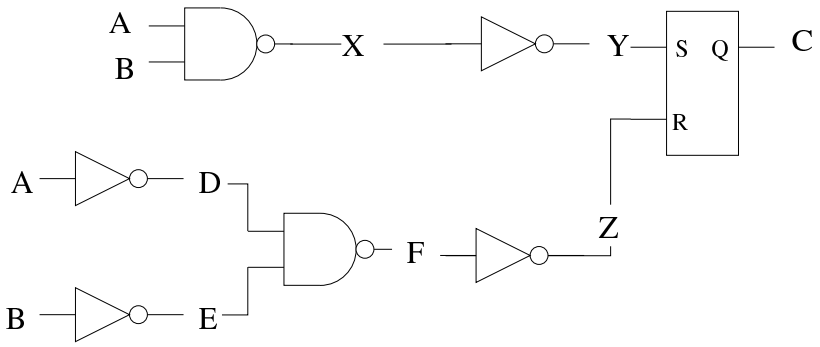
(b)

Figure 3.5. The genetic toggle switch's (a) genetic design and (b) experimental results (courtesy of [14]).

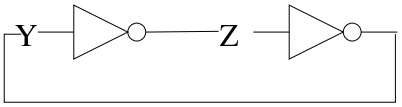
The genetic toggle is used to hold state in the genetic toggle Muller C-element. The logical design for our toggle gate design is shown in Fig. 3.6(a) and the GCM is shown in Fig. 3.6(b). The GCM for the toggle gate design contains 9 species and 11 influences. The gate is constructed from two NAND gates, several inverters, and the toggle element. The key element is the the toggle element that is set by Y and reset by Z. Both Y and Z are used to hold state in this design. When both A and B are high, then X is low, which results in Y going high and Z going low. This sets the toggle element to be in the high state. When both A and B are low, then D and E are high, resulting in Z going to high and Y going low, resetting the toggle element to be in the low state. When the inputs are mixed, Y and Z hold their current level, and the toggle element maintains the previous state.

The genetic circuit schematic is shown in Fig. 3.7. Both Y and Z, the molecules of the toggle element, are produced from two different sources, one in the toggle element, and one outside the toggle element. When both A and B are high, then X is not produced, resulting in the production of Y from outside the toggle element. This shuts down the production of Z inside the toggle element, allowing Y to be produced from the toggle element, and thus setting the toggle element. Once set, C can also be produced, resulting in a high output. When both A and B are low, then D and E are produced, which represses the production of F, a repressor molecule for the production of Z from outside the toggle element. This shuts down the production of Y and C in the toggle, resulting in a low output.

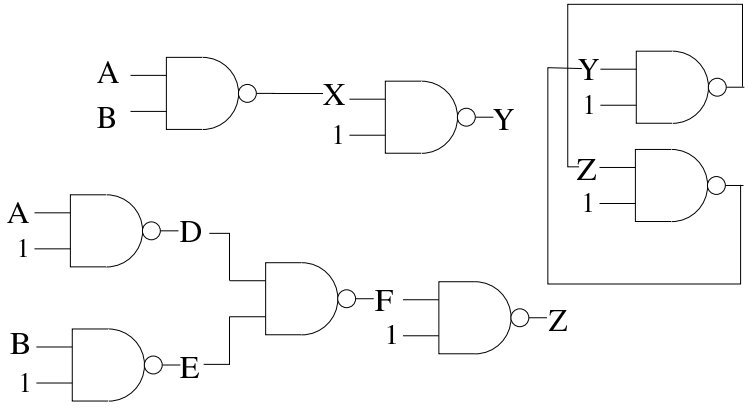
The generated SBML before and after abstraction is shown in Fig. 3.8. While the labels have been removed, Fig. 3.8 shows the reduction in the SBML model. Before abstraction, the SBML contains 34 species and 31 reactions. After abstraction, the SBML contains 9 species and 15 reactions. The simulation results for 1,000 stochastic simulation runs in Fig. 3.9 show the behavior of the circuit. Initially, both inputs are low, so C remains low. At 2500 seconds, A goes high, which causes C to rise slightly. At 5000 seconds, B also goes high, which causes C to rise to its high level. At 7500 seconds, A is removed, and the level of C continues to remain high. Finally, B is removed and C rapidly drops back down to the low level. The simulation results show that the toggle switch design also behaves correctly. The simulation time for abstracted versus unabstracted is 17 minutes versus 2 hours and 18 minutes. The simulation results also show that the abstracted and unabstracted results agree quite well.



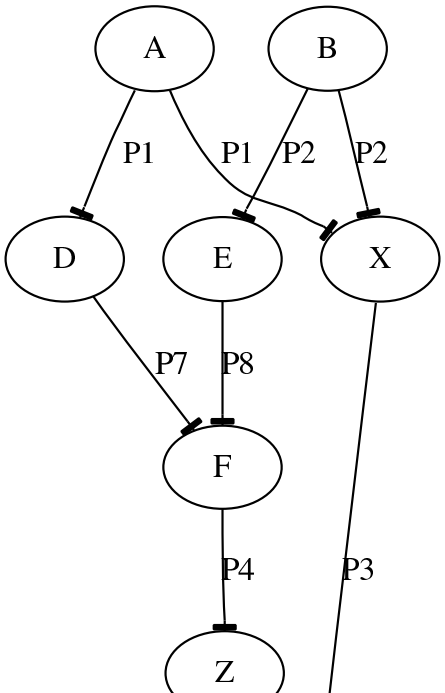
Where the S–R Latch is as follows



So using only 2NAND, our system is:



(a)



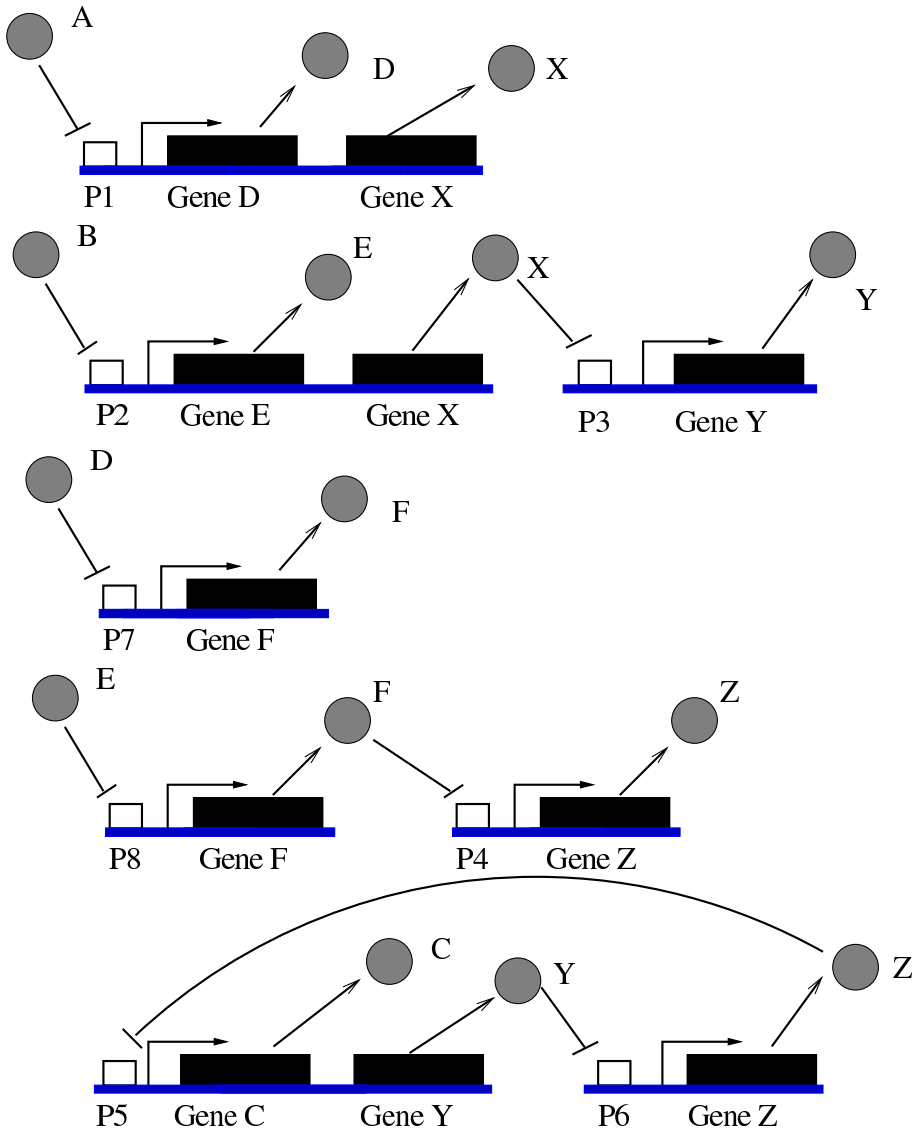
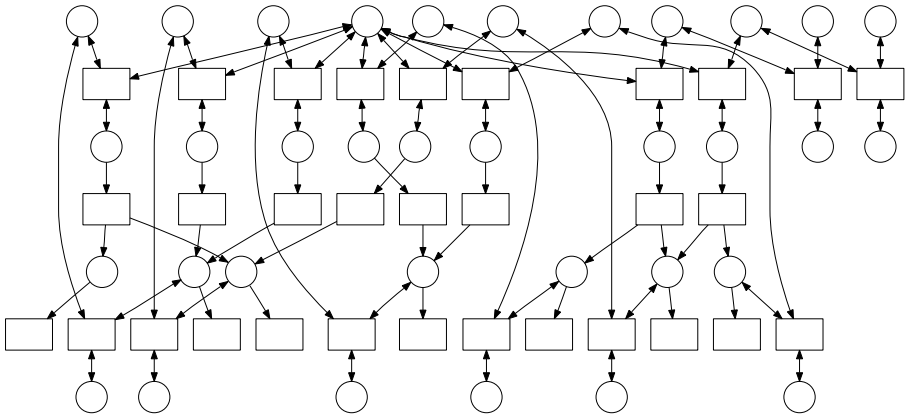
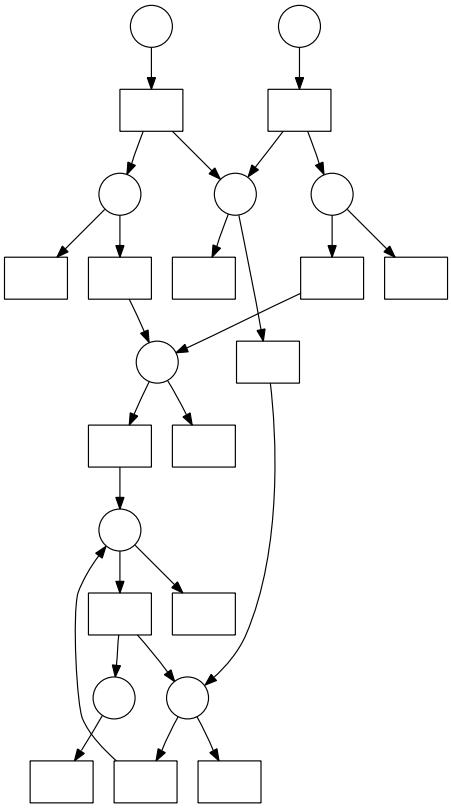


Figure 3.7. Genetic circuit diagram for the genetic toggle C-element.



(a)



(b)

Figure 3.8. SBML (a) before abstraction and (b) after abstraction for the genetic toggle C-element.

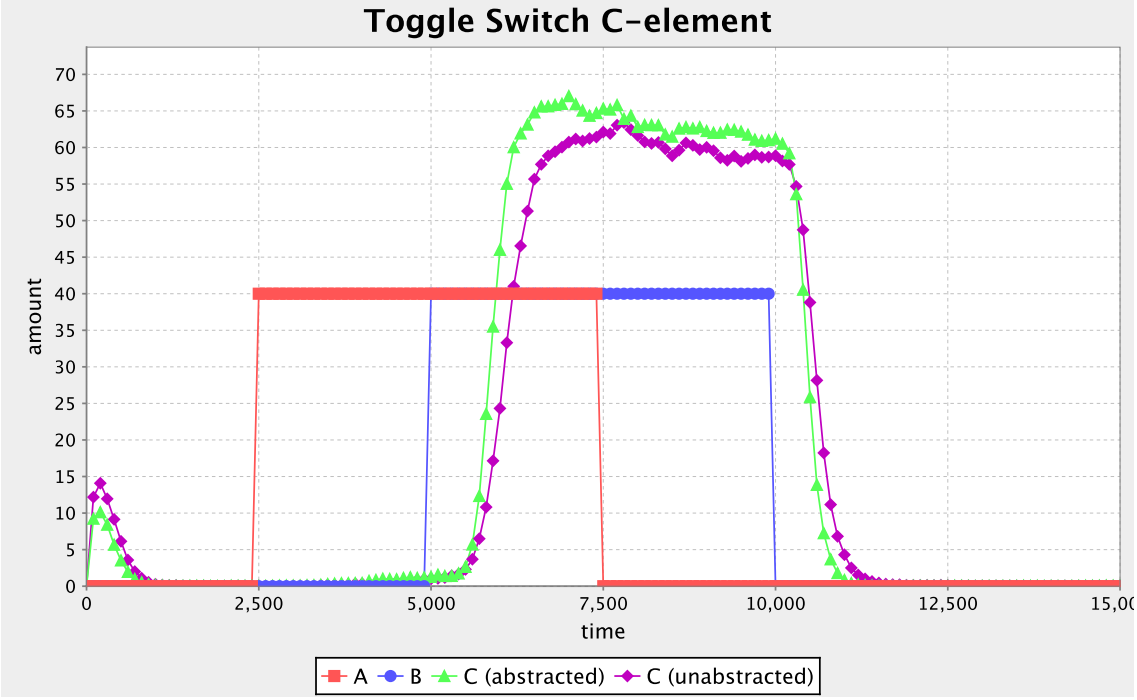


Figure 3.9. Simulation results for the genetic toggle Muller C-element.

3.1.3 Speed Independent Design

Our third design is based on the *unbounded gate delay model*. Under this model, circuits are guaranteed to work regardless of the delay at the output of each gate. The model also assumes that the delay on the wire is negligible. A circuit designed using this model is a *speed-independent* (SI) circuit. Speed-independent circuits tend to be more complex, but result in more robustness by avoiding timing hazards. The SI Muller C-element shown in the logical diagram in Fig. 3.10(a)[27] and the GCM is shown in Fig. 3.10(b). The GCM for the SI design contains 10 species and 14 influences. The circuit is constructed from an OR gate, three 2-input NAND gates, and an AND gate. The state holding element results from the P2 and P3 NAND gates feeding into each other, forming a toggle element. When both A and B are low, then P1 is low, and P4 is high. This results in P2 going to high and P3 going to low, setting the output C to low. When both A and B are high, then P1 is high. The value of P4 is dependent on the value of P2. If P2 is high, then P4 is set to low. This results in P3 going to high which causes P2 to go low. If P2 is low, then P3 and P4 are set to high. The end result is that P3 and P4 always go to high when both A and B are high. When the signals are mixed, the values of P2 and P3 do not change, so the output C does not change.

The genetic circuit schematic is shown in Fig. 3.10(c). The OR is implemented by inverting the inputs to a NAND gate. In this case, the species A and B repress X and Y, respectively. Both X and Y repress production of P1. If A or B is high, then P1 is high. The AND is implemented by inverting the output of a NAND. In this case, if both P3 and P4 are present, then no Z is produced, which results in the level of C going high. If P3 or P4 is present, then Z is produced, which represses production of C. The toggle element can be seen in that P2 represses P3, and P3 represses P2, and that P2 and P3 both have a source outside of the toggle element that can be used to set and reset the element.

The generated SBML before and after abstraction is shown in Fig. 3.11. While the labels have been removed, Fig. 3.11 shows the reduction in the SBML model. Before abstraction, the SBML contains 41 species and 38 reactions. After abstraction, the SBML contains 10 species and 18 reactions. The average simulation results for 1,000 stochastic simulations in Fig. 3.12 shows the behavior of the circuit. Initially, both inputs are low, so C remains low. At 2500 seconds, A goes high which causes C to rise slightly. At 5000 seconds, B also goes high, which causes C to rise to its high level. At 10000 seconds, A

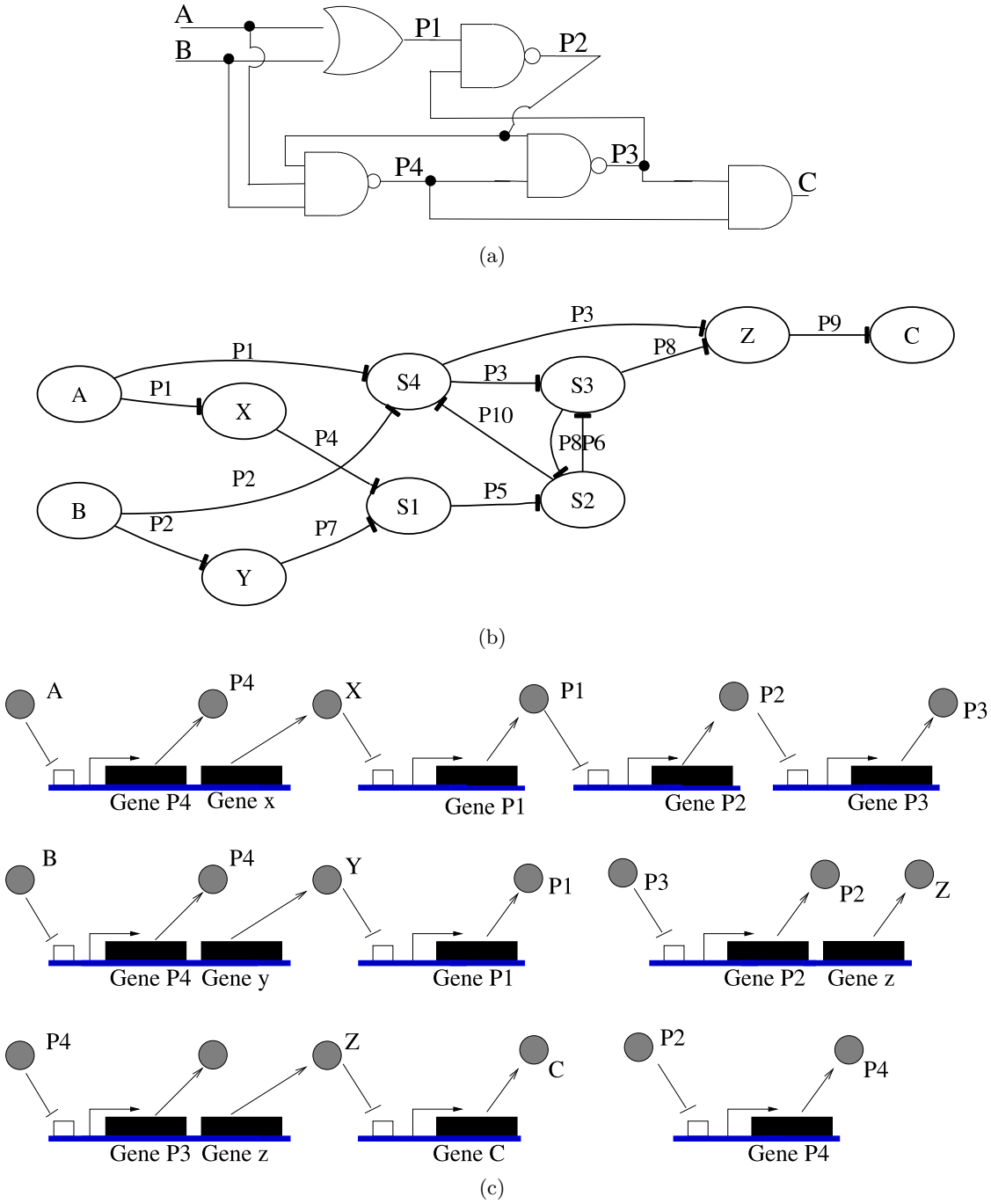
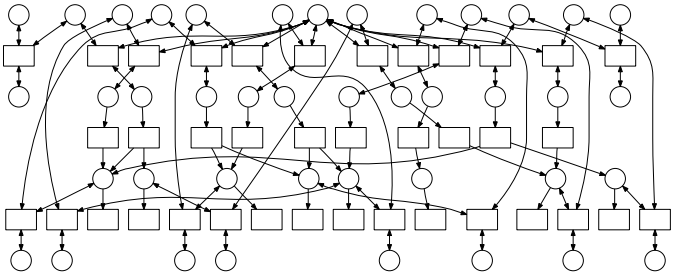
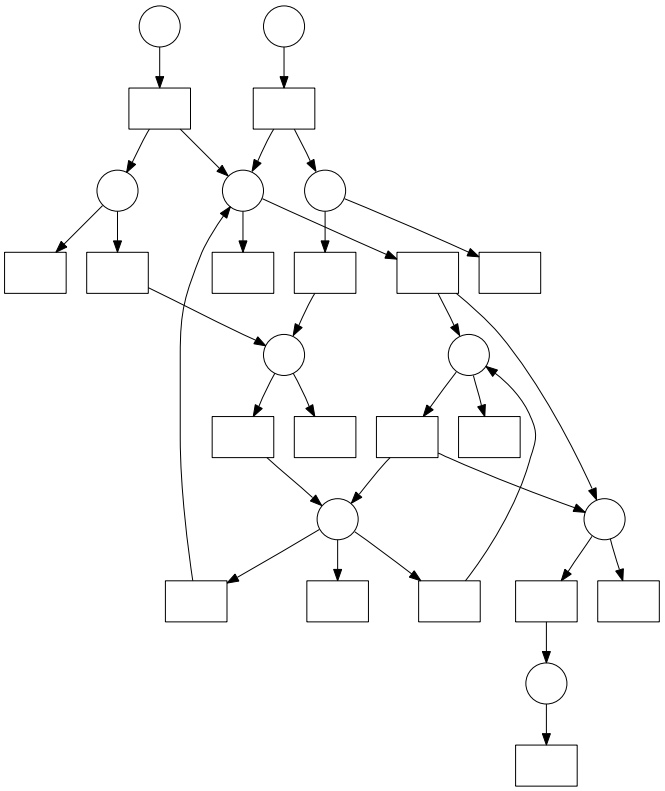


Figure 3.10. Genetic SI C-element designs. (a) Logical model, (b) GCM, and (c) genetic circuit for the genetic SI C-element.



(a)



(b)

Figure 3.11. SBML (a) before abstraction and (b) after abstraction for the genetic SI C-element.

is removed, but output C remains high. Finally, when B is removed, C drops back down. The simulation results show that the speed independent design also behaves correctly. Compared to the majority and the toggle gate, the SI gate has a slight delay before the gate switches from low to high and vice versa. The simulation time for abstracted versus unabstracted is 21 minutes versus 2 hours and 41 minutes. The simulation results also show that the abstracted and unabstracted results agree quite well.

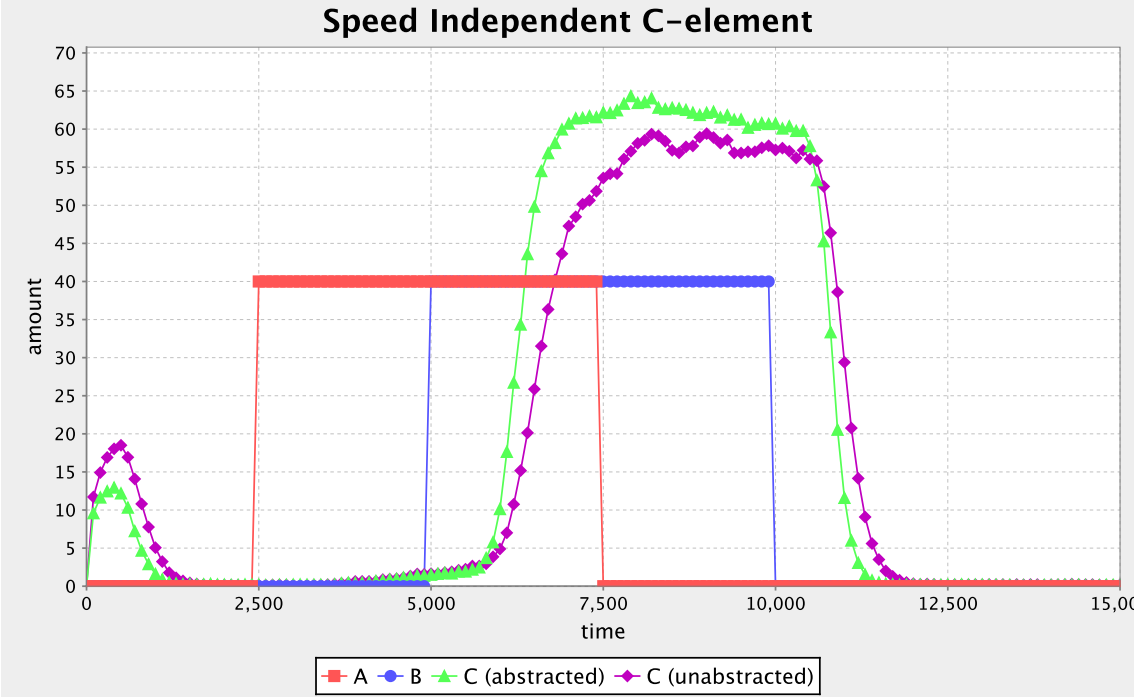


Figure 3.12. Simulation results for the SI genetic Muller C-element.

3.2 Mathematical Analysis

While the three Muller C-elements are logically equivalent, each gate consist of different designs. The simulation results for all three Muller C-element designs have similar behavior. In order to perform a more detailed analysis, this section models each circuit as a system of ordinary differential equations and analyzes the steady state behavior of the circuit.

In our simulations, we model the binding of RNAP to form the open complex, the production of proteins from the open complex, the decay of proteins, and the binding of the repressor molecule to the operator site to repress production. The ODEs for these events are listed below. Equation 3.1 tracks open complex formation, where OC is the number of open complexes, n_r is the number of RNAP, K_o is the equilibrium constant for the binding and unbinding of RNAP, n_g is the number of promoters for a given gene, OC is the number of open complexes formed from the promoter.

$$\frac{d[OC]}{dt} = K_o[n_r][n_g] - [OC] \tag{3.1}$$

Equation 3.2 tracks protein count, where $Protein$ is the number of protein molecules in the system, n_p is the number of transcripts per mRNA strand, k_o is the rate of mRNA production, OC is the number of open complexes, and k_d is the decay rate of the protein.

$$\frac{d[Protein]}{dt} = n_g\, k_o\, [OC] - k_d[Protein] \tag{3.2}$$

Equation 3.3 tracks the number of repressed genes, where RP is the number of repressed genes, K_r is the equilibrium constant for the binding and unbinding of the repressor molecule, $Repressor$ is the number of repressor molecules present, n_c is the number of operator sites that need to be bound in order to repress production.

$$\frac{d[RP]}{dt} = (K_r[Repressor][n_g])^{n_c} - [RP] \tag{3.3}$$

Applying abstraction methods as detailed by Kuwahara [26], we can reduce Equations 3.1-3.3 to one equation that describes the rate of change of protein count:

$$\frac{d[Protein]}{dt} = f([Repressor]) - k_d[Protein] \tag{3.4}$$

where $f([Repressor])$ is

$$f([Repressor]) = \frac{[n_p]k_o\, n_g\, K_o[n_r]}{1 + K_o[n_r] + (K_r[Repressor])^{n_c}} \tag{3.5}$$

Equation 3.4 can be used to graph the *nullclines* of the circuit in a *phase plane diagram*. Nullclines are lines on a graph where the derivatives are zero. A phase plane

diagram is a graph of the nullclines. Intersection of nullclines represents an equilibrium point. Stable equilibrium points are marked with a black circle in the diagrams and unstable equilibrium points are marked with an open circle. Analysis of the nullclines show whether or not the circuit has the correct behavior and also gives a relative idea on the robustness of the circuit. Each circuit should have exactly one stable state when both inputs have the same value. If the circuit has more than one stable state, then there is a possibility to be in the incorrect state. When the inputs are mixed, each circuit must have two stable states separated by an unstable state. This is because the circuit must be able to take on either a high or low state when the input signals are mixed. The separation distance between the stable states and the unstable state affect the robustness of the circuit. The larger the separation distance, the harder it becomes to move into an incorrect state.

3.2.1 Majority Gate Design

The majority gate can be modeled by Equations 3.6-3.11. The input is a and b, and the output is c. In Equations 3.6-3.8, there are two production terms. If two of the three inputs, a, b, or c is high, then one of x, y, or z is low, since there is little production. If x, y, or z is low, then one of the the production terms in 3.9 is near the full rate of production, resulting in a high value of d, a low value of e, and subsequently, a high value of c. Therefore, d and c are inevitably linked, and by examining the level of d, it is possible to determine the level of c.

$$\frac{d[x]}{dt} = f([a]) + f([b]) - k_d[x] \tag{3.6}$$

$$\frac{d[y]}{dt} = f([a]) + f([d]) - k_d[y] \tag{3.7}$$

$$\frac{d[z]}{dt} = f([b]) + f([d]) - k_d[z] \tag{3.8}$$

$$\frac{d[d]}{dt} = f([x]) + f([y] + f([z]) - k_d[d] \tag{3.9}$$

$$\frac{d[e]}{dt} = f(d) - k_d[e] \tag{3.10}$$

$$\frac{d[c]}{dt} = f(e) - k_d[c] \tag{3.11}$$

Applying steady state assumptions, we find y and z are both functions of a, b, and d. We also find that x is a constant value that is set by the initial value of a and b. Setting $\frac{d[d]}{dt}$ to zero, we can rewrite Equation 3.9 as

$$0 = g([a],[b],[d]) - k_d[d] \tag{3.12}$$

where $g([a],[b],[d])$ is a composition of the functions for x, y, and z. While we cannot explicitly find a solution, we can find the equilibrium points by plotting a phase plane diagram shown in Fig. 3.13-3.15. The equilibrium points are where $g([a],[b],[d])$ crosses $\frac{d[d]}{dt} = 0$. When both inputs are low or both are high, there is exactly one intersection. When the inputs are mixed, there are three intersections, two are stable points, and one is unstable. However, there is very little separation between the low stable state and the unstable state. This implies that it would be easy for the circuit to flip between the low to the high state. The second important result from our analysis is that the system uses only one species to maintain state. If species D has a spontaneous degradation or production, the circuit can easily change into the incorrect state.

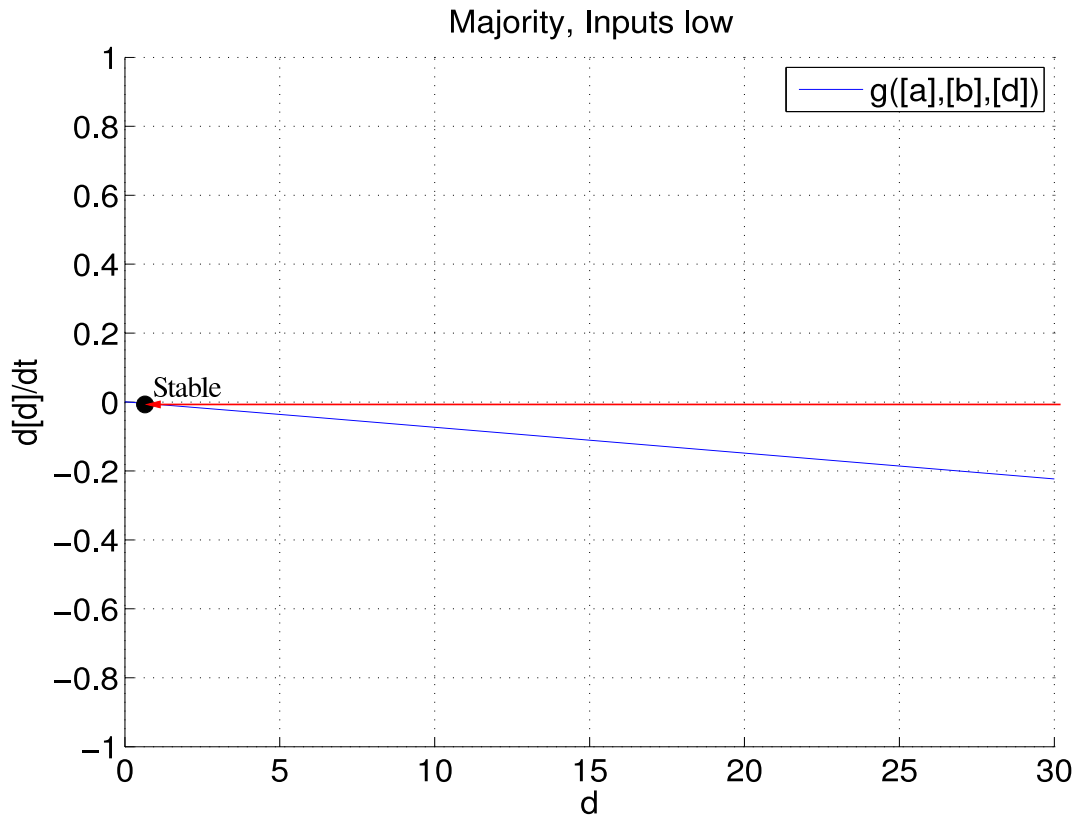


Figure 3.13. Phase plane diagram for the genetic majority C-element when inputs are low.

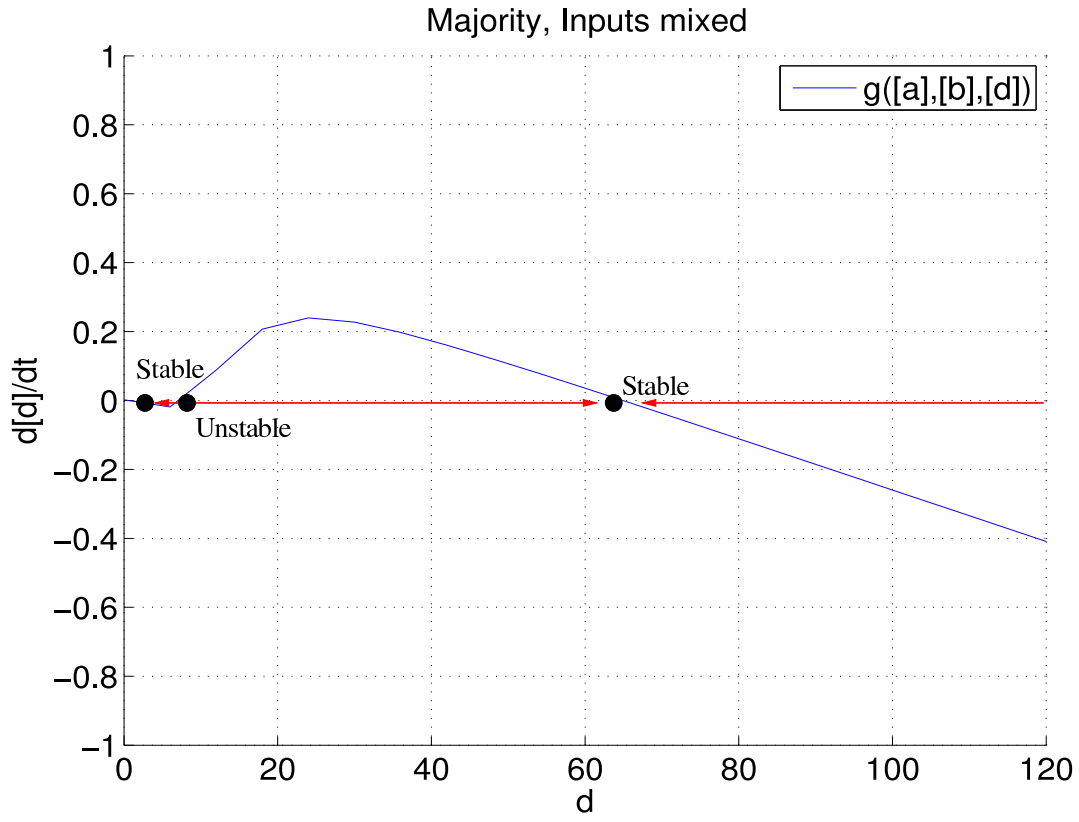


Figure 3.14. Phase plane diagram for the genetic majority C-element when inputs are mixed.

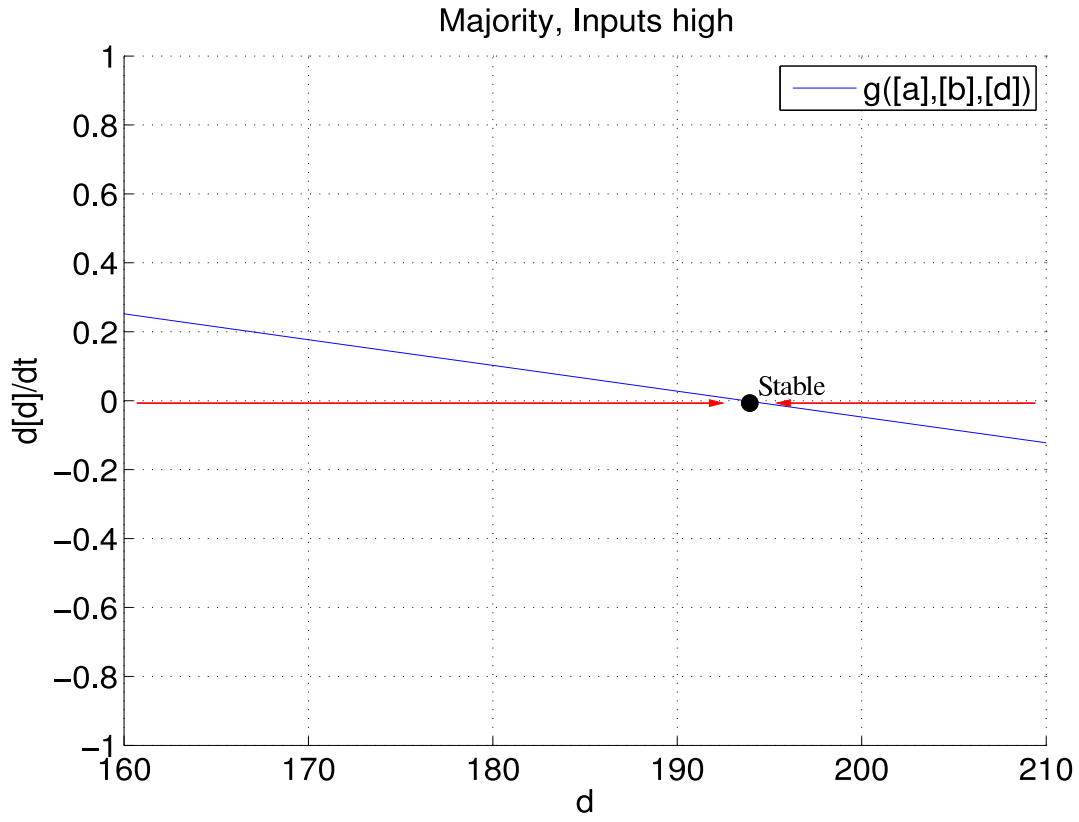


Figure 3.15. Phase plane diagram for the genetic majority C-element when inputs are high.

3.2.2 Toggle Switch Design

The toggle switch design can be modeled by Equations 3.13-3.19. The input is a and b, and the output is c. The toggle element is described by Equations 3.17-3.18, as the production rate of y is repressed by z and the production rate of z is repressed by y. Equations 3.13-3.15 describe the AND gate. Species a and b repress the production of d and e, respectively. When a and b are at low levels, then d and e are at high values, which combine together to produce f, as seen in Equation 3.15. A high value of f causes one of the production terms in Equation 3.18 to be near full production. This resets the toggle. If a and b are at high values, then the production terms in 3.16 are low, resulting in a low value of x. This makes the one production term in Equation 3.17 be near full production rate, causing the y value to increase. This sets the toggle. Since the level of c can be tracked by the level of y, only the nullclines for y and z need to be plotted to determine whether c is in a low or high state.

$$\frac{d[d]}{dt} = f([a]) - k_d[d] \tag{3.13}$$

$$\frac{d[e]}{dt} = f([b]) - k_d[e] \tag{3.14}$$

$$\frac{d[f]}{dt} = f([d]) + f([e]) - k_d[f] \tag{3.15}$$

$$\frac{d[x]}{dt} = f([a]) + f([b]) - k_d[d] \tag{3.16}$$

$$\frac{d[y]}{dt} = f([x]) + f([z]) - k_d[y] \tag{3.17}$$

$$\frac{d[z]}{dt} = f([f]) + f([z]) - k_d[z] \tag{3.18}$$

$$\frac{d[c]}{dt} = f([z]) - k_d[c] \tag{3.19}$$

Applying steady state assumptions, we find that y and z are both functions of a, b, y, and z. We can rewrite Equations 3.13-3.19 as:

$$[y] = g([a], [b], [y], [z]) \tag{3.20}$$

$$[z] = h([a], [b], [y], [z]) \tag{3.21}$$

The stable point of this system are found by graphing the nullclines of y and z from Equations 3.20 and 3.21 and finding the intersection of the nullclines. Fig. 3.16-3.18 shows the nullclines for the toggle switch design. When both inputs are low or both are high, there is exactly one stable point. When the inputs are mixed, there are two stable points and one unstable point. Unlike the majority design, there is a large separation

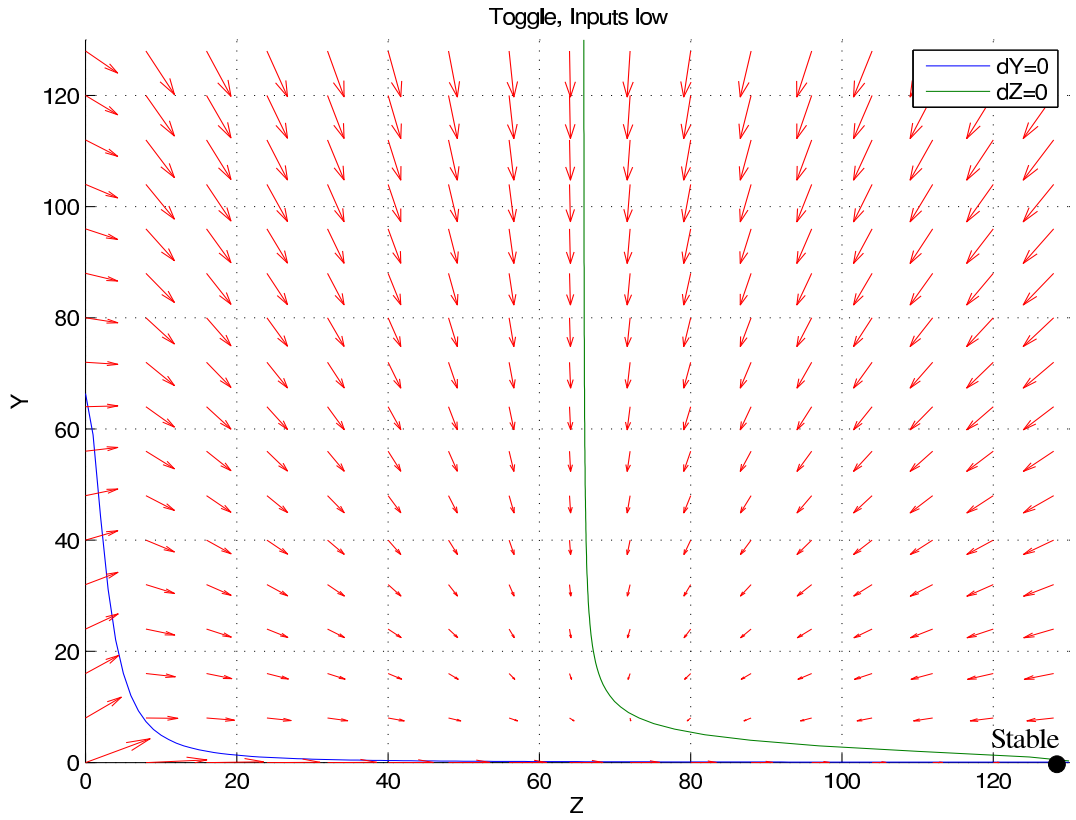


Figure 3.16. Phase plane diagram for the genetic toggle C-element when inputs are low.

between the stable states and unstable steady state. This means that it is harder to switch states, once locked into a stable state. Unlike the majority C-element design, the system depends on both Y and Z to hold state. In order to change state, there must be a change in both Y and Z, making the system more robust than the majority C-element.

3.2.3 Speed Independent Design

The speed independent gate can be modeled by Equations 3.22-3.29. The inputs are a and b, and the output is c. The OR gate is modeled in Equations 3.22, 3.23, and 3.25. When a and b are low, then x and y are high, which results in P_1 going to a low value. When either a or b is high, then x or y is low, resulting in one of the production terms in Equation 3.25 being near the full production value. The 3-input NAND is modeled by Equation 3.28. The three production terms show the inputs to the NAND gate, a, b, and P_2. If any of the inputs are low, then one of the production terms is near full production value, resulting in P_4 being produced. The AND gate can be seen in Equations 3.24 and

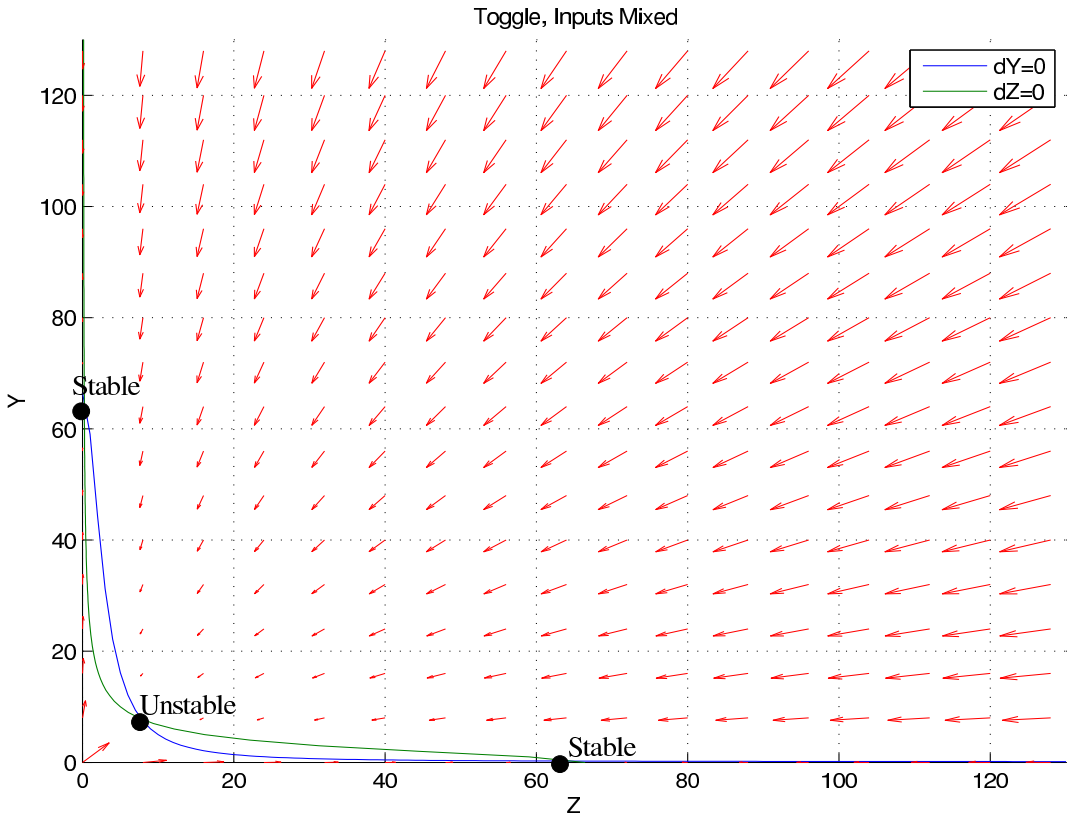


Figure 3.17. Phase plane diagram for the genetic toggle C-element when inputs are mixed.

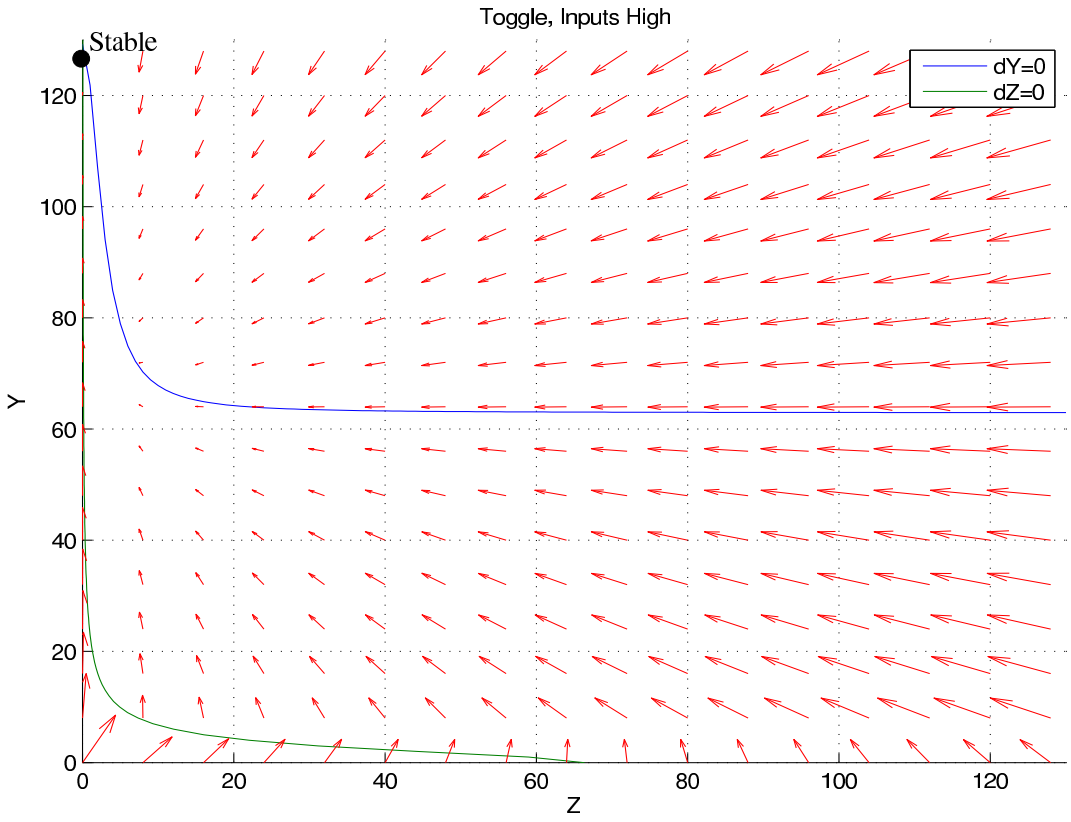


Figure 3.18. Phase plane diagram for the genetic toggle C-element when inputs are high.

3.29. Both P_3 and P_4 must be present to lower the production rates of z. Just as y and z in the toggle switch, P_2 and P_3 acts as the toggle element since P_2 lowers the production of P_3, and P_3 lowers the production P_2, as seen in Equation 3.26-3.27. By finding the values of P_2 and P_3, we can determine the value of c. The value of c closely follows the value of P_3, and so that when P_3 is low, c is also low. When P_3 is high, then c is also high.

$$\frac{d[x]}{dt} = f([a]) + -k_d[x] \tag{3.22}$$
$$\frac{d[y]}{dt} = f([b]) - k_d[y] \tag{3.23}$$
$$\frac{d[z]}{dt} = f([P_3]) + f([P_4]) - k_d[z] \tag{3.24}$$
$$\frac{d[P_1]}{dt} = f([x]) + f([y]) - k_d[P_1] \tag{3.25}$$
$$\frac{d[P_2]}{dt} = f([P_1]) + f([P_3]) - k_d[P_2] \tag{3.26}$$
$$\frac{d[P_3]}{dt} = f([P_2]) + f([P_4]) - k_d[P_3] \tag{3.27}$$
$$\frac{d[P_4]}{dt} = f([a]) + f([b]) + f([P2]) - k_d[P_4] \tag{3.28}$$
$$\frac{d[c]}{dt} = f([z]) - k_d[c] \tag{3.29}$$

Applying steady state assumptions, we find P_2 and P_3 are functions of a, b, P_2, and P_3. We can rewrite Equations 3.22-3.29 as:

$$[P_2] = g([a], [b], [P_2], [P_3]) \tag{3.30}$$
$$[P_3] = h([a], [b], [P_2], [P_3]) \tag{3.31}$$

The stable point of this system is found by graphing the nullclines of P_2 and P_3 from Equations 3.30 and 3.31 and finding the intersection of the nullclines. Fig. 3.19-3.21 shows the nullclines for the speed independent design. When both inputs are low or both are high, there is exactly one stable point. When the inputs are mixed, there are two stable points and one unstable point. Unlike the majority design, there is a large separation between the stable states and unstable steady state. This means it would be harder to switch states, once locked into a stable state. Like the toggle C-element, the speed independent C-element uses two species, P_2 and P_3, to hold state. The results for this circuit are very similar to the toggle switch design, with the difference being that in the high state, there is a possibility of a bifurcation if the parameters are slightly

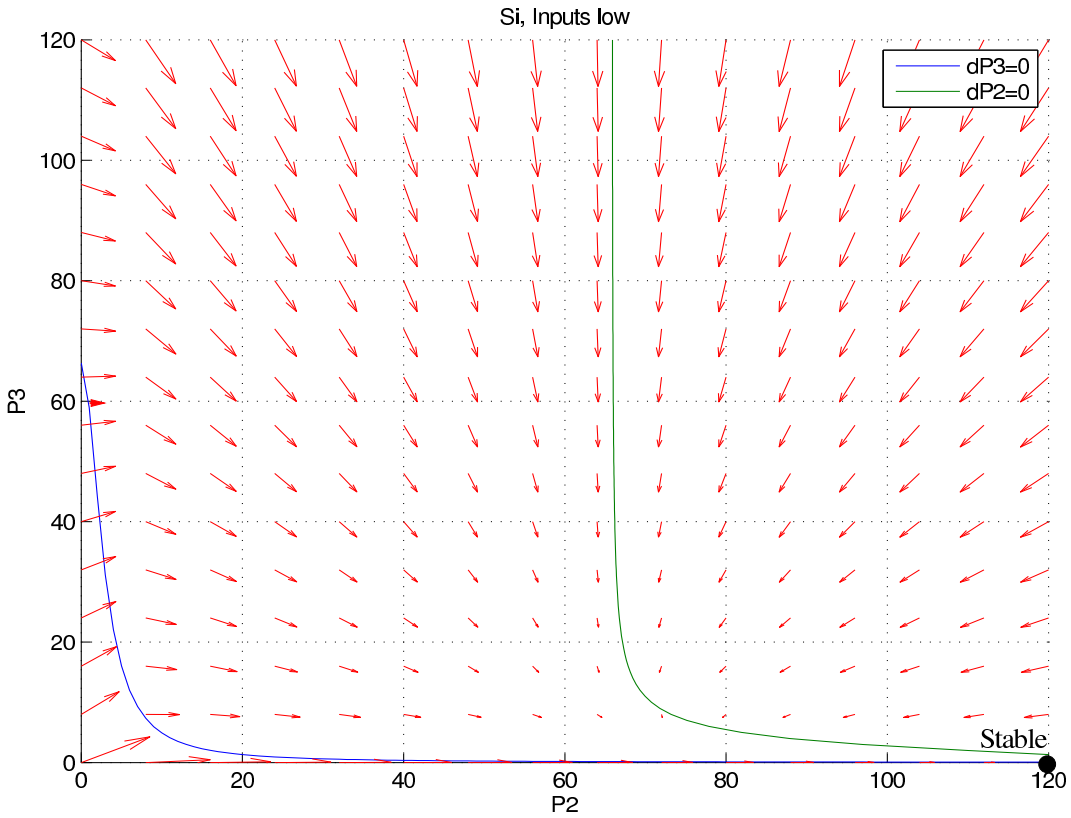


Figure 3.19. Phase plane diagram for the genetic SI C-element when inputs are low.

different. This would imply that the speed independent version might ironically be less robust than the toggle switch design, even though the gate has less timing assumptions than the toggle switch design.

3.3 Failure Rate Analysis Using Stochastic Simulation

According to our differential equation model, once a gate is locked into a stable state, the gate should remain in a stable state. However, the molecule counts in genetic circuits are extremely small, making the continuous-deterministic assumption inadequate. Genetic circuits are never going to be perfect as there is always a possibility that stochastic effects could accumulate to cause the circuit to fail. The robustness of the gates can be analyzed using stochastic simulations to track the failure rates of each gate. A gate fails when for a given input, the gate is in an incorrect state.

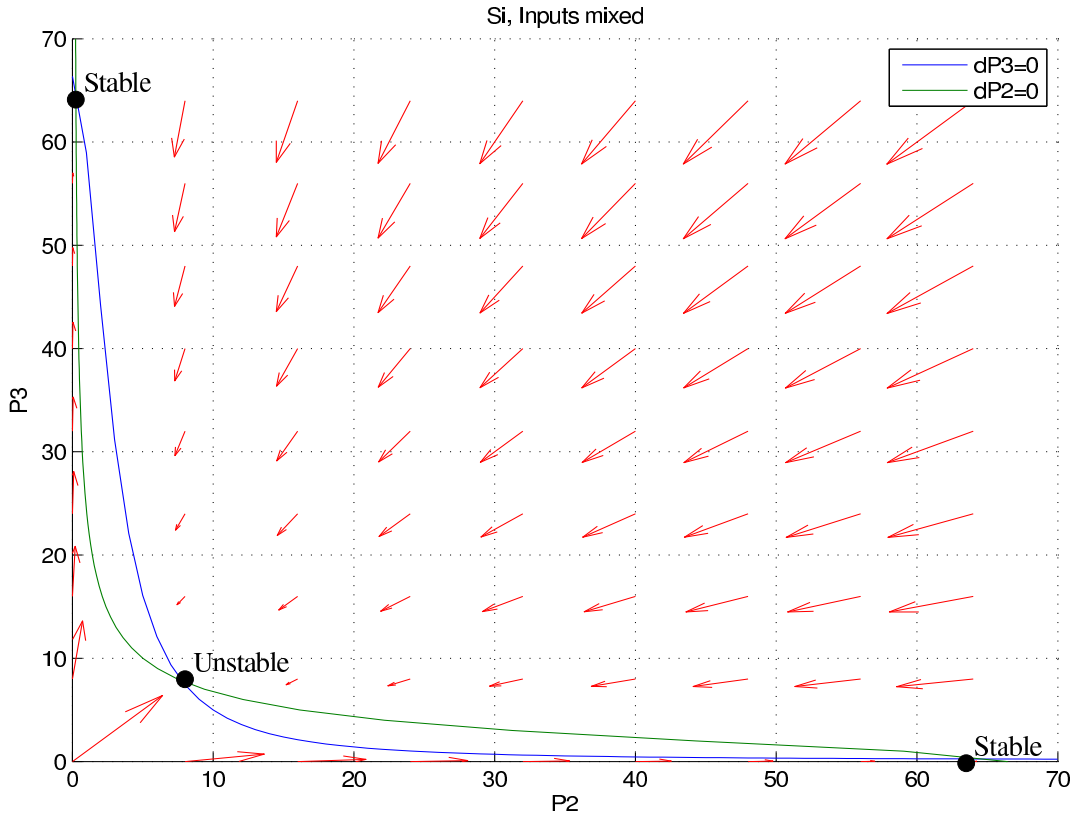


Figure 3.20. Phase plane diagram for the genetic SI C-element when inputs are mixed.

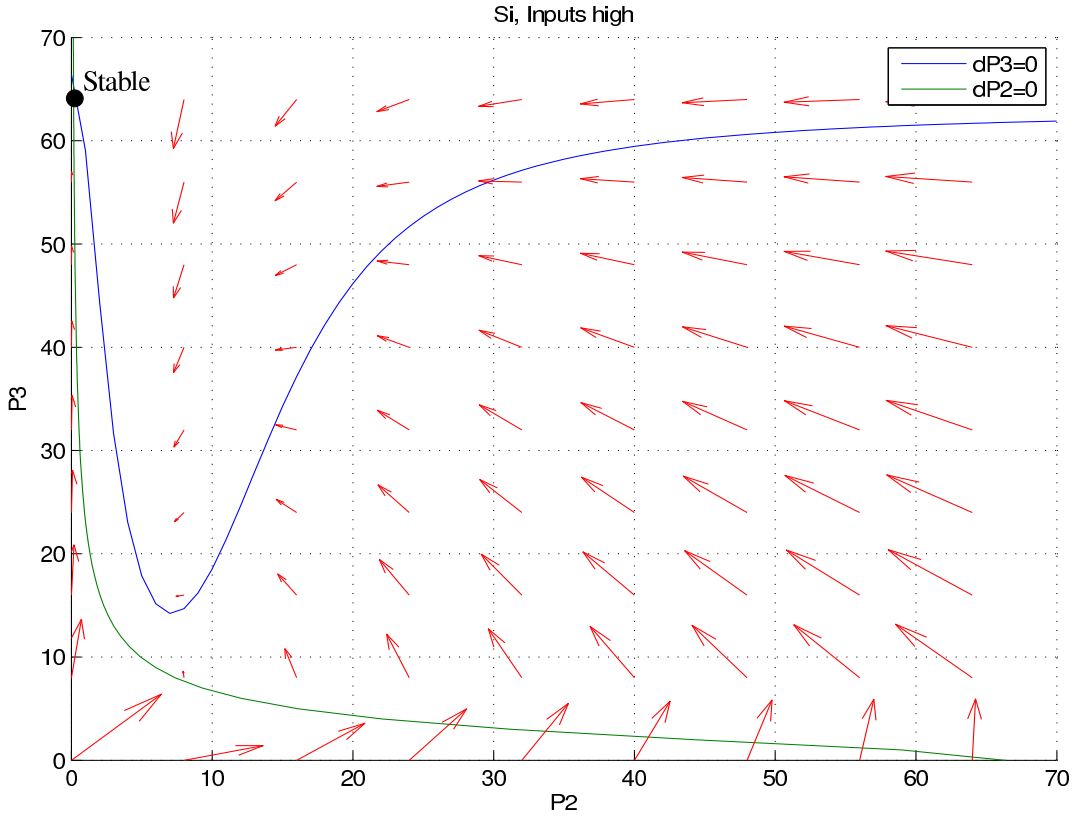


Figure 3.21. Phase plane diagram for the genetic SI C-element when inputs are high.

There are two possible methods for determining which species to track in determining the state of the gate. One possible method is measuring the level of the output species c. If the level of c is high when it should be low or vice versa, then the gate has failed. This method relies only on the output species, and is analogous to a *single rail system* in asynchronous design. It takes one signal to carry one bit of information. However, this method fails to account for the fact that there are other species in the system. For example, in the toggle gate, suppose that c should be high, and several spontaneous degradation reactions cause c to degrade to its low level. However, y is still high and z is still low, and this drives c back into its high region.

The second method for determining the state of the gate is to measure the levels of a set of key species. This method accounts for the possibility that the output species might not be representative of the actual state of the gate. For example, in the toggle gate, the toggle element reflects the state of the gate. The species in the toggle element are y and z. When y is high and z is low, then the gate is in the high state. When y is low and z is high, then the gate is in the low state. When y and z are in agreement, then the state of the gate is indeterminate. In order for a failure to occur, there must be failures in both signals, making failure much more rare. This is analogous to a *dual-rail system* in asynchronous design, where two signals are necessary to carry one bit of information. For the SI gate, P_2 and P_3 act as the toggle species. While the majority gate does not have a toggle element, we can create a dual-rail system by measuring the output c and the internal input signal e.

In order to determine the levels for the low and high state of each gate, we first set both inputs to high to find the average high value and both inputs to low to find the average low value. We divided the range into three equal regions, a low region, an indeterminate region, and a high region. We recorded a failure when all the state holding species move into the opposite region. We did not mark moving into the indeterminate region as a failure because there are other molecules in the system that may push the gate back into the correct region, but due to a stochastic effect, the output molecule dropped into the indeterminate value briefly.

There are several different experiments that can be run to check failure rates. For example, one can set both inputs to low, and measure the number of gates that switch to high after a set amount of time. Likewise, one can set both inputs to high, and measure the number of gates that switch to low after a set amount of time. Both of these

experiments are uninteresting, because as shown in the mathematical analysis, when both inputs have the same value, there is only one stable state. The output always tends to go to this stable state. The more interesting case is if the inputs are mixed since all the gates have two stable states. Failure is more likely to occur in this experiment because stochastic production and degradation may cause the gate to lock in the wrong state.

The failure rate is evaluated using 1,000 stochastic simulation runs under several conditions for the original majority gate C-element, the toggle switch C-element, and the speed independent C-element. The simulations are run for 2100 seconds, roughly one cell cycle for an *E. coli*.

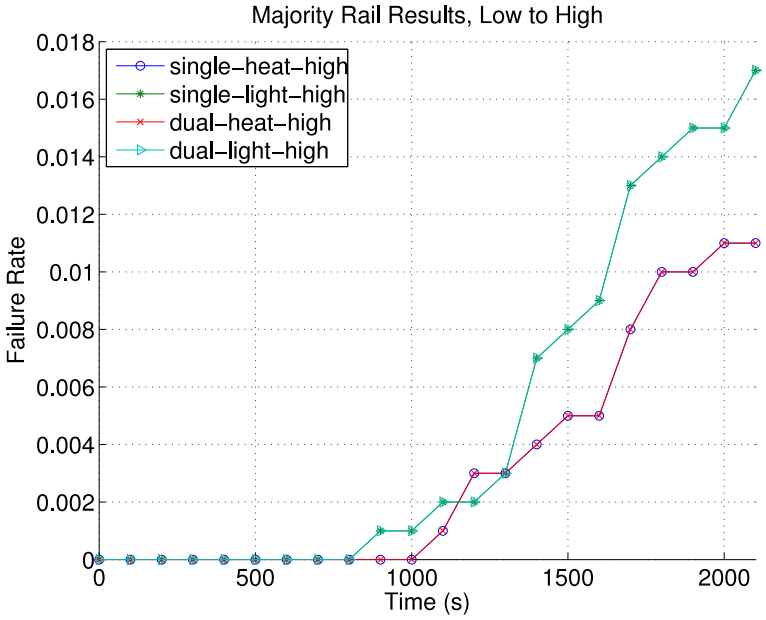
The failure rates for single-rail versus dual-rail for each gate are shown in Fig. 3.22-3.24. In every case, the dual-rail gate performs the same or better than the single-rail gate. The gain from the dual-rail system is much more pronounced in maintaining the high state, as seen by Fig. 3.22(b), 3.23(b), and 3.24(b). The results suggest that the output does not always accurately reflect the state of the gate, and that the gates can recover from error.

Next, consider only dual-rail outputs, let us compare the probability of failure of each genetic Muller C-element. In our first test, the C-element is put into its low state and either a or b is applied. If the C-element changes into its high state, then a failure is recorded. The results shown in Fig. 3.25 indicate that the gates seldom switch from low to high. The SI and majority gates fails at a slightly higher rate in this simulation, with one to two percent failing after one cell cycle, while the toggle gate performs the best.

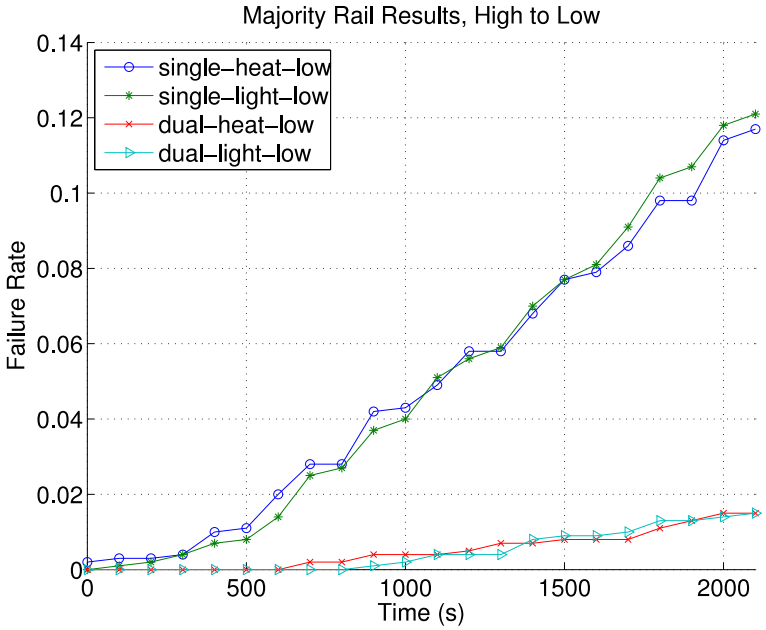
Next, the C-element is put into its high state and either a or b is removed. If the C-element changes into its low state, then a failure is recorded. The results shown in Fig. 3.26 indicate that all three gates can reliably hold the high state after an input has been removed. As with previous results, the toggle has the best performance, while the majority gate has the worst performance. These results suggest that a toggle element improves robustness. The results also suggest that timing assumptions and robustness are not correlated. The majority gate has the most timing assumptions, while the SI gate has the fewest timing assumptions. Ironically, the toggle gate outperforms both gates.

3.4 Parameter Variation Analysis

Our next set of simulations varies several parameters to see how they affect the failure rates and the switching time of the gates. We examine the effects of five parameters on the failure rates of C-elements: gene copy number, repressor strength, cooperativity of

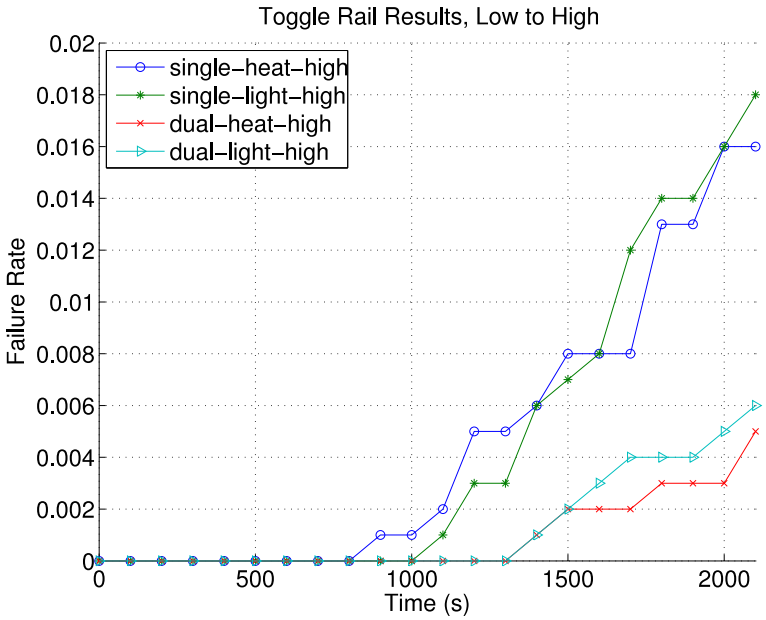


(a)

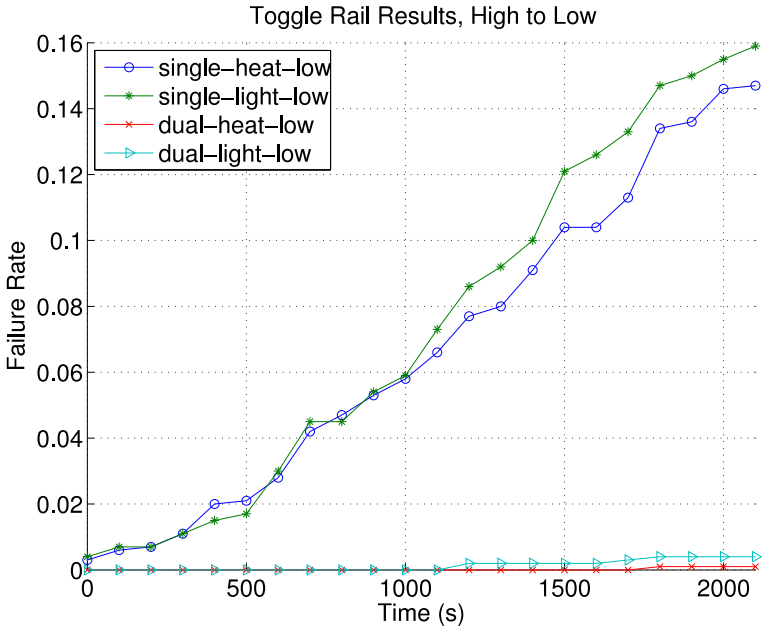


(b)

Figure 3.22. Probability of failure for single-rail versus dual-rail for the majority gate after (a) one of the inputs has been applied and (b) after one of the inputs has been removed.

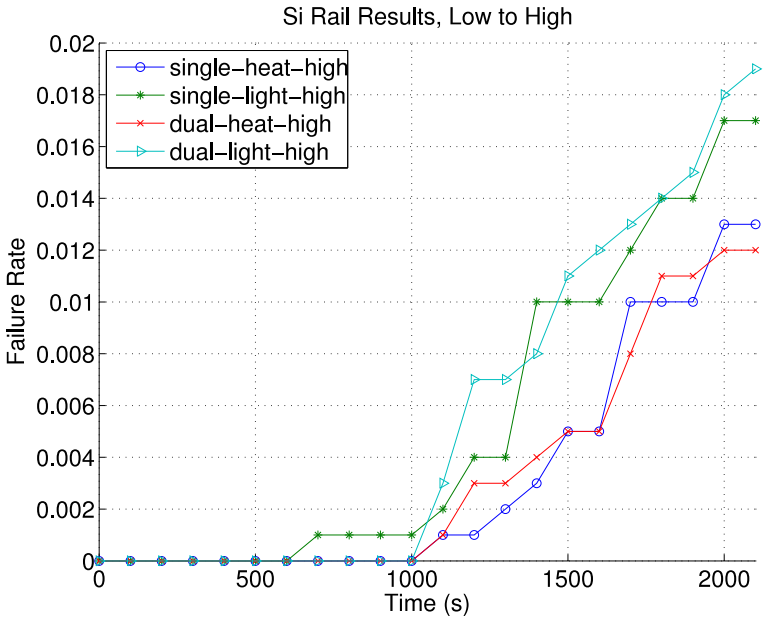


(a)

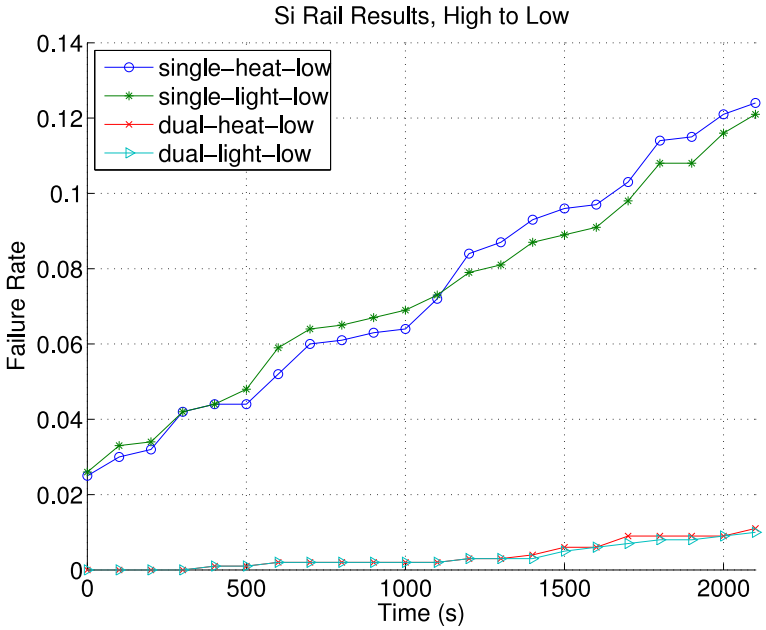


(b)

Figure 3.23. Probability of failure for single-rail versus dual-rail for the toggle gate after (a) one of the inputs has been applied and (b) after one of the inputs has been removed.



(a)



(b)

Figure 3.24. Probability of failure for single-rail versus dual-rail for the SI gate after (a) one of the inputs has been applied and (b) after one of the inputs has been removed.

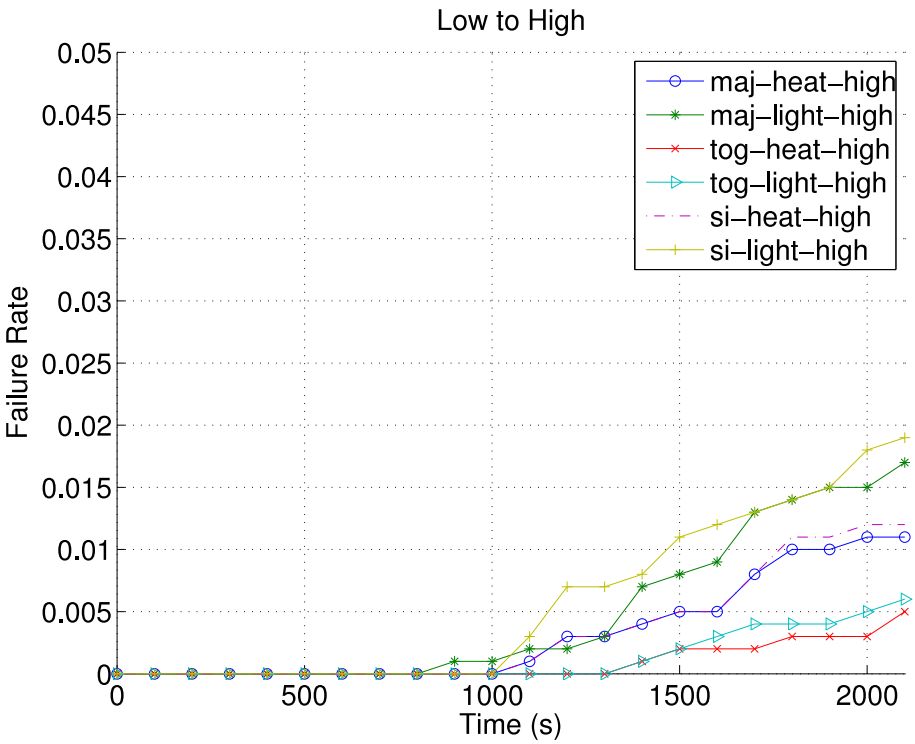


Figure 3.25. Probability of the C-element state to change from low to high after only one of the inputs has been applied.

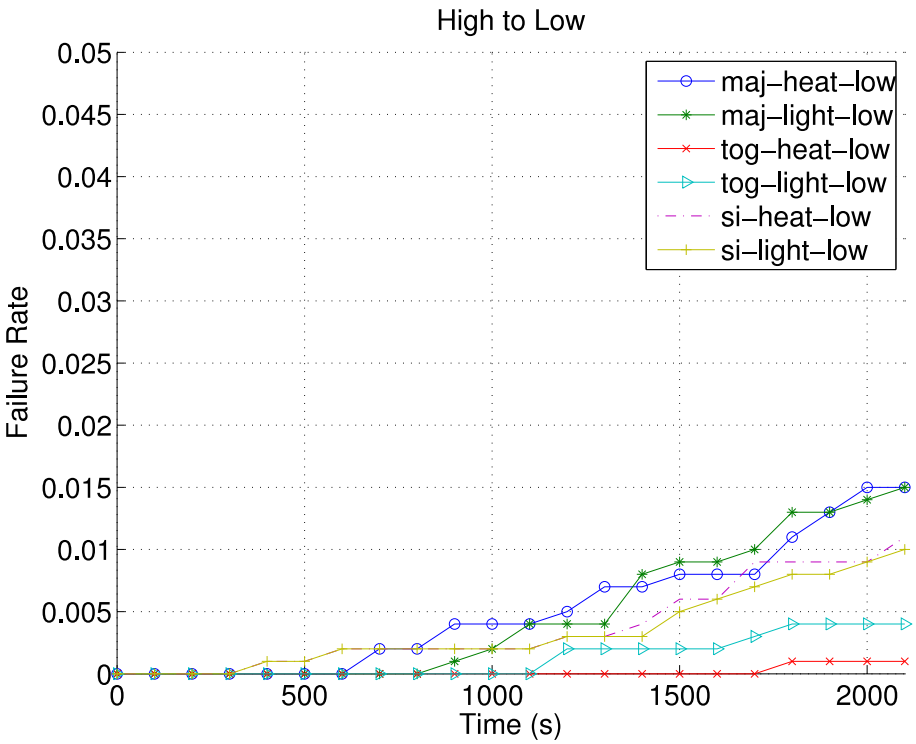


Figure 3.26. Probability of the C-element state to change from high to low after only one of the inputs has been removed.

repression, decay, and production to degradation ratio. While parameters are impossible to set to a specific number in genetic circuits, biologists do have some control of some of the parameters. By analyzing the effects of the parameter variation on failure rates, we can identify which parameters are vital for a robust C-element, and how the gradient of the parameter affects the failure rates. However, trade offs exist for each parameter choice. By changing the value of a parameter, this may affect the switching time of the circuit. We also present the switching time for each parameter choice. These results use 1,000 stochastic runs for each experiment and rates of failure are checked after one cell cycle.

3.4.1 Changing Gene Copies

First, gene copies are varied to see how increasing the number of genes affects the failure rates and switching time. When building and inserting a circuit into a bacteria, it is very rare to only insert one copy inside of a bacteria. It is more likely that the host organism receives several copies. We believe that more copies provide redundancy which helps prevent errors and lowers failure rates. The default value of the number of genes is 2. The experiments varied the number of genes from 1 to 5.

The results for the rates of failure are shown in Fig. 3.27 and 3.28. The results show that increasing gene copy number tends to reduce failure rates of all the Muller C-elements in all the experiments. The most gain comes from increasing the gene copy number from one to two. Fig. 3.29 shows the nullclines of the toggle gate for a gene copy value of 1 and 2. As the gene copy value increases, the separation between the two equilibrium points increase, making the toggle gate less likely to switch states. While not shown, the results for the majority and SI are similar.

The switching time for the gates are shown in Fig. 3.30. The results for the switching time show that increasing the gene copy number tends to increase the time to switch to the high state and tends to slightly decrease the time to switch to the low state. Several factors affect the switching time. Increasing the gene copy number changes the equilibrium point, requiring more time to produce species to reach the equilibrium. At the same time, the regions of both the high and low increase, and it may be possible that the increase in the low region is large enough that the switching time decreases to reach the low state.

This may be one avenue to explore when designing circuits. However, there is a tradeoff in that it may also put more stress on the host organism in making the extra

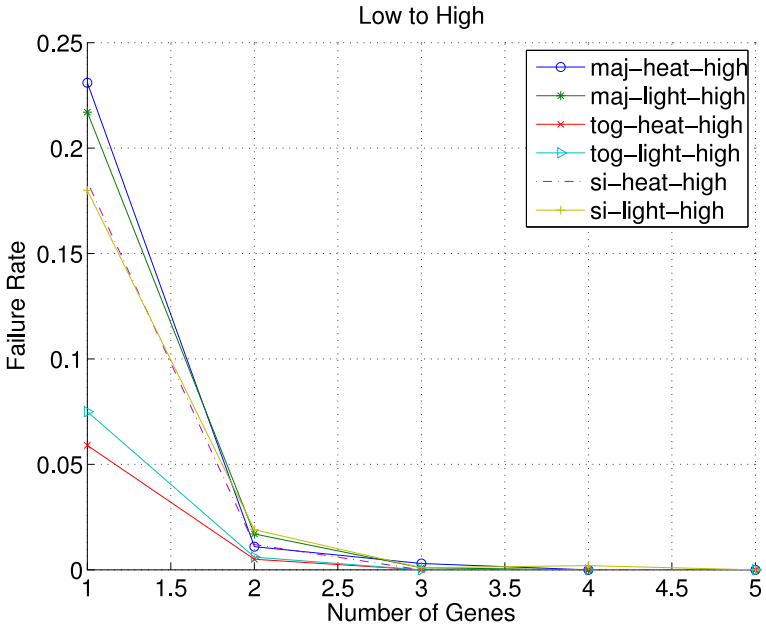


Figure 3.27. The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the number of gene copies in a circuit being varied between 1 to 5.

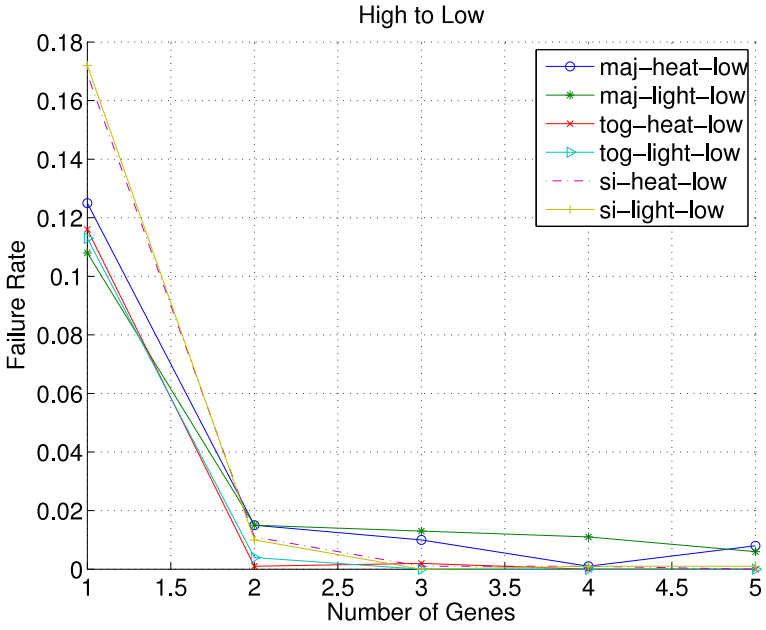
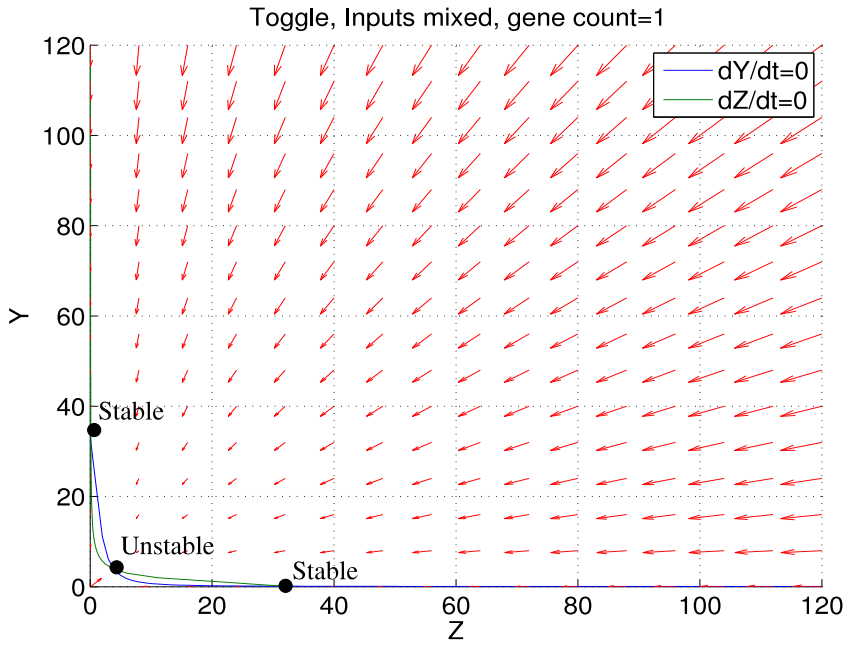
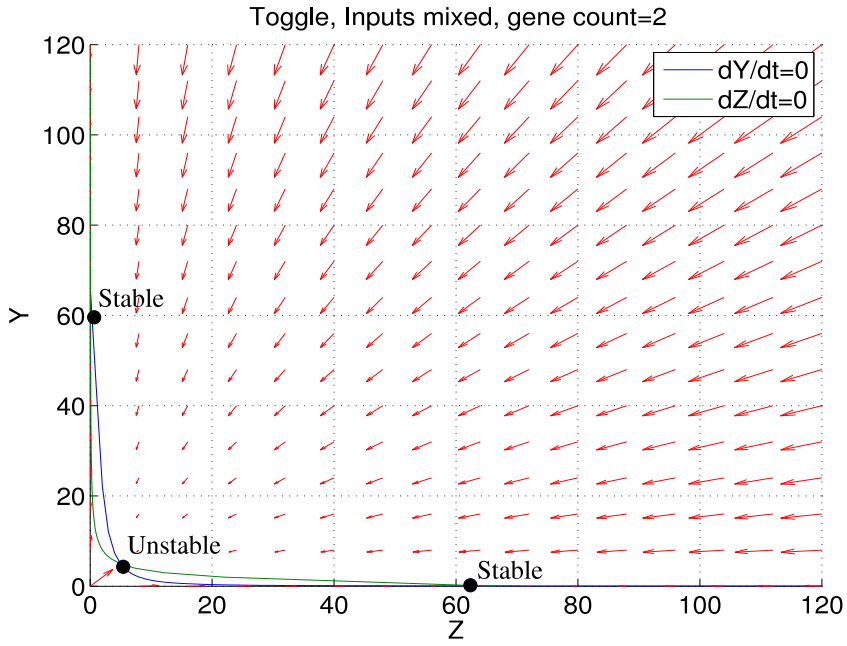


Figure 3.28. The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the number of gene copies in a circuit being varied between 1 to 5.



(a)



(b)

Figure 3.29. The nullclines of the toggle gate when both inputs are mixed, and the gene copy is (a) 1, and (b) 2. The results show that increasing gene copy increases the separation distance between the stable fixed points.

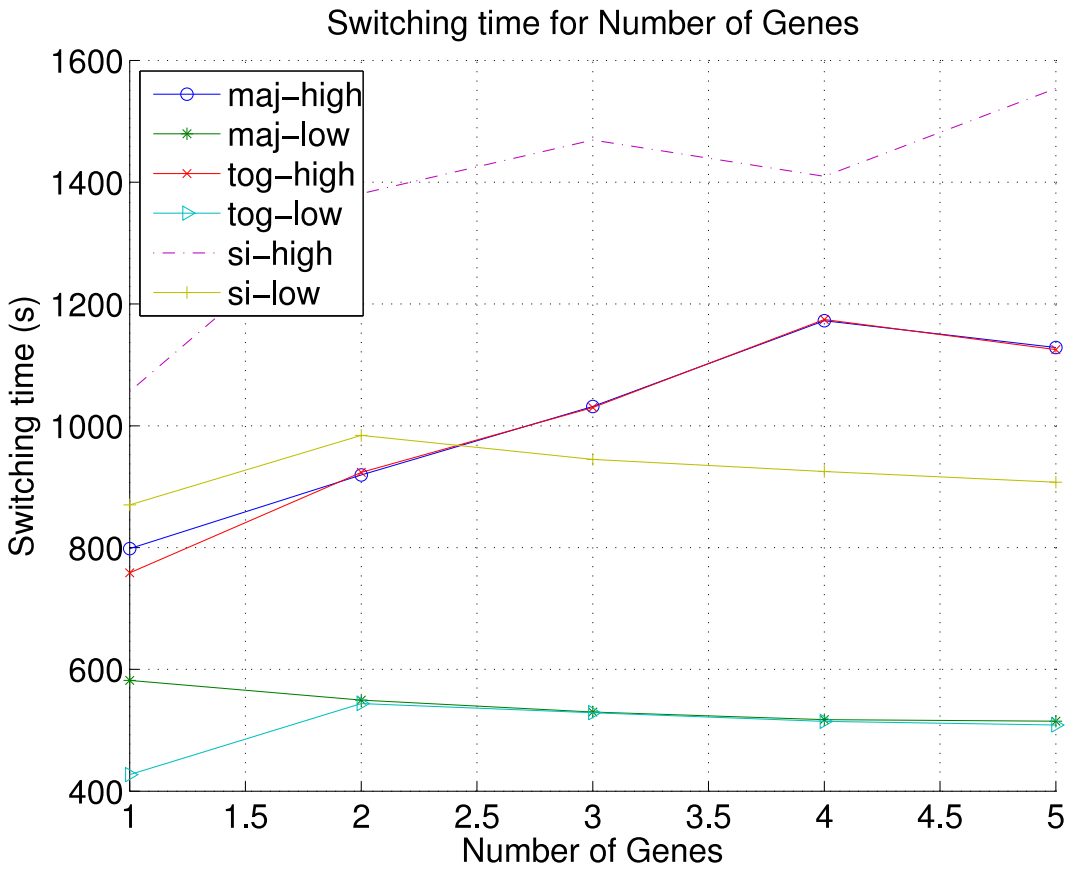


Figure 3.30. Switching time of the C-element, varying the gene count.

proteins. Increasing gene count may lower response time because more proteins are produced, requiring more time for them to degrade to change state and more protein to be produced to reach a high signal. The results show that the toggle gate outperforms the majority gate and SI gate. A gene copy value of two yields the most gain in robustness.

3.4.2 Changing Cooperativity

Second, cooperativity is varied to see how increasing the number of operator sites affects the failure rates and switching time. The value of cooperativity affects the steepness of the production curve compared with the number of repressor molecules present. This gives the gates a more switch like behavior. This can be seen in Fig. 3.31. The experiments varied the cooperativity from 1 to 5.

The results for the rates of failure are shown in Fig. 3.32 and 3.33. Increasing cooperativity reduces failure rates of all the Muller C-elements in all simulations. The results show that a cooperativity of two is necessary for a functional C-element. The reason can be seen by plotting the nullclines. Fig. 3.34 shows the nullclines of the toggle gate for a cooperativity of 1 and 2. When cooperativity is less than 2, there is only one stable point, and so the gate can only hold a high state or a low state when the inputs are mixed. The switching time for the gates are shown in Fig. 3.35. Changing the cooperativity has negligible effect on the switching time of the gates. A cooperativity value of three gives the most gain. However, cooperativity is difficult to engineer in genetic circuits, and so a default cooperativity value of two is selected.

3.4.3 Changing Repression Strength

Next, repression strength is varied to see how changing the value of the disassociation constant affects the failure rates and switching time. The strength of repression affects the number of molecules necessary to repress production. The higher the strength of repression, the less molecules of repressor is necessary to repress the production rate. This can be seen in Fig. 3.36. Each line represents repression strength.

Experiments are run with the strength of repression varied from 0.1 to 1.0. The default value of the strength of repression constant is 0.5. The results for the rates of failure are shown in Fig. 3.37 and 3.38. The results show that increasing repression strength decreases failure rates for all C-elements in all experiments. The reason can be seen by plotting the nullclines. Fig. 3.39 shows the nullclines of the toggle gate and SI gate for the repression strength of 0.1. The nullclines show that the SI gate has only one

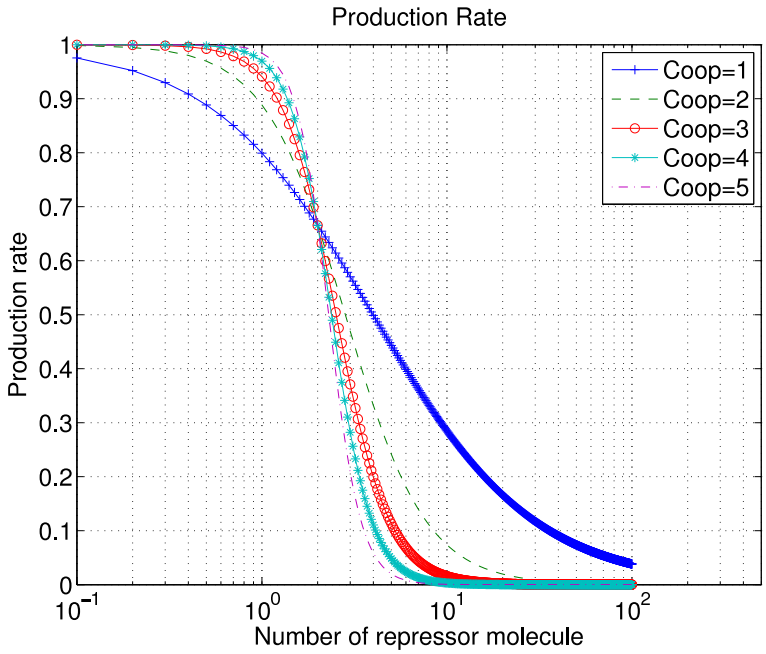


Figure 3.31. The effects of changing cooperativity on the production rate versus the number of repressor molecules. As the cooperativity increases, the steepness of the production curve increases.

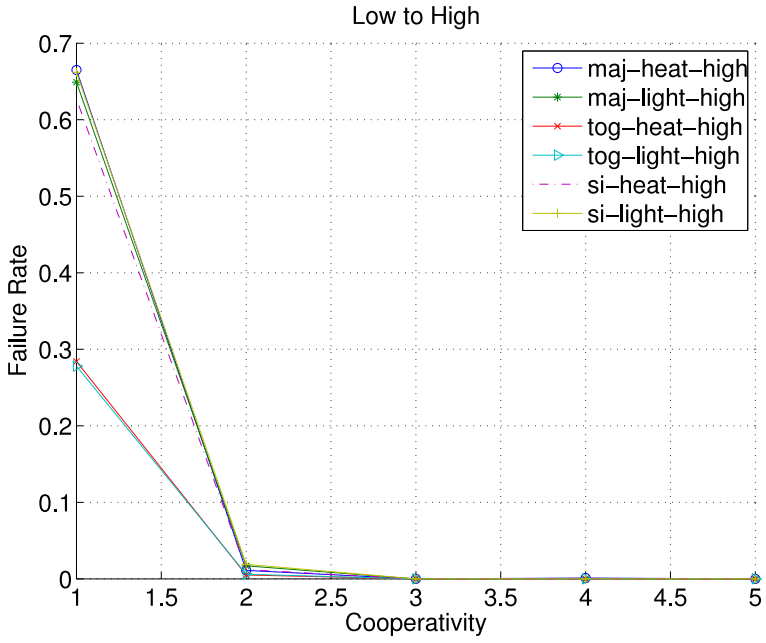


Figure 3.32. The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the cooperativity in a circuit being varied between 1 to 5.

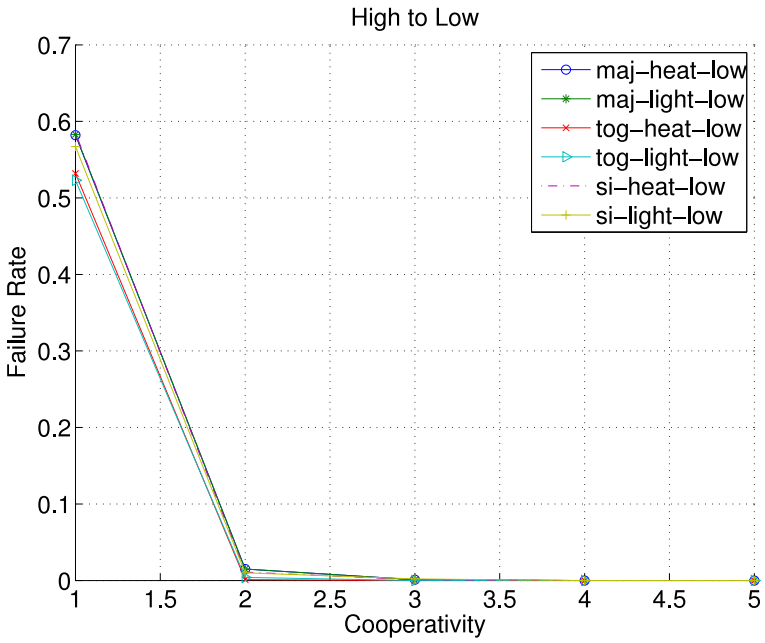


Figure 3.33. The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the the cooperativity in a circuit being varied between 1 to 5.

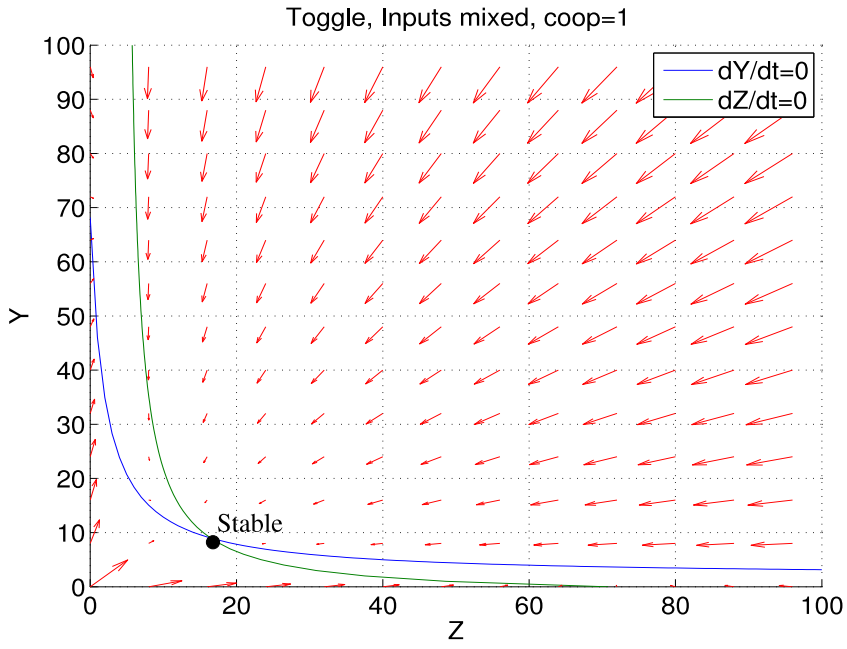
stable point, while the toggle gate has two stable points. This results explains why the toggle gate outperforms the SI gate when repression strength is low.

The switching time for the gates are shown in Fig. 3.40. As repression strength increases, the switching time for the gate increases. In order for the gate to switch, more repressor molecules must degrade away. If repression strength is extremely high, then most repressor molecules must degrade away before production can begin.

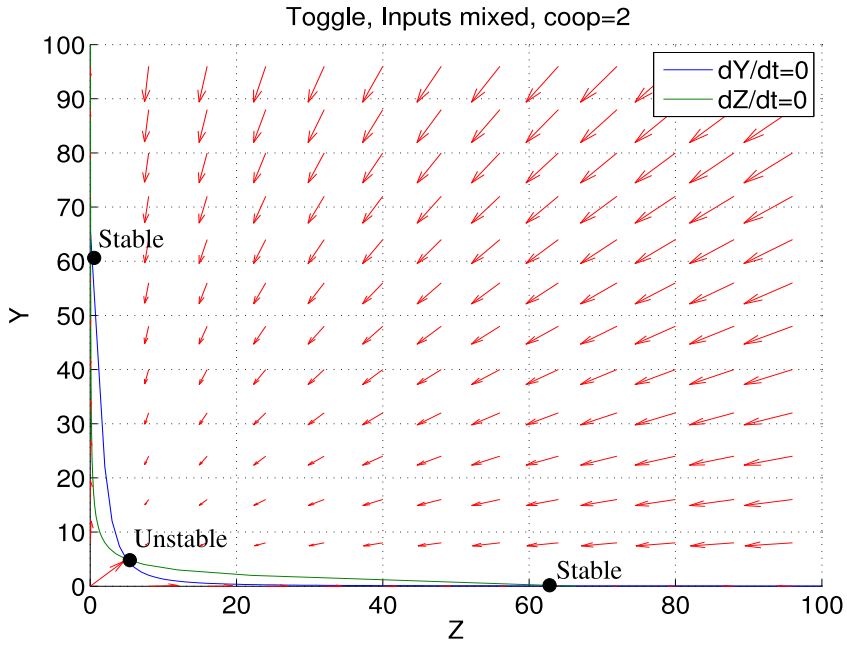
The results suggest that strong repression is necessary for a robust gate. A repression strength of one yields the most gain in robustness, but may be unrealistic in practice. If repression is set at one, then one molecule decreases production by almost half. Setting repression to 0.5 gives tolerable failure rates, while keeping the repression constant at a reasonable value.

3.4.4 Changing Decay Rates

Next, decay rates of the proteins are varied to see the affects on the failure rates and switching time. Decay rates are one parameter where biologists have the ability to tune and control by adding on special tags on the end of proteins to affect decay rates. For



(a)



(b)

Figure 3.34. The nullclines of the toggle gate when both inputs are mixed, and the cooperativity is (a) 1, and (b) 2. The results show a cooperativity of at least two is necessary for there to be two stable states.

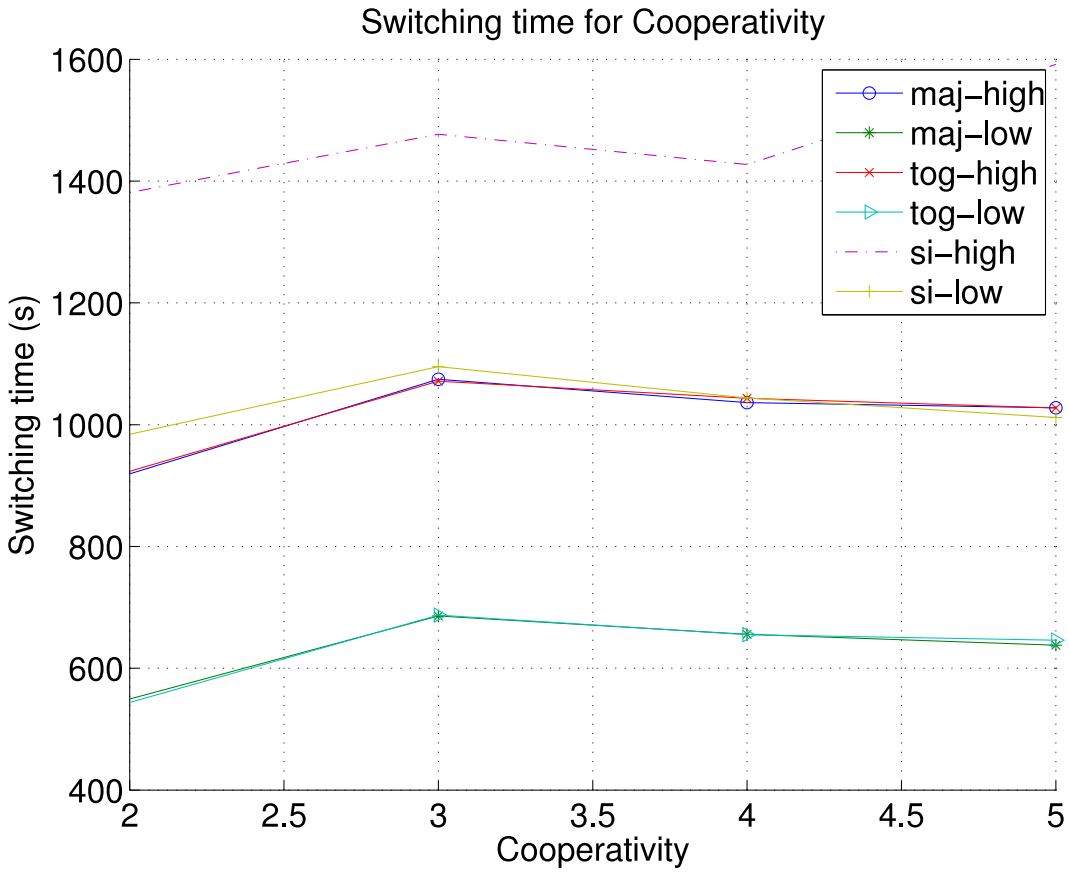


Figure 3.35. Switching time of the C-element, varying the cooperativity constant.

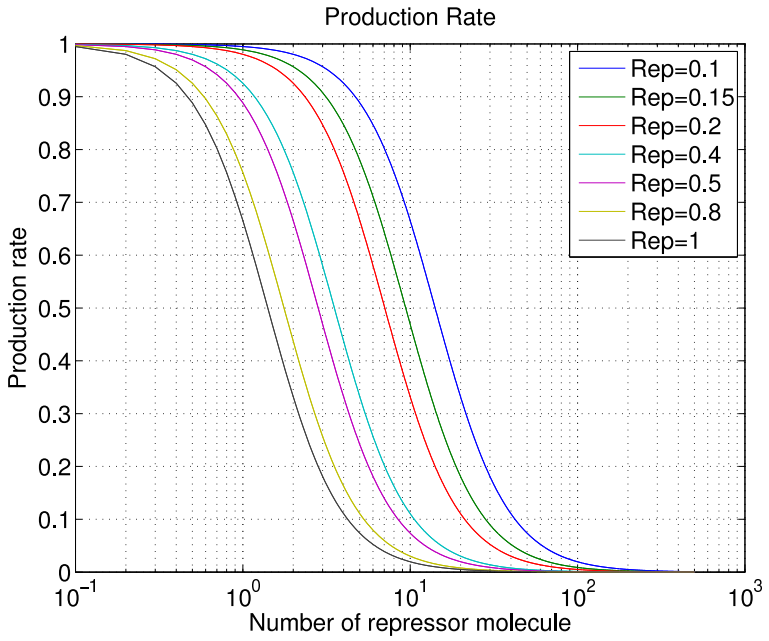


Figure 3.36. The effects of changing repression strength on the production rate versus the number of repressor molecules. As the repression strength increases, the less repressor molecules is necessary in order to get the same level of repression.

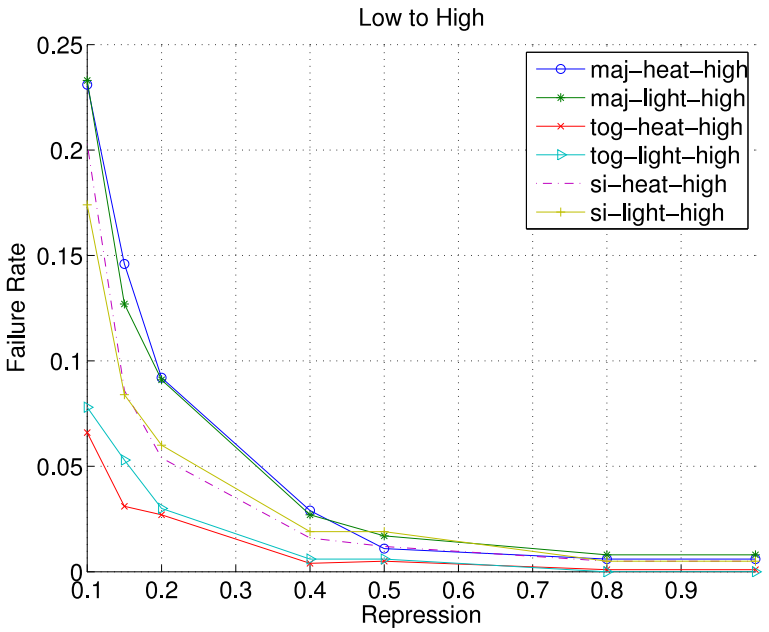


Figure 3.37. The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the repression constant being varied between .1 to 1.

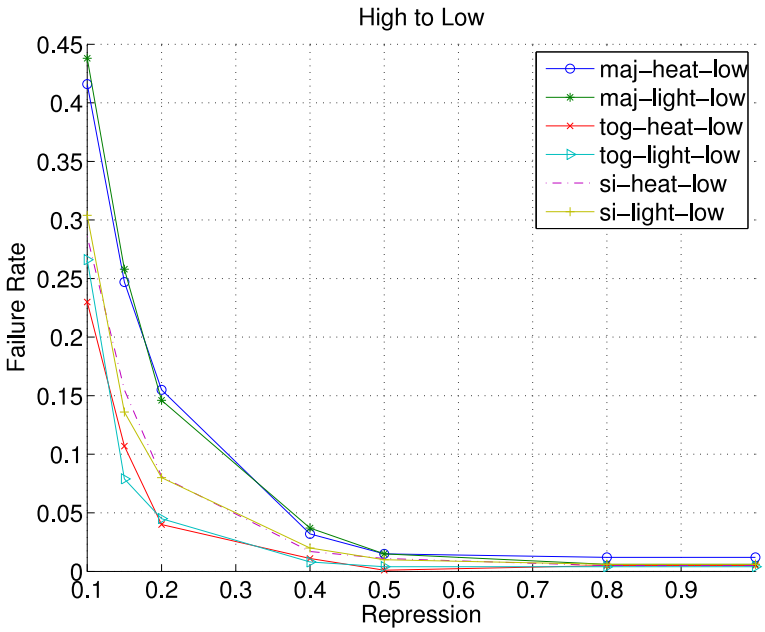
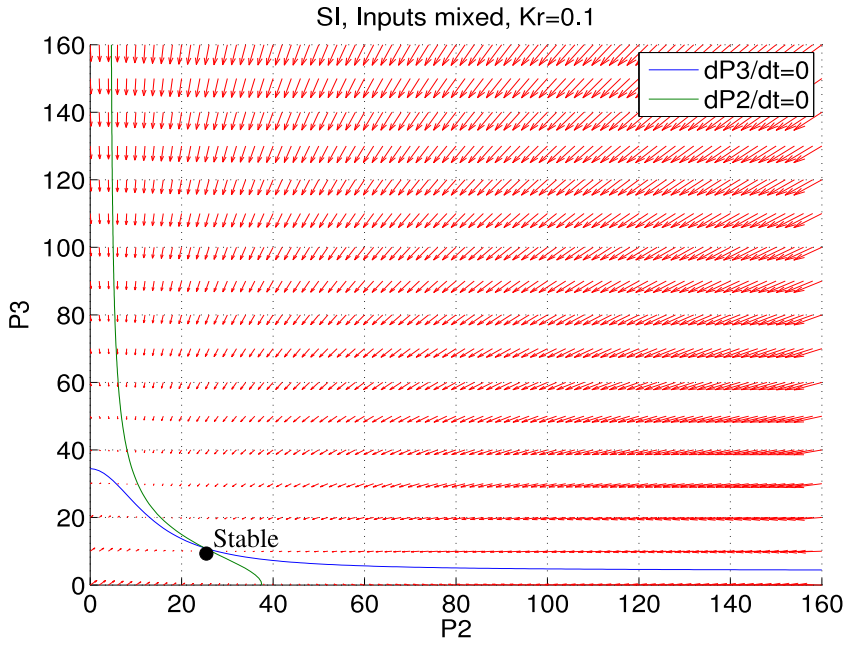


Figure 3.38. The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the repression constant being varied between .1 to 1.

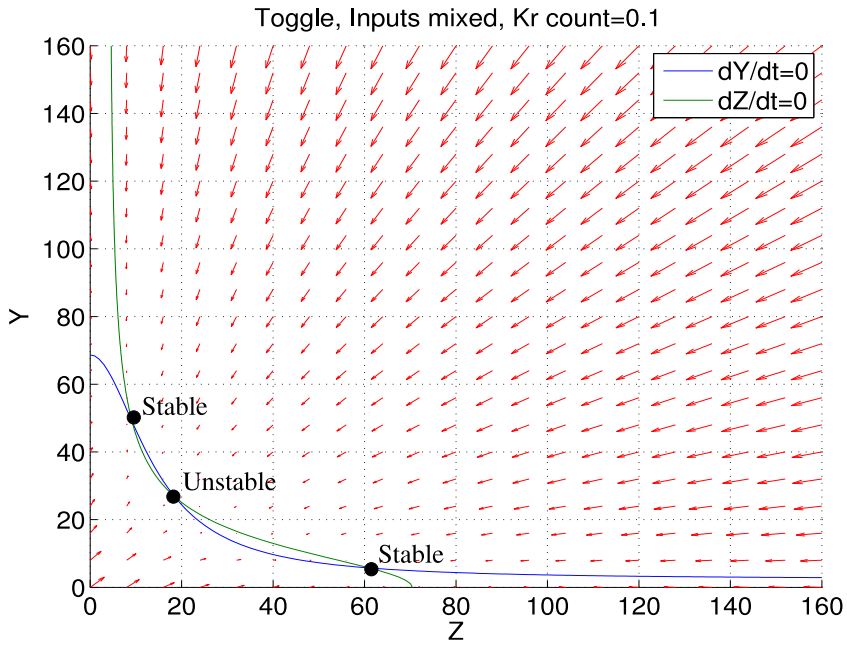
example, this technique is used in tuning the genetic oscillator [13]. Increasing the decay rate might make the circuit switch faster since proteins leave the system faster.

The decay rate is varied from 1.875e-3 to 0.03. The results for the rates of failure are shown in Fig. 3.41 and 3.42. The results show that slightly increasing the decay rate may cause a slight decrease in failure rate in the low to high experiments. The reason can be seen by plotting the nullclines. Fig. 3.43 shows the nullclines of the toggle gate for k_d of 0.005 and 0.0075. Increasing the decay rates decreases the separation distance between the equilibrium points. This makes holding state more difficult as fluctuations in productions can more easily flip state. The switching time for the gates are shown in Fig. 3.44. Increasing decay rate decreases switching time since species degrade faster. A decay rate of 0.00375 yields the most gain in robustness.

These results are similar to a digital Muller C-element using weak feedback to hold state as shown in Fig. 3.45. If the feedback is too weak, then the Muller C-element loses state easily. If the feedback is too strong, then the switching time is higher. If the degradation rate in the genetic Muller C-element is too low, then the state holding species decay more rapidly, and the genetic Muller C-element loses state easily. If the degradation rate is too high, then it is much harder to switch states and the switching



(a)



(b)

Figure 3.39. The nullclines of the (a) SI gate and (b) toggle gate when both inputs are mixed, and K_R is 0.1.

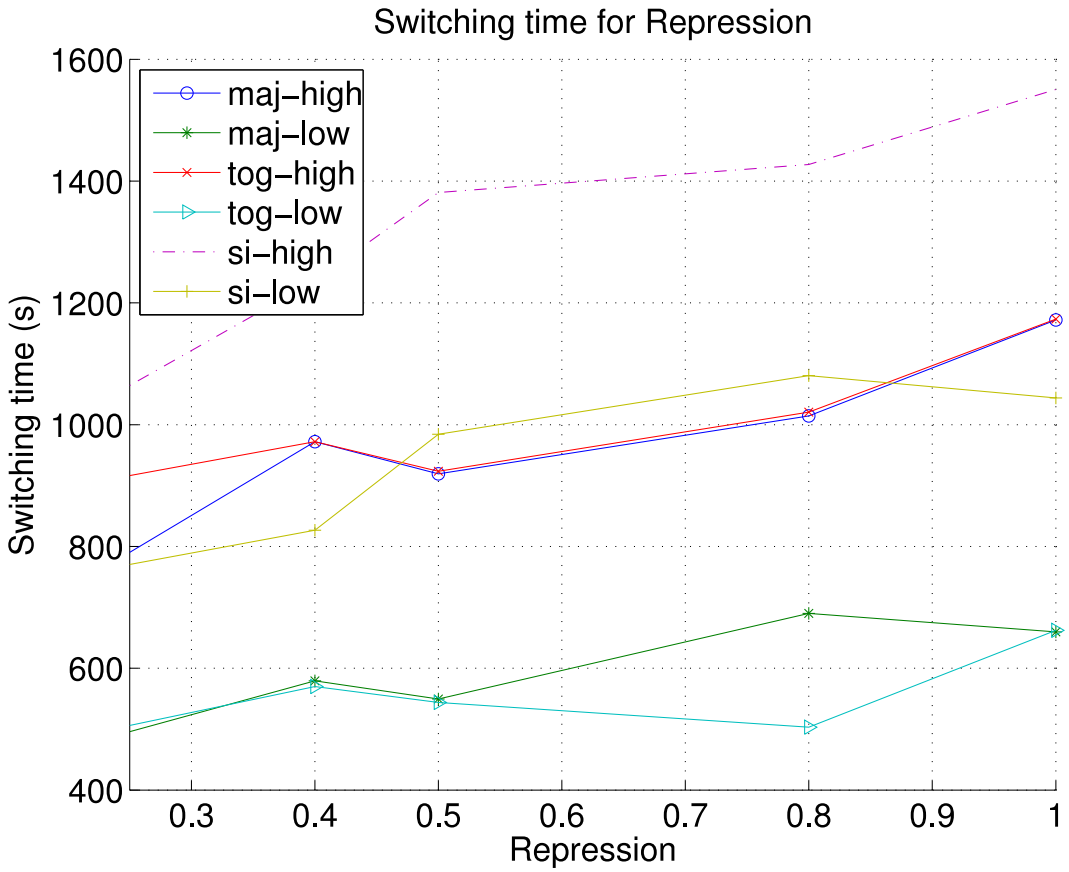


Figure 3.40. Switching time of the C-element, varying the repression strength.

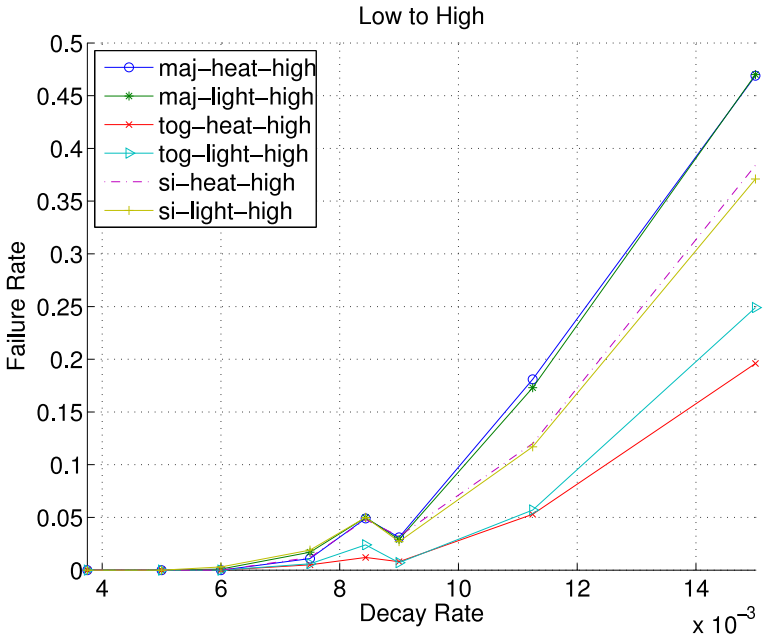


Figure 3.41. The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the decay rate being varied between 1.875e-3 to 0.03.

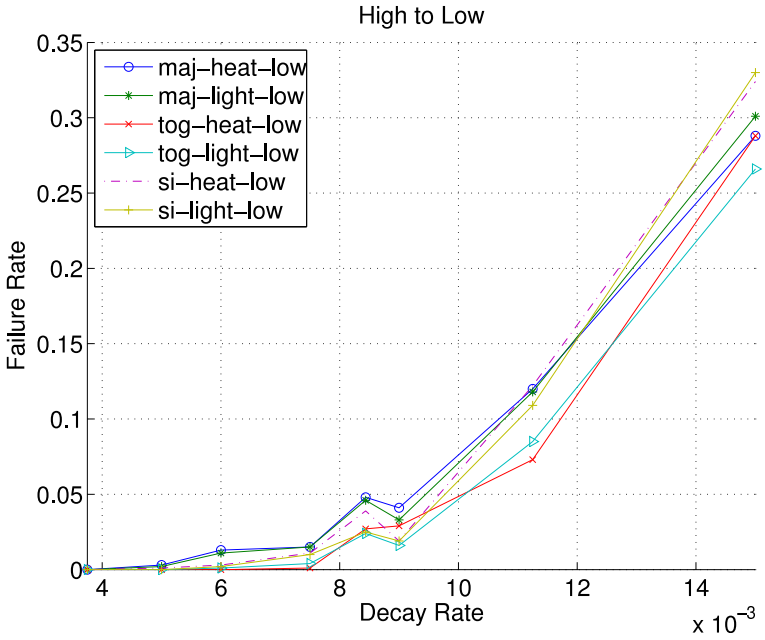
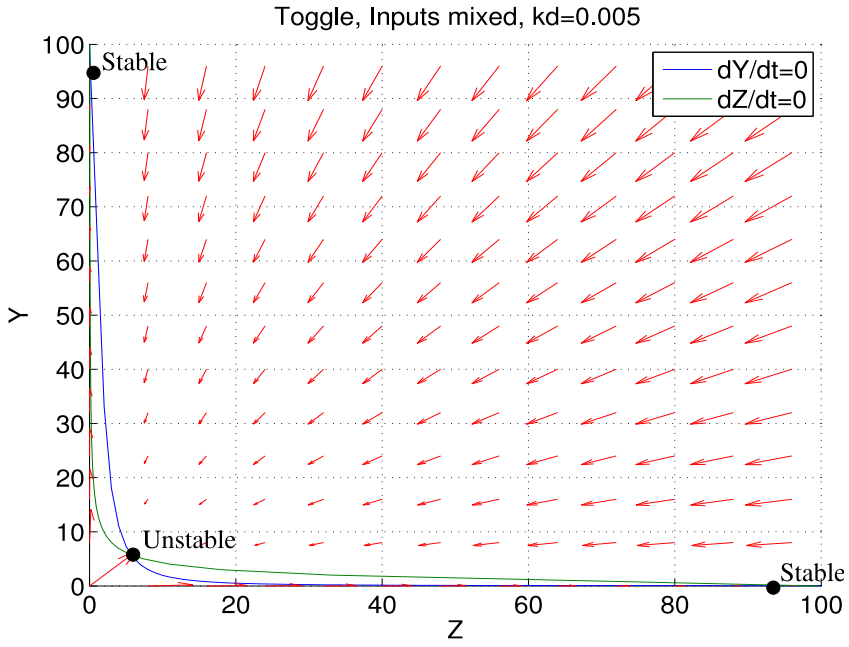
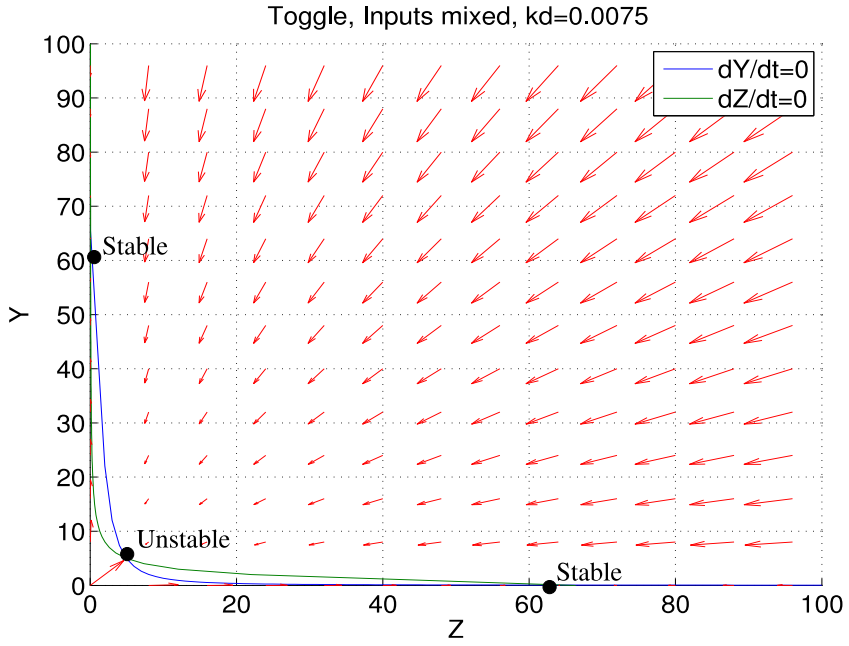


Figure 3.42. The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, with the decay rate being varied between 1.875e-3 to 0.03.



(a)



(b)

Figure 3.43. The nullclines of the toggle gate when both inputs are mixed, and k_d is (a) 0.005, (b) 0.0075. The results show that as k_d decreases, the separation distance between the stable point increases.

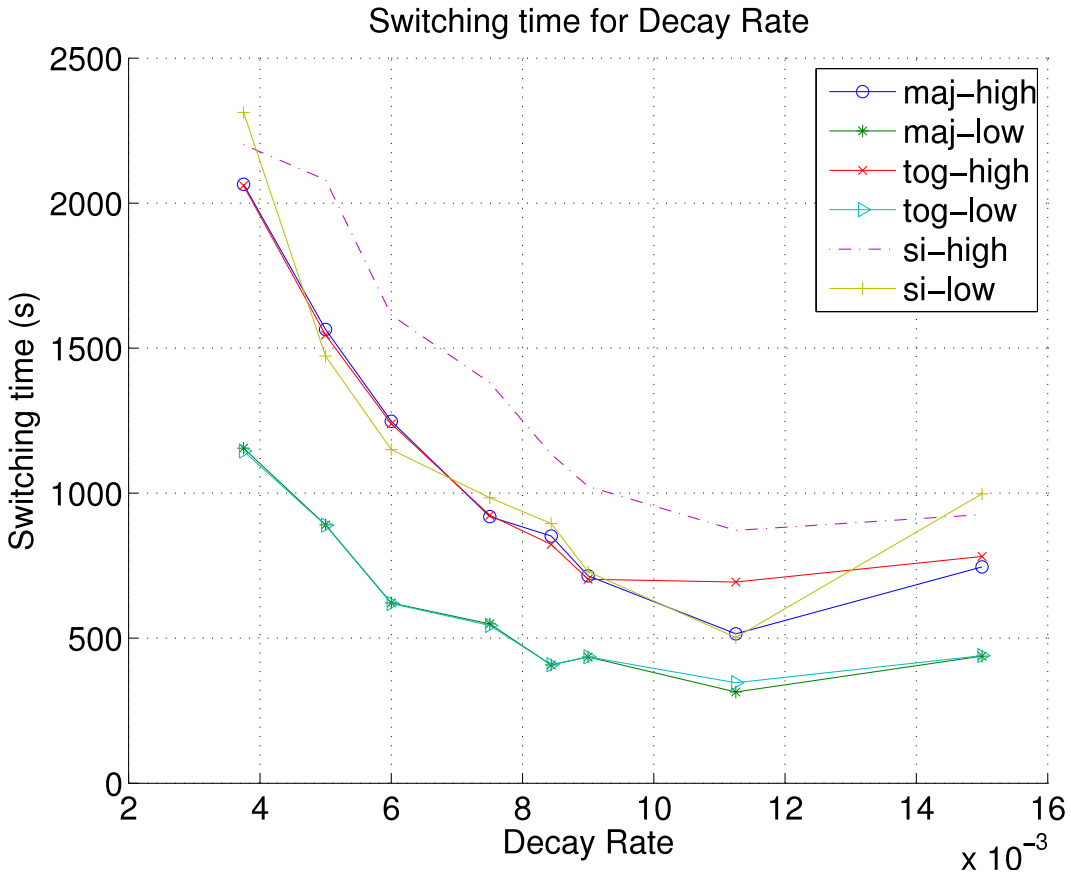


Figure 3.44. Switching time of the C-element, varying the decay rate.

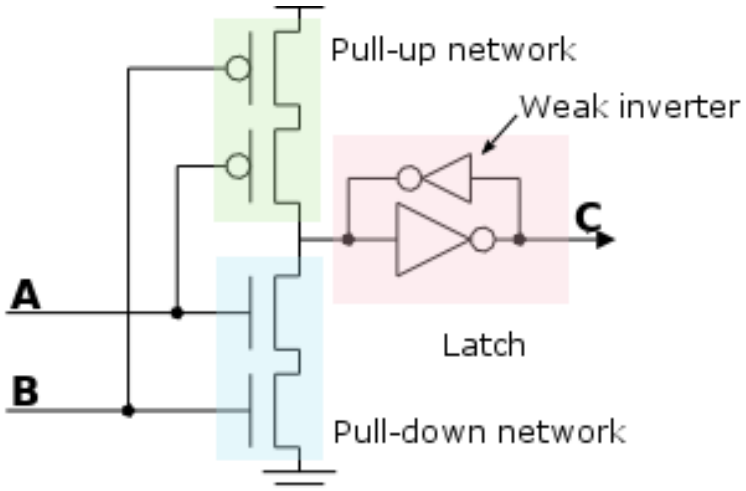


Figure 3.45. A schematic for a digital Muller C-element with weak feedback.

time increases. There must be a balance between degradation rates in order to have a genetic Muller C-element that switches rapidly and is robust to failure.

3.4.5 Changing Production and Degradation Rates

Next, the production and decay rates are changed in lockstep by a constant multiplier. This has an effect of keeping the equilibrium point of the system the same. Increasing the production and degradation rates by the same multiplier also decreases the switching time. The gate can reach the high level faster due to increased production and can reach the low level faster due to the faster degradation.

The production and degradation rates are multiplied by values 0.5 to 4. The results for the rates of failure are shown in Fig. 3.46 and 3.47. The results show that increasing the ratio of production increases the failure rate. This may be due to the increase influence of stochastic effects. As production rates increase, spontaneous production is more likely to occur. At the same time, the increase in decay rate causes more spontaneous degradation reactions. It becomes more difficult to maintain equilibrium when both production and degradation can push the gate into the incorrect state. Fig. 3.49 shows the nullclines of the SI gate for the constant multiplier of 0.5 and 2.0. As expected, the equilibrium point remains the same as the production and decay rate increases. The switching time for the gates are shown in Fig. 3.48. Also as expected, increasing the constant multiplier decreases

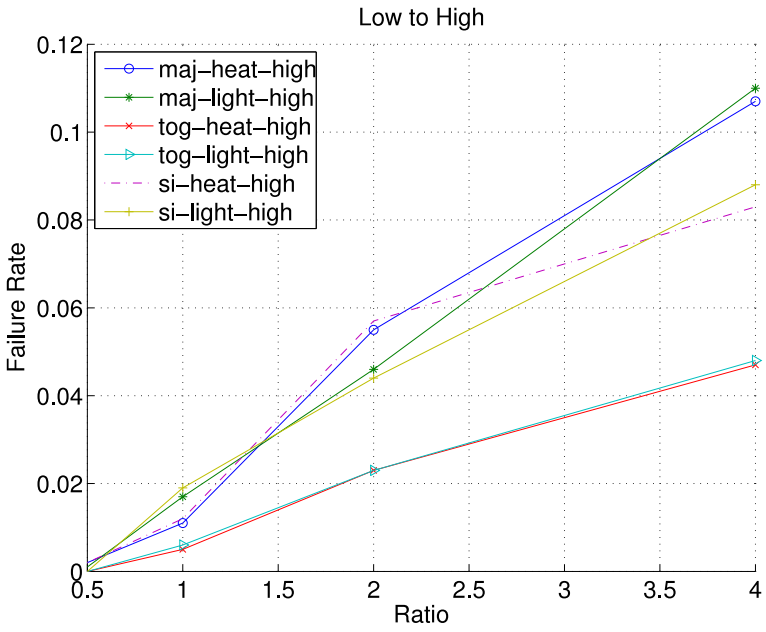


Figure 3.46. The failure rate of the C-element at 2100 seconds after one of the inputs has been added, with the ratio of production to decay being varied between 0.5 to 4.0.

the switching time. A constant multiplier of 0.5 yields the most gain in robustness. However, there is a larger penalty in switching time for a constant multiplier of 0.5.

3.5 Design Principles

The utility of the GCM language proved to be invaluable in the analysis of the various genetic Muller C-element designs. The GCM improved design time by providing a compact graphical representation for each genetic Muller C-element. The SBML generation reduces errors by automatically generating the SBML. Simulation time is improved by leveraging the ease of abstraction of the SBML format. The parameter variation simulations are easy to generate due to the global parameter fields of the GCM file. The environment of the parameter variation simulations are easy to create since the GCM can be merged with SBML. These advantages of using the GCM language over directly using SBML greatly aided in the design and analysis of the genetic Muller C-element.

From the mathematical analysis and stochastic simulation analysis, there are several design principles that we can deduce for synthetic biology. First, of the three designs of a genetic Muller C-element, the toggle gate design is the most robust design. The toggle gate performed better than the majority gate and SI gate for nearly all the

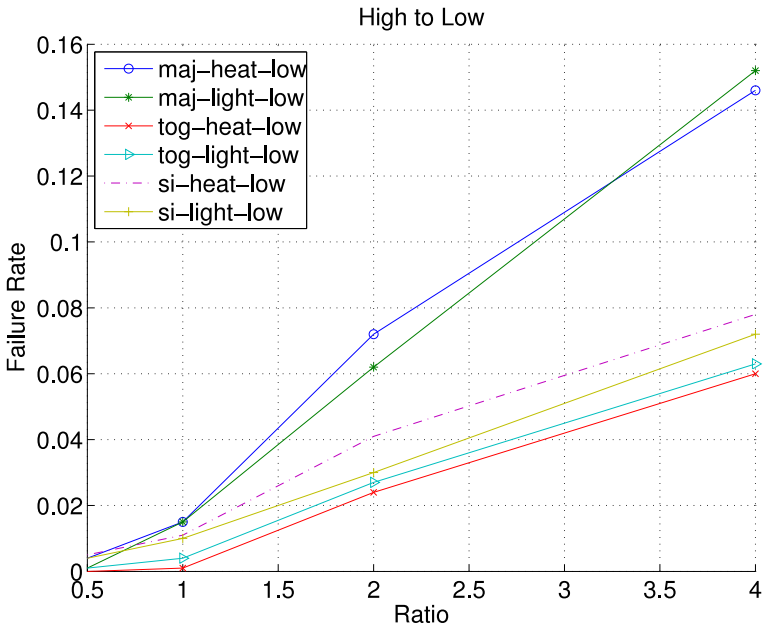


Figure 3.47. The failure rate of the C-element at 2100 seconds after one of the inputs has been removed, the ratio of production to decay being varied between 0.5 to 4.0.

cases. The toggle gate shows that the number of timing assumptions needed to function correctly are not necessarily correlated to robustness. The SI gate has the fewest timing assumptions, and ironically, it performed worst than the toggle gate. The results also show that modifications to traditional synthesis methods are necessary since all three designs are logically equivalent, but their robustness characteristics are different. The results also show that robustness can be increased by adding more genes, at the cost of increased switching time. Second, cooperativity is vital in designing functioning gates. Cooperativity is necessary for switch-like behavior in gates, and without cooperativity, gates cannot maintain state. Third, increasing repression also causes gates to have more switch-like behavior, requiring fewer molecules of repressor to shut down production. The trade off is switching time increases as repression strength increases. More repressor molecules must degrade before the same level of production is reached. Fourth, high decay results in improved switching time at the cost of increased failure rate. Therefore, a balance between production rates and degradation rates are necessary because if the ratio is too low, then switching times are greatly increased, but if the ratio is too high, then failure rates are greatly increased.

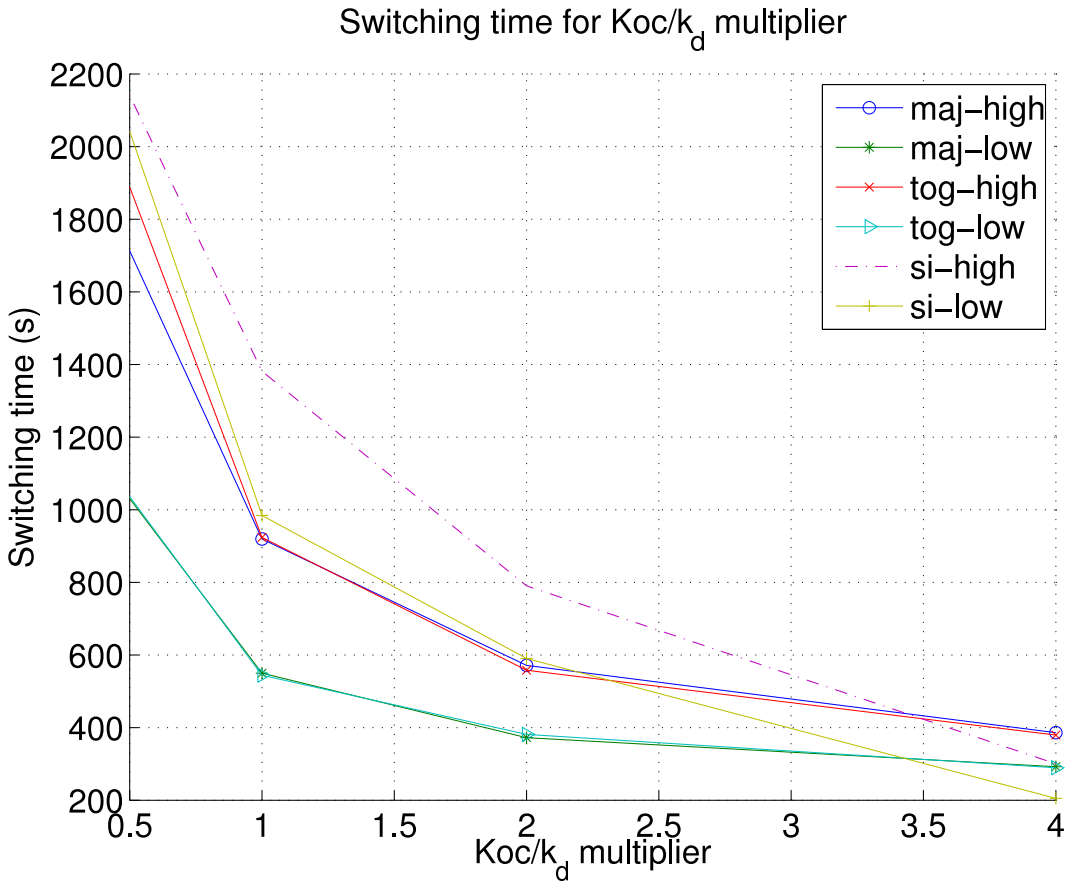
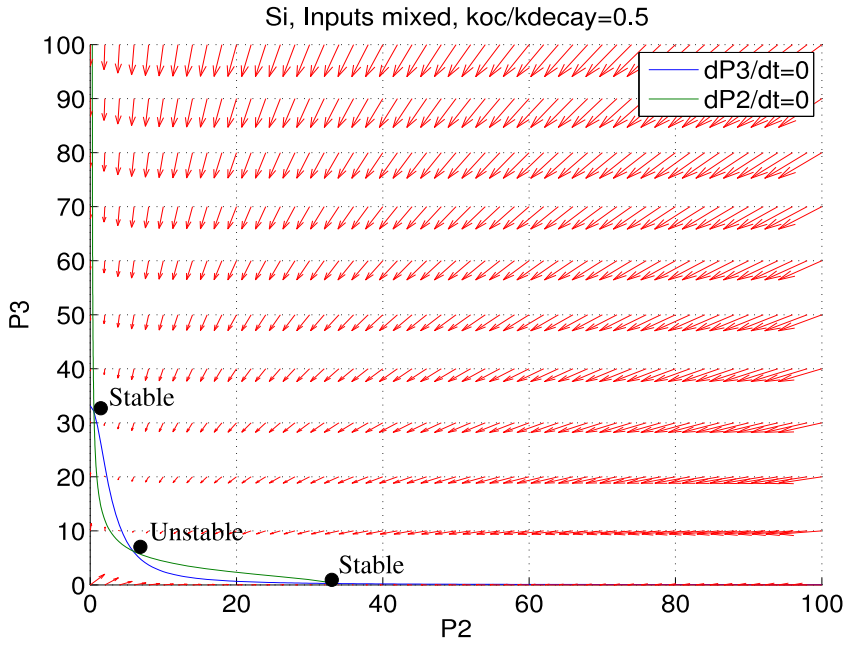
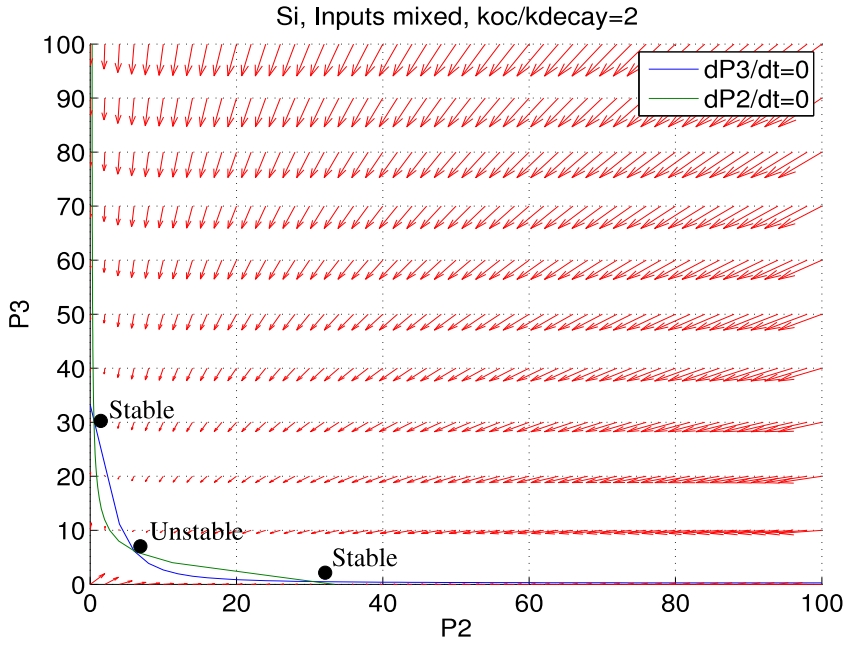


Figure 3.48. Switching time of the C-element, varying the production to decay rate.



(a)



(b)

Figure 3.49. The nullclines of the SI gate when both inputs are mixed, and the constant multiplier is (a) 0.5, (b) 2.0. The results show that as the constant multiplier increases, the separation distance remains the same.

3.6 Application

Anderson et al. are designing bacteria that can detect and kill tumors [4]. The bacteria colony starts the invasion of the cancer cells when a signal is detected. Three invasion signals have been tested: an inducible signal using arabinose, an environmental signal using hypoxic conditions, and a density dependent signal using lux quorum sensing circuit. Three strains of bacteria have been created to detect the three different signals. They have not, however, tried to create a strain of bacteria that detects two or more signals. One potential application of the genetic Muller C-element is to use it to synchronize two signals to determine whether or not to start the invasion. Combining two signals increases the confidence in the decision to start the invasion.

We created a simulation in which each bacteria uses a Muller C-element to determine when to start invasion of tumor cells. The bacteria activates when it detects the environmental signal and the communication signal. Activated bacteria sends the communication signal to its neighbors that the bacteria has activated. This communication signal is used by the bacteria to reach consensus. The communication signal accumulates in the environment and is removed through diffusion. The detection of the environmental signal has a rate of error because detection of the environmental signal is noisy. The bacteria behave unreliably, and have a rate of switching state randomly.

The population size of the bacteria colony is 200. Each bacteria is initially set to a random state. The simulation is run for 2500 time steps. Initially, the environmental signal is not present. The detection rate of the environmental signal has an associated error rate of 0.40, so each bacteria has a 40 percent chance of incorrectly detecting the environmental signal. After 500 time steps, the environmental signal is applied, and each bacteria has a 60 percent chance of correctly detecting the environmental signal.

Fig. 3.50 shows the result of the simulation when each bacteria has a rate of randomly switching state of 0.005. This rate is roughly equivalent to the failure rate of the genetic toggle Muller C-element. The result shows that the majority of the bacteria colony correctly remain unactivated when the environmental signal is not present, and the majority of the bacteria colony correctly activates when the environmental signal is present. Fig. 3.51 shows that the bacteria colony still maintains the correct behavior if the error rate of switching is increased by an order of magnitude to 0.05. Interestingly enough, if the error rate of switching is set to 0, then the bacteria colony fails to behave correctly. Fig. 3.52 shows that the bacteria colony does not activate, even after the environmental

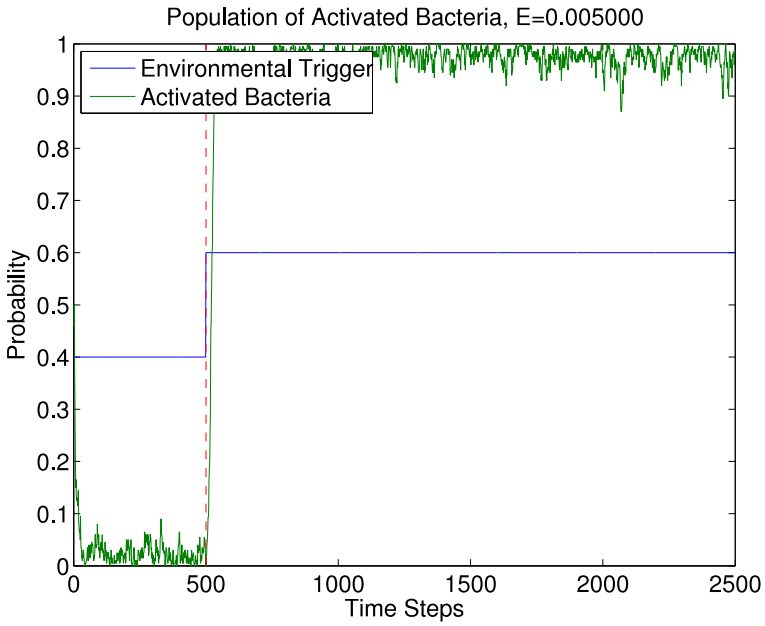


Figure 3.50. Bacteria consensus with an error rate of switching of 0.005.

signal is present. This result indicates that not only can the genetic Muller C-element handle error, the error is, in fact, necessary for the correct behavior.

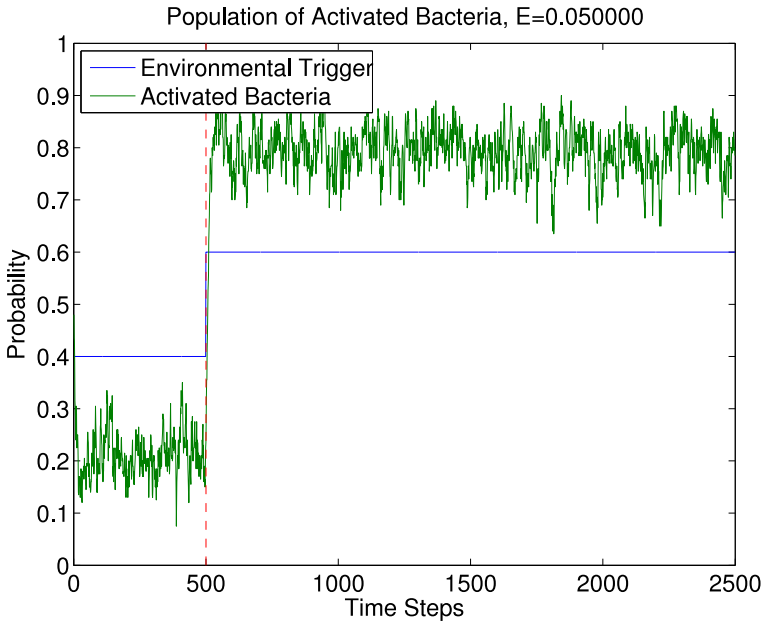


Figure 3.51. Bacteria consensus with an error rate of switching of 0.05.

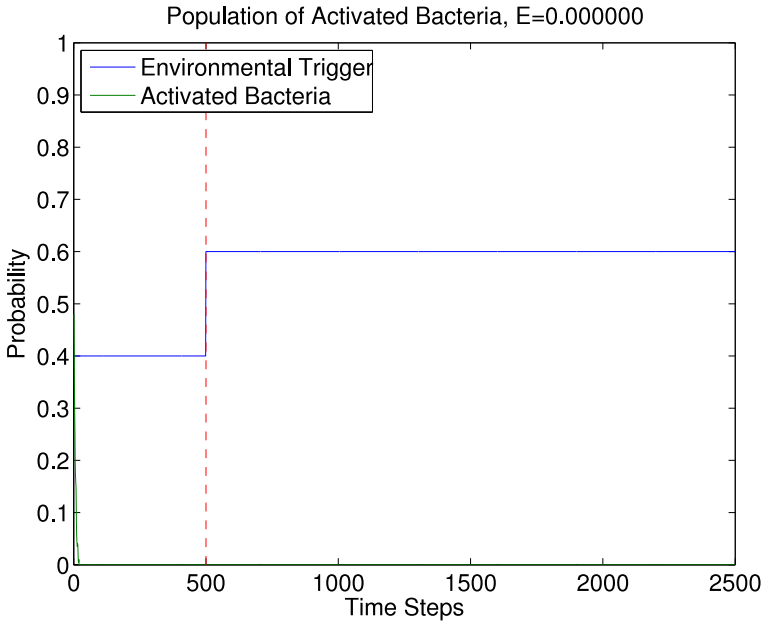


Figure 3.52. Bacteria consensus with no error rate of switching.

CHAPTER 4

CONCLUSION

Synthetic biology is an emerging new field that holds many promising applications. However, current techniques of developing genetic circuits at the reaction level is too time consuming and prone to error. Existing support for SBML require designers to manually design the layout of their genetic circuits. Directly building complex SBML models requires the user to write out each species and reaction. Similar advances found in EDA tools such as automated design, design verification, and logic simulation may speed the advances in synthetic biology. This thesis presents the GCM format and a synthesis method from GCM to SBML models in a first step in developing a powerful GDA tool. Finally, this thesis uses the GCM language to design and analyze three different genetic Muller C-element designs.

4.1 Summary of Work

There are two distinct sections presented in this work. The first section is the presentation of the GCM language. This section also presents the translation of the GCM into a simulatable model. The second section is the case study of the design of a genetic Muller C-element using the GCM language.

The GCM language is a graphical specification language which provides a more compact representation of a genetic circuit. The GCM only includes species and the influences of the species on each other. This abstraction allows designers to more efficiently develop models of genetic circuits. Once a GCM is completed, the translation algorithm can transform the GCM into an SBML model. This process reduces errors introduced from manual model creation. The translation algorithm also leverages design patterns so that the resulting SBML can be automatically abstracted, increasing simulation efficiency.

The GCM language contains several features to aid in experiment design. The GCM allows editing of both local and global parameters. Global parameters allows designers to easily set up simulations to sweep key parameters. Local parameters allow designers

to fine tune species and reactions to have specific behaviors. Finally, the GCM allows for the merging of an existing SBML file. Designers can use this capability to add initial assignments, rules, constraints, and events found in SBML. Using these features, the designer can have multiple GCM's use the same test environment during simulation.

These features of the GCM are leveraged in the design of the genetic Muller C-element. A Muller C-element is a state holding gate that is crucial in the design of asynchronous circuits. A genetic Muller C-element can theoretically allow for the design of any asynchronous stating holding gate. This thesis presents and analyzes the design of three different Muller C-elements: the majority gate design, toggle gate design, and SI gate design. Each design has less timing assumptions than the previous. The SBML merging feature of the GCM allowed us to rapidly design and simulate each gate under the same environment. We are also able to easily set up robustness to parameter variation simulations by using the global parameters in the GCM. The results show that timing assumptions may not have as much of an effect on robustness as other factors, as the toggle gate design is the most robust design, but contains more timing assumptions than the SI gate and less timing assumptions than the majority gate. Design priciples gathered from the simulations are that dual-rail outputs are essential, high gene count increases robustness, cooperativity greater than one is necessary, repression needs to be strong, and decay rates must be balanced for high robustness and low switching time. The results also suggest that adaptation is necessary for the application of logic design and synthesis to genetic circuits.

Finally, one potential application of the genetic Muller C-element is determining when to start the invasion of cancer cells. The two input signals are an environmental signal, and a communication signal. Using these signals, the bacteria colony can correctly reach consensus on when to begin the invasion. One interesting result is that noise is necessary in correctly switching into the invasion state.

4.2 Future Work

We have presented the utility of the GCM using the design and analysis of the genetic Muller C-element as a case study. There are still many extensions to the GCM that can improve the usefulness of the language. This section summarizes several such improvements.

4.2.1 Hierarchy

The GCM is the first step towards the design of a high level *hardware description language* (HDL) for genetic circuits. HDLs are formal descriptions that provide a higher level of description of the behavior. Examples for electronic circuits include Verilog and VHDL. One key feature found in HDLs is the inclusion of hierarchy. This allows smaller circuits to be pieced together to form larger and more complex circuits. Currently, SBML does not have support for hierarchy. For example, it is not possible to take an SBML model representing a genetic inverter and make two more copies of the model and stitch the copies together to make a three ring inverter that oscillates. In order to stitch together the SBML copies, the designer must change each species in each inverter to be unique, and then make sure the that output species of the previous inverter make the input species of the next inverter. This is tedious work, and the designer must make these changes for every SBML model that is to be stitched together.

The second step is to add levels of hierarchy into the GCM formation. In particular, we plan to create structural constructs that allow us to connect GCM's for separate modules through species ports. The species ports would provide specific connections between the separate GCMs. For example, we can take a genetic inverter template, and connect three of these inverters together by specifying that the output of the previous inverter connects to the input of the next inverter. During the SBML generation process, the translation algorithm would then create unique species names for each inner species in each inverter, and also ensure that any species used to connect the species ports have the same name. This would mimic hierarchy that is currently missing in the SBML format.

4.2.2 Technology Mapping

Once hierarchy has been implemented, the next step is develop a gate library such that *technology mapping* can be implemented. Technology mapping is the process of generating a circuit from logic equations using only gates found in a gate library for a specified technology. The Muller C-element will be a key gate in our gate library, allowing us to use methods in synthesizing asynchronous circuits and apply them in synthesizing genetic circuits. Users can input logic equations, and the technology mapping algorithm would automatically generate a genetic circuit implementing the logic equations. Asynchronous circuits would be a good source of knowledge to draw from since there's no global clock in a cell, timing is difficult to characterize, and cells tend to have state holding elements. These

are the same issues that asynchronous circuit designers must also deal with. Technology mapping is NP-Hard, however, as long as circuits as well as the library are small, we can use brute force methods to generate a circuit. If we find that the problem quickly becomes intractable, we will draw on ideas from DAGON [24] to try to come up with accurate approximations.

4.2.3 Synthesizing the Genetic Muller C-element

While this thesis represents a promising first step, there are several technical issues that must be further explored. First, due to the stochastic nature of the behavior of the gates, it is necessary to develop methods to build reliable circuits out of unreliable parts. One possible way to handle failure is to have multiple copies of the circuit in a cell, as shown in the parameter variation results. Second, it is necessary to examine which parameters are vital to the correctness of the genetic Muller C-element and how much variation is possible before it fails to work correctly. Furthermore, the design methods must take into account the limitations that biologists have in tuning parameters. While controlling decay rates may not be too difficult, production rates may be more difficult to tune. By running many different parameter variation experiments, it may be possible to come up with a reasonable set of parameters for a robust circuit that is physically possible to build. Many issues cannot be fully explored until the DNA sequences for these genetic circuits are tested *in vivo* (i.e., within a living organism). One possible schematic is shown in Figure 4.1. Here, the inputs are heat and light, and the output is *Green Fluorescent Protein* (GFP), a protein that fluoresces under the presence of UV light. This allows us to easily control the inputs to the circuit, and the output can be measured by the intensity of GFP. There are currently several companies which produce DNA given a sequence. This DNA can then be inserted into an organism such as *E. coli* for testing. Building a functional genetic Muller C-element would validate our models and abstractions.

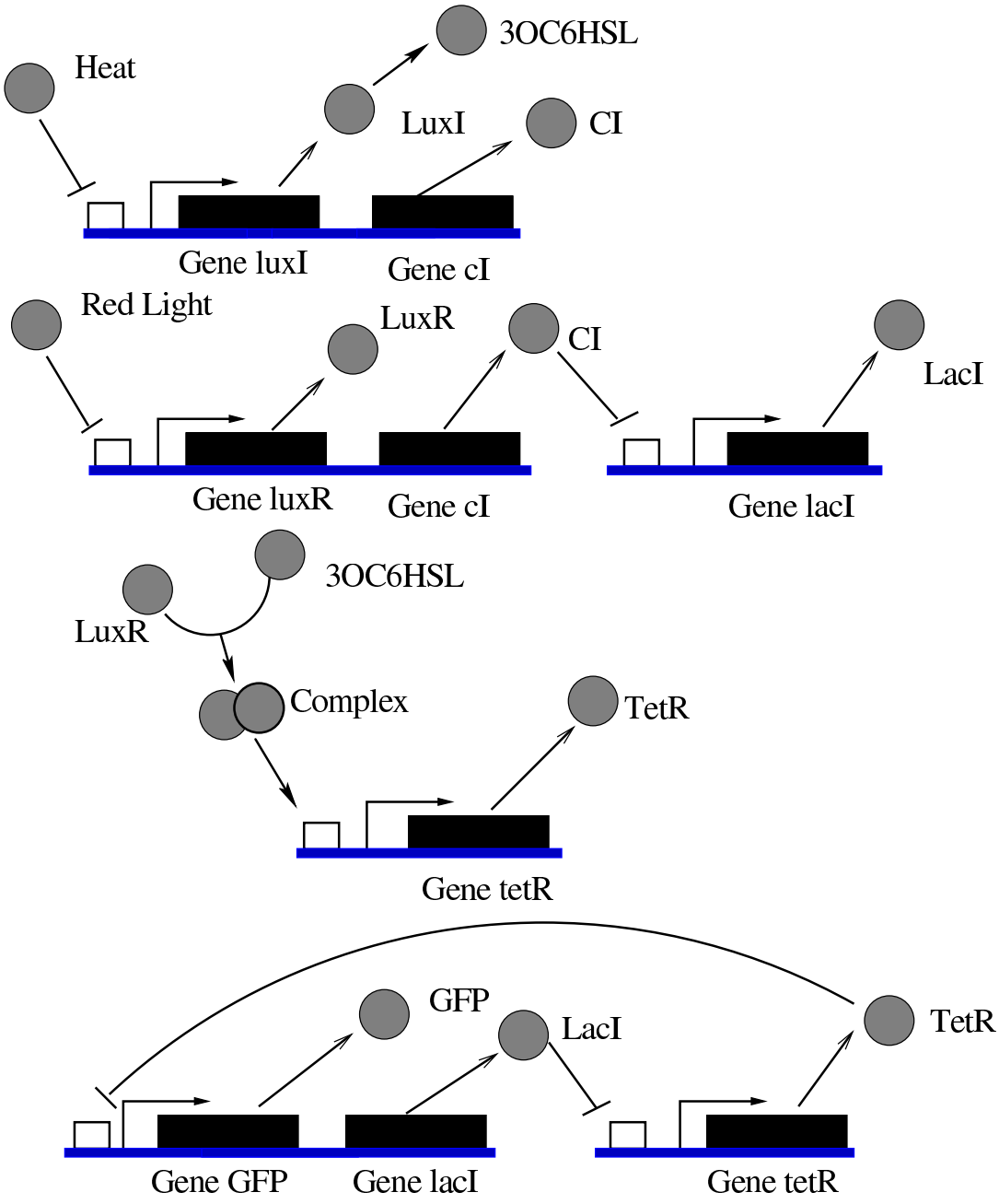


Figure 4.1. A possible realization of a genetic toggle Muller C-element.

APPENDIX

GCM SPECIFICATION LANGUAGE

The GCM format specifies a genetic circuit using the same format as the GraphViz graph drawing tool [31]. The vertices in the graph are the species in the genetic circuit, and the edges in the graph represent the activation and repression relationships between the species. An activation relationship is shown with a blue (blue4) arrow (vee) and a repression relationship is shown with a red (firebrick4) tee. The label field in the species declaration is the name of the species. The arrowhead field in the relationship declaration represents the type of relationship between the species. Repression is labeled with a tee and activation is labeled with a vee. The label field in the relationship declaration represents how many molecules are necessary to activate or repress the production of the species. The GCM in Fig. A.1 shows a simple genetic circuit in which the species CI represses CII while CII activates CI production. The s1->-s2 edge has a label field of "2" which means two molecules of CI are required to form a dimer to repress CII.

More advanced behavior can be modeled by using extra fields. The GCM in Fig. A.2 shows how the promoter field works. In the genetic circuit model below, species A

```
digraph G {
  s1 [shape=ellipse,color=black,label="CI"]
  s2 [shape=ellipse,color=black,label="CII"]
  s2 -> s1 [color="blue4",arrowhead=vee]
  s1 ->- s2 [color="firebrick4",label="2",arrowhead=tee]
}
Global {
}
Promoters {
}
```

Figure A.1. Part of the phage λ circuit

```
digraph G {

  s1 [shape=ellipse,color=black,label="A"]
  s2 [shape=ellipse,color=black,label="B"]
  s3 [shape=ellipse,color=black,label="C"]
  s1 ->- s2 [color="blue4",arrowhead=tee,promoter="P1"]
  s1 ->- s3 [color="blue4",arrowhead=tee,promoter="P2"]
}
Global {
}
Promoters {
P1 [Name=P1]
P2 [Name=P2]
}
```

Figure A.2. Simple example with different promoters.

represses the production of species B and C, independently. If there is exactly 1 molecule of species A, it would only be able to repress production of species B or C, but not both.

With the promoter field, as shown in The GCM in Fig. A.3, one species, A, now represses the promoter "P1", which produces both species B and C. This means that species B and C are being produced from the same promoter. This allows one molecule of species A represses the production of both species B and C.

```
digraph G {

  s1 [shape=ellipse,color=black,label="A"]
  s2 [shape=ellipse,color=black,label="B"]
  s3 [shape=ellipse,color=black,label="C"]
  s1 ->- s2 [color="blue4",arrowhead=tee,promoter="P1"]
  s1 ->- s3 [color="blue4",arrowhead=tee,promoter="P1"]
}
Global {
}
Promoters {
  P1 [Name=P1]
}
```

Figure A.3. Simple example with same promoters.

The promoter field can also be used to separate production reactions. In the GCM in Fig. A.4, both species A and B can repress the production of species C. If either species is present, then very little species C is produced. This behavior is the same as a NOR gate.

However, if there needs to be two different sources of production for species C, the promoter field can be used to accomplish this. In the GCM in Fig. A.5, A represses the production of C by binding to the P1 promoter, and B represses the production of C by binding to the P2 promoter. Both A and B need to be present to fully repress the level of C. If either is at a low level, then the level of C will be high. This behavior is the same as a NAND gate.

The GCM in Fig. A.6 shows how to model an AND gate. The reactions contain a promoter label "P1". This means that the species C can be activated by both s1 and s2. Combined with the type flag of biochemical, this creates a biochemical reaction where species A and B combine together to form a complex to activate production of species C. While not necessary, the species A and B have the constant flag set to true. This means that A and B have no production and degradation reactions. This additional parameter allows the user to model contanst inputs to the gate.

Global parameters can be created and manipulated through the global parameter field. GCM A.7 shows an example that contains global parameters. A mapping between

```
digraph G {

  s1 [shape=ellipse,color=black,label="A"]
  s2 [shape=ellipse,color=black,label="B"]
  s3 [shape=ellipse,color=black,label="C"]
  s1 ->- s3 [color="blue4",arrowhead=tee,promoter="P1"]
  s2 ->- s3 [color="blue4",arrowhead=tee,promoter="P1"]
}
Global {
}
Promoters {
  P1 [Name=P1]
}
```

Figure A.4. Simple example that behaves as a NOR.

```
digraph G {

  s1 [shape=ellipse,color=black,label="A"]
  s2 [shape=ellipse,color=black,label="B"]
  s3 [shape=ellipse,color=black,label="C"]
  s1 ->- s3 [color="blue4",arrowhead=tee,promoter="P1"]
  s2 ->- s3 [color="blue4",arrowhead=tee,promoter="P2"]
}
Global {
}
Promoters {
  P1 [Name=P1]
  P2 [Name=P2]
}
```

Figure A.5. Simple example that behaves as a NAND.

```
digraph G {

  s1 [shape=ellipse,color=black,label="A",const=true]
  s2 [shape=ellipse,color=black,label="B",const=true]
  s3 [shape=ellipse,color=black,label="C"]
  s1 ->- s3 [color="blue4",arrowhead=vee,promoter="P1",type=biochemical]
  s2 ->- s3 [color="blue4",arrowhead=vee,promoter="P1",type=biochemical]
}
Global {
}
Promoters {
  P1 [Name=P1]
}
```

Figure A.6. Simple example that behaves as an AND.

```
digraph G {

  s1 [shape=ellipse,color=black,label="A",const=true]
  s2 [shape=ellipse,color=black,label="B",const=true]
  s3 [shape=ellipse,color=black,label="C"]
  s1 ->- s3 [color="blue4",arrowhead=vee,promoter="P1"]
  s2 ->- s3 [color="blue4",arrowhead=vee,promoter="P1"]
}
Global {

  Promoter count=2
  Krep=.05
}
Promoters {
  P1 [Name=P1]
}
```

Figure A.7. Simple example that contains global parameters.

Table A.1. Mapping of Parameter List.

Parameter	GCM Name	Symbol	Structure
Initial RNAP count	RNAP count	n_r	model
Initial species count	Initial amount	n_s	species
Dimerization equilibrium	Kassociation	K_d	species
Degradation rate	kdecay	k_d	species
Biochemical equilibrium	Kbio	K_b	influence
Degree of cooperativity	Binding site count for transcription factors	n_c	influence
N-mer as transcription factor	N-mer as transcription factor	nd	influence
Repression binding equilibrium	Krep	K_r	influence
Activation binding equilibrium	Kact	K_a	influence
Initial promoter count	Initial amount	n_g	promoter
RNAP binding equilibrium	KRNAP	K_o	promoter
Basal production rate	kbasal	k_b	promoter
Open complex production rate	kocr	k_o	promoter
Activated open complex production rate	Activated kocr	k_a	promoter
Stoichiometry of production	Stoichiometry of production	np	promoter

the global parameter names and the symbol of the global parameter in the SBML can be found on Table A.1.

REFERENCES

[1] BioSPICE. http://biospice.sourceforge.net/.

[2] Registry of standard biological parts. http://parts.mit.edu/.

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

[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] Arkin, A., and Ross, J. Computational functions in biochemical reaction networks. *Biophysical Journal 67* (August 1994), 560–578.

[6] Barker, N. *Learning Genetic Regulatory Network Connectivity from Time Series Data*. Phd in computer science, University of Utah, 2007.

[7] Biopathwise. http://bioanalyticsgroup.com/.

[8] Brazil, G. M., Kenefick, L., Callanan, M., Haro, A., de Lorenzo, V., Dowling, D. N., and O'Gara, F. Construction of a rhizosphere pseudomonad with potential to degrade polychlorinated biphenyls and detection of *bph* gene expression in the rhizosphere. *Applied and Environmental Microbiology 61*, 5 (1995), 1946–1952.

[9] 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.

[10] Celldesigner. http://celldesigner.org/.

[11] Cherry, J. L., and Adler, F. R. How to make a biological switch. *J. theor. Biol. 203* (2000), 117–133.

[12] Davidson, E. A., and Ellington, A. D. Engineering regulatory rnas. *Science 23*, 3 (2005).

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

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

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

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

[17] Gillespie, D. T. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. of Comp. Phys. 22* (1976), 403–434.

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

[19] Goler, J. A. Biojade: A design and simulation tool for synthetic biological systems. Tech. rep., Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2004. AI Technical Report 2004-003.

[20] Guet, C. C., Elowitz, M. B., Hsing, W., and Leibler, S. Combinatorial synthesis of genetic networks. *Science 296* (2002), 1466–1470.

[21] ibiosim. `http://www.async.ece.utah.edu/tools/index.html`.

[22] Jacob, F., and Monod, J. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology 3* (Jun 1961), 318–356.

[23] Kærn, M., Blake, W. J., and Collins, J. The engineering of gene regulatory networks. *Annu. Rev. Biomed Eng. 5* (2003), 179–206.

[24] Keutzer, K. Dagon: Technology binding and local optimization by dag matching. In *Annual ACM IEEE Design Automation Conference* (1987), ACM Press, pp. 341–347.

[25] Kuwahara, H. *Model Abstraction and Temporal Behavior Analysis of Genetic Regulatory Networks*. Phd in computer science, University of Utah, 2007.

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

[27] Mayevsky, O., Varshavsky, V., Marahovsky, V., and Mamrukov, Y. Ussr patent certificate n1081801, the inventions bulletin n 13, 1983.

[28] Nguyen, N., Kuwahara, H., Myers, C., and Keener, J. Design of a genetic muller c-element. In *13th IEEE International Symposium on Asynchronous Circuits and Systems* (2007), IEEE Computer Society, pp. 95–104.

[29] Ptashne, M. *A Genetic Switch*. Cell Press & Blackwell Scientific Publishing, 1992.

[30] 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).

[31] Graphviz. `http://www.graphviz.org/`.

[32] Simbiology. `http://www.mathworks.com/products/simbiology/`.

[33] Synthetic biology. `http://syntheticbiology.org/`.

[34] Sprinzak, D., and Elowitz, M. B. Reconstruction of genetic circuits. *Nature 438* (2005), 443–448.

[35] Thijn R. Brummelkamp, Renè Bernards, R. A. A system for stable expression of short interfering rnas in mammalian cells. *Science 296* (2002), 550–553.