Programming Molecules and Cells: Design Architectures for Chemical Reaction and Gene Regulatory Networks

Kevin Takashi Oishi

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Eric Klavins, Chair

Georg Seelig

James M. Carothers

Program Authorized to Offer Degree:

Electrical Engineering

University of Washington

Abstract

Programming Molecules and Cells: Design Architectures for Chemical Reaction and Gene Regulatory Networks

Kevin Takashi Oishi

Chair of the Supervisory Committee:

Associate Professor Eric Klavins

Electrical Engineering

The stages of cell differentiation are often illustrated as a sequence of events and chemical cues that move a cell from one state to another. Differentiated cells send and receive signals to compute functions on their environments and perform complex tasks such as pattern formation. But how would one program a cell, *de novo*, to have these behaviors? Great strides have been made in developing tools for genetically modifying organisms to carry out simple tasks, and the wealth of literature in quantitative biology and genome engineering speaks to these efforts. However, what may be missing is an engineering framework—a formal layering of mathematical abstractions connected to physical implementations via a "biomolecular compiler". Engineering frameworks and compilers are instrumental to the design and implementation of other technological systems—it is the reason that complex commercial airplanes can be proven safe, and that computers are useful tools and not one-of-a-kind nests of unreliable circuitry. It seems clear that similar technomimetic frameworks for synthetic biology should exist, however may questions remain. What abstractions or specification languages are suitable for engineering living organisms? Are the abstractions for specifying single celled behaviors suitable for specifying multicellular behaviors? How are such specifications physically instantiated? The original contribution of this thesis is to

develop two frameworks for engineering dynamical and computational systems: a compiler taking a linear I/O system as input and producing a chemical reaction network as output, and a framework for compiling a finite state machine specification into a gene regulatory network.

Linear I/O systems are a fundamental tool in systems theory, and have been used to design complex circuits and control systems in a variety of settings. In Chapter 2 I present a principled design method for implementing arbitrary linear I/O systems with biochemical reactions. This method relies on two levels of abstraction: first, an implementation of linear I/O systems using idealized chemical reactions, and second, an approximate implementation of the ideal chemical reactions with enzyme-free, entropy-driven DNA reactions. The ideal linear dynamics are shown to be closely approximated by the chemical reactions model and the DNA implementation. The approach is illustrated with integration, gain, and summation as well as with the ubiquitous robust *proportional-integral* (PI) controller.

Finite state machines are fundamental computing devices at the core of many models of computation. In biology, finite state machines are commonly used as models of development in multicellular organisms. In Chapter 3 I describe a method by which *any* finite state machine can be built using nothing more than a suitably engineered network of readily available repressing transcription factors. In particular, I show the mathematical equivalence of finite state machines with a Boolean model of gene regulatory networks. I describe how such networks can be realized with a small class of promoters and transcription factors. To demonstrate the robustness of our approach, I show that the behavior of the ideal Boolean network model approximates a more realistic delay differential equation model of gene expression. Finally, I explore a framework for the design of more complex systems via an example, synthetic bacterial microcolony edge detection, that illustrates how finite state machines could be used together with cell signaling to construct novel multicellular behaviors.

The results presented in this work contribute to both engineering and basic science. To the engineer, these frameworks provide a possible method by which living dynamical and computational systems may be specified and physically realized. To the scientist these frameworks provide a hypothesis about the computational limits of single cells, and a new light in which examine and compare multicellular behavior.

TABLE OF CONTENTS

LIST OF FIGURES

NOTATION

$\mathbb{B}$	Boolean values ($\{0,1\}$, $\{false, true\}$, and $\{off, on\}$ will be used interchangeably, and the intended meaning should be clear from context)
$G(V,E)$	directed graph G with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{v_i \to v_j\}$, where $v_i \to v_j$ in this context means there is a directed edge from vertex v_i to vertex v_j
$\mathbb{N}$	the natural numbers $\{0, 1, 2, \ldots\}$
$\mathbb{R}, \mathbb{R}^+$	the reals, positive reals
$\mathbb{Z}, \mathbb{Z}^+$	the integers, positive integers

ACKNOWLEDGMENTS

My detour into synthetic biology began in 2006 when I joined Professor Eric Klavin's lab with the fuzzy notion that I wanted to learn more about algorithms and control for self-organizing and distributed systems. After all, isn't molecular programming basically like working with an immense number of very tiny, very stupid robots? Since that time I've had the good fortune to collaborate and work with many wonderful people who have helped sustain my enthusiasm for basic research, scientific understanding, and the pursuit of pie-in-the-sky crazy technologies and ideas.

Many thanks go to my committee members, collaborators, and labmates, who have been a wellspring of interesting conversation and ideas. I would specifically like to acknowledge Nils Napp for his continual enthusiasm and for reminding me that I'll always be a roboticist; Josh Bishop, whom I had the pleasure of working with on several in vitro projects; David Soloveichik, who was always willing to share ideas and offer insightful feedback; and Kyle Havens, an outstanding collaborator and wet lab mentor. Additionally, I've benefited greatly from the knowledge and expertise of Rob Egbert, Chris Takahashi, Shelly Jang, Nick Bolton, David Thorsley, and Georg Seelig. My years in Seattle have been terrific, and some of the many people who helped make it so enjoyable are Charlie, Laura, Justin, and Sandra. Of course, I'd also like to thank Kelsey–I can't imagine how different the last few years of my doctoral program would have been without her.

Finally, there's no way I could ever thank my family enough. My sister for her endless support, and my parents whom always encouraged my pursuits, and worked so hard and selflessly to provide my sister and I with educational opportunities.

Chapter 1

BACKGROUND AND MOTIVATION

Natural biological systems are capable of complex and robust behavior. For example, sea shells and plants grow into regular and predictable fractal patterns, starfish regenerate limbs, and a correctly functioning adaptive immune system is a collection of cells that sense antigens from malevolent pathogens and produce a tailored response to eliminate the threat. Underlying these high level behaviors are a collection of growing, dividing, cells running a genetic program specified by genomic DNA.

Engineering novel organisms to simulate existing behavior may improve our understanding of the mechanisms and organizing principles of living systems. Additionally, understanding how to specify and synthesize new behaviors has the potential to transform many industries including medicine and diagnostics, food and agriculture, and energy and biofuel production. Some kind of design theory for biochemical reaction networks and gene regulatory networks ought to be obtainable. Such a design theory would enable the synthesis of chemical reactions and gene regulatory networks from an abstract, human-understandable specification, and allow the designer to identify which behaviors can and cannot be implemented using a particular set of parts and design theory. Synthetic biochemical reaction networks and gene regulatory networks with desired dynamic behaviors are difficult to design because of inherent nonlinearities and substantial uncertainties in reaction mechanisms. Yet natural systems abound in which reliable behavior is obtained from the composition of enormous numbers of molecular subsystems.

Broadly speaking, there exists two types of synthetic biochemical devices: *in vitro* and *in vivo*. Systems synthesized from DNA *in vitro* [1–6] are becoming substantially more complex and reliable. With DNA, well-understood models of hybridization and strand

displacement are available to design and predict molecular interactions [7–9]. Recently, it was shown that *any* physically realistic abstract chemical reaction can be well-approximated by an appropriately designed DNA strand-displacement reaction [10]. Furthermore, similar systems have been demonstrated experimentally: DNA and RNA have been used to design a variety of devices including catalysts and amplifiers [3], logic gates [1], detectors [11], Hopfield associative networks [12], and finite state automata [13]. In contrast, a variety of dynamic devices have been constructed *in vivo*, including toggle switches [14], timers [15], conters [16], oscillators [17], logic gates [18], and band pass filters [19, 20]. Several multi-celled devices relying on cell-cell communication have also been constructed, including synchronized oscillators [21], population control circuits [22], logic networks [23, 24], and striped pattern formation [25]. However, these examples represent one-of-a-kind systems and not general design methodologies for biochemical systems.

1.1 Abstract Models of Multicelled Systems

The task of creating such a design theory is not a new problem. Multi-celled organisms have served as inspiration for computer scientists, roboticists, and theoretical biologists interested in understanding the capabilities and limits of a collection of memory-limited agents or cells using local sensing and communication to effect global behavior. The models introduced in this body of work balance abstraction and simplification against the realities of low level implementations, in an effort to qualitatively model or specify distributed or multicelled behavior without relying on explicit electromechanical or biomolecular devices.

Inspired by the growth patterns of algae, Lindenmayer introduced the *L-system*, a parallel rewriting grammar to formally specify and simulate fractal growth [26, 27]. Lindenmayer's systems contextualized a simple model of cell differentiation and growth in formal languages and computability [28, 29].

Von Neumann and Ulam introduced *cellular automata* and the notion of the *universal constructor* [30] as an extension to Turing's notion of *computational universality* [31] to explore the question: how can machines be made to self-replicate? In the cellular automata

model, cells are finite state machines situated in space on a discrete lattice. The state of all cells evolve synchronously in discrete time steps, and the state of each cell depends on its own state and the state of adjacent cells in the lattice at the previous time step. In this framework von Neumann illustrated sufficient conditions for self-replication and showed that although a Turing machine can be self-replicating, a self-replicating machine need not be Turing universal. Many variations on the cellular automata model followed, including Conway's well-known and biologically inspired, game of Life [32] and Wolfram's systematic study of 1D cellular automata rule sets [33, 34].

The field of *amorphous computing* examines a relaxed version of cellular automata, where cells are scattered randomly in space, and need not be arranged on a discrete and regular lattice [35]. In an amorphous computing system, the state of each cell at a particular time depends on its own state as well as the states of cells within a predefined communication radius at the previous time step. An important contribution from this field are programming languages that compile high-level, human-understandable, pattern formation specifications into sets of local rules that are executed in parallel on every cell in the system [36–41].

Though important for their theoretical contributions, it is difficult to imagine a direct translation from these abstract models to realizable synthetic gene regulatory and biochemical reaction networks. For example, it is not clear how to implement arbitrary finite state machines that are the basis for cellular automata theory, or generate locally unique identifiers and communicate arbitrary data, which are necessary primitives in programming languages for amorphous computing systems. Notably the specification and simulation language `gro` was introduced to address these modeling shortcomings and provide a framework for bridging the gap between synthetic biology and distributed systems theory [42]. However, `gro` is specification and simulation framework, and not a design theory.

1.2 Finite State Machines as a Building Blocks

Finite state machines are a fundamental model of computation, and can be used to represent and reason about many useful machines including counters, adders, and any other

kind of sequential logic. An FSM can be in one of a finite number of states at a given time. An FSM takes as input a string of symbols that belong to a finite set. Upon the arrival of an input symbol, the FSM updates its state according to a *transition function* that depends on the current state of the machine and the current input symbol. As a consequence, an FSM has a memory, and may respond differently to the same input depending on its current state. Although the restriction to finiteness makes FSMs theoretically less capable than other computing machines (for example, pushdown automata or Turing machines), FSMs nevertheless form the basis of most modern models of computation [43]. Additionally, FSMs have been richly explored and applied to engineering problems in electronics, computer architecture, and computer science. In computer science, the FSM formalization is fundamental in categorizing and understanding the limitations of theoretical and physical machines that compute [44–46]. FSMs were fundamental in shaping the design of modern computers where state is typically stored in latches or flip-flops [47, 48].

One of the first forms of finite automata explored in the literature was a model of nerve cell networks investigated by theoretical biologists and mathematicians [49, 50] that was later shown to be equivalent to FSMs [46]. In fact, several synthetic biological mechanisms in the literature have been shown to implement simple finite state machines *in vivo* with just a few states. Broadly speaking, these mechanisms fall into the categories of cells that compute, cells that remember, and cells that communicate. Simple computation using Boolean logic has been demonstrated in multiple organisms with some degree of cascaded modularity [18, 51–53]. Genetic toggle switches have been heavily explored as a mechanisms for remembering stimulus events [14, 54–56]. Similar architectures have been used to implement more complex machines, such as a counter [19] and a timer [15]. Recently, elements of simple computation and memory have been combined to implement all two-input Boolean logic functions in *Escherichia coli* and store the result in DNA over the subsequent 90 cell divisions [57]. Synthetic multicellular systems have employed cell-cell communication mechanisms with simple computing or memory circuits to implement robust and complex behaviors such as synchronized oscillation [21], population density control [22], photosensitive edge detection [58], stripe formation [25], and logic networks [23, 24]. Although these

examples hint at the potential power of synthetic biology to build arbitrary FSMs in cells, a general framework [59] for the design and of synthesis of any given FSM—a sort of 'FSM compiler'—has yet to emerge.

A framework for engineering finite state machines from biomolecular parts would enable the design, construction, and characterization of complex circuits from simple functional parts. In light of the success of simple synthetic circuits, and the availability of libraries of new parts, such as CRISPR, TALE, and Zinc Finger transcription factors [52, 60–67], tunable promoters [68–70], and intercellular communication devices [66,71] I pose the questions: Are these parts sufficient to realize the multi-state behaviors observed in nature? How powerful of a computer can be made using only transcription factors in a GRN? To answer these questions, I demonstrate via explicit mathematical construction that GRNs are, in fact, exactly as powerful as FSMs.

1.3 Overview

Despite success in both engineering synthetic biochemical reaction networks and distributed systems theory, a gap remains connecting the results of mathematical models to the experimental and engineering realities of biological systems. Given a set of tools such as a library of customizable transcription factors, a collection of small molecules that can diffuse through cell walls, and programmable chemical kinetics, how does one take a specification of a multicellular behavior such as leader election, stripe formation, branching, or microcolony edge detection, and compile that into a gene regulatory network and ultimately into a sequence of DNA for insertion into a chassis organism?

In this work I begin to bridge this gap by applying tools of *control theory* and *computer science* to synthesize and analyze design architectures for biomolecular systems, specifically, methods for implementing linear I/O systems and finite state machines. In Chapter 2 I show that any linear I/O system can be built from the composition of three types of reactions, namely catalysis, degradation, and annihilation. I then show how to implement these reactions with DNA devices, but of course, any other programmable system of molecules

could also be used. In Chapter 3 I explore to what extent gene regulatory networks are equivalent to FSMs. I demonstrate a method for implementing any finite state machine with a network of suitably wired repressing transcription factors. Notably, the biomolecular gene regulatory parts we consider consist of only repressing transcription factors. Mathematically, repressing transcription factors by themselves represent a minimal set of part types (i.e. an activating relation can be build from two repressing relations in series). Practically speaking, repressing transcription factors may be easier to engineer than activators. The correctness and effectiveness of the construction is then examined assuming the Boolean network and DDE models for two example systems: a simple two state machine, and a four state modulo-two pulse counter. Coupled with cell division and cell-cell communication, this framework can be used to design a logical control system for cell fate interfacing various biomolecular sensors and effectors. The approach is illustrated through the design and simulation of a bacterial microcolony edge detection system where cells grow, divide, and dynamically differentiate into red fluorescing "edge" cells and non-fluorescing "center" cells. Finally, in Chapter 4 I conclude with possible future research objectives that build naturally on results of this dissertation.

Chapter 2

BIOMOLECULAR IMPLEMENTATION OF LINEAR I/O SYSTEMS

In engineering, the design of dynamic systems from unreliable or poorly modeled basic components is not a new problem. The field of *control systems engineering* is focused on the task of designing dynamic I/O systems that interact with and augment the behavior of unknown or poorly modeled dynamic modules. I have explored the applicability of a standard design theory, *linear I/O systems*, to the design of predictable and robust synthetic biochemical systems. Linear I/O systems are present in almost any engineered device, from stereo amplifiers to aircraft autopilots, and can be implemented electronically, mechanically, and in software. It has been shown that chemical reaction networks can approximate arbitrary polynomial ordinary differential equations [72]. However, prior work in this area lacked a practical, modular, design methodology for implementing a specified ODE, and did not address the implementation of more abstract I/O systems. In this chapter, I show that linear I/O systems can also be implemented with chemical reactions and, in particular, with enzyme-free entropy-driven DNA reactions.

2.1 Biomolecular Device as an I/O System

A biomolecular device can be abstracted to an I/O device taking an input signal, such as a time varying concentration of some chemical species, and producing an output signal. The I/O systems abstraction allows for the *composition* of devices into systems of interacting sub-systems: the output of one device is "wired" to the input of another.

Formally, an I/O system is specified by an input space, output space, an internal state, and a mapping that describes the output signal generated given an input signal and an initial internal state. Often an I/O system corresponds to a modular part of some physical system.

In equations, an I/O system in *state space* is described as [73]

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.1.1}$$

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}), \tag{2.1.2}$$

where $\boldsymbol{u} : \mathbb{R} \to \mathbb{R}^n$ is an input signal, $\boldsymbol{y} : \mathbb{R} \to \mathbb{R}^m$ is an output signal, $\boldsymbol{x} : \mathbb{R} \to \mathbb{R}^p$ is the internal state of the system, and $n, m, p \in \mathbb{Z}^+$. Note that signals such as $\boldsymbol{u}$ are functions of time; however, for simplicity I will write $\boldsymbol{u}$ instead of $\boldsymbol{u}(t)$.

I/O systems may be composed in parallel or in series, to obtain a composite I/O system, by combining output signals or by setting the output signal of one system to be the input signal for another system. Composition can be represented graphically in a *block diagram* by drawing a *block* for each subsystem and representing the connections with directed edges, as illustrated in Fig. 2.1.1a.

Of particular interest are *controllers*. A controller is an I/O system built around a particular system to be controlled (called the "plant"). For example, the controller illustrated in Fig. 2.1.1a is a PI controller built around the plant $P(s)$ with control input u and output y. Often the plant corresponds to a module with unknown or poorly modeled dynamics. The controller is designed to produce an output from the plant that is a desired function of the control input u. Controlled systems can be categorized into *open-loop* and *closed-loop* controllers. In an open-loop system the input to the plant does not depend on its output. In a closed-loop system (such as in Fig. 2.1.1a) the input to the plant is a function of its output, involving a comparison between the input to the controller and the output of the plant. Although open-loop systems are generally easier to analyze than closed-loop systems, feedback allows the engineer to design systems that are robust to modeling uncertainty and errors such as exogenous disturbances and retroactivity [74, 75].

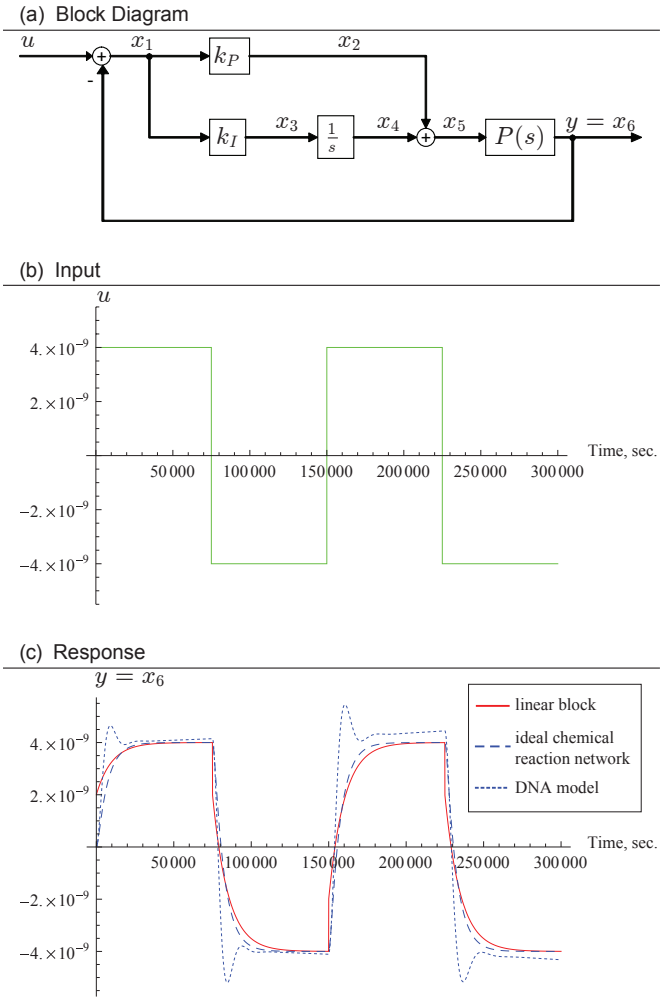


Figure 2.1.1: PI controller block diagram and behavior. (a) Block diagram for a PI controller. The signal u is an input, y is an output signal, and $x_1, \ldots, x_6$ are internal signals. The negative sign next to the edge going into the left summation block means that the output of the summation is $x_1 = u - y$. The PI controller is a feedback system that tracks an input signal over a large class of plants $P(s)$. Here the plant $P(s)$ is implemented with reactions (2.3.22–2.3.23). (b) Input signal driving the PI controller. The input signal u is a square wave. (c) Output trajectories for the ideal PI controller as well as the PI controller implemented with ideal chemical reactions and the DNA model. The steady-state error observed in the DNA model of the PI controller is a result of the sequestration of signal molecule $y^{\pm}$ in intermediate reaction species involved in the left summation block.

2.2 Linear I/O Systems

The most well-understood and useful class of I/O systems are those constructed from linear subsystems. The ubiquitous PI control scheme is composed of linear subsystems. Linear systems are trivially composable—the serial composition of two linear systems is again linear, as are the parallel composition and sum. The analysis of a large linear system is no more difficult than the analysis of its smaller components. Furthermore, although the class of linear systems may appear limiting, essentially all physical systems are approximately linear near their desired operating regions. For this reason linear controllers are used to control a large class of nonlinear plants.

There are two fundamental representations of linear systems, the *state space* representation and the *frequency space* representation. Both representations are useful and complementary in design and analysis. A typical single-input single-output linear system can be represented in state space by

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u \tag{2.2.1}$$

$$y = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}u. \tag{2.2.2}$$

Where $u, y : \mathbb{R} \to \mathbb{R}$, $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are appropriately sized real matrices. The frequency space representation of this system can be found by taking the *Laplace* transform of Equations (2.2.1–2.2.2).

$$s\boldsymbol{X}(s) = \boldsymbol{A}\boldsymbol{X}(s) + \boldsymbol{B}U(s) \tag{2.2.3}$$

$$Y(s) = \boldsymbol{C}\boldsymbol{X}(s) + \boldsymbol{D}U(s) \tag{2.2.4}$$

The frequency space representation is only defined for linear systems. However, in frequency space, linear systems can be analyzed based on their I/O behavior, abstracting away the internal state x through the *transfer function*,

$$\frac{Y(s)}{U(s)} = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}. \tag{2.2.5}$$

Two plants may have different state space representations, yet have identical transfer functions. The transfer function reduces composition and comparison of dynamical systems based on their I/O behavior to simple algebra.

The atomic components of linear I/O systems are linear zero and first order systems. In particular, signal splitting, integration, gain, and summation (see Fig. 2.2.1) form the basis of all linear systems. To apply the results of linear systems to synthetic biology, our task is to implement these basic primitives and describe how they can be physically composed to obtain any arbitrary linear system.

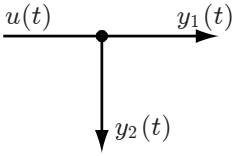
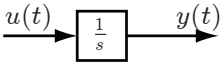
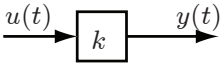
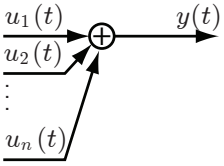
Component Type	Block Diagram	State Space Equations	Transfer Function
Signal Splitting		$\dot{x}(t) = 0$ $y_1(t) = u(t)$ $y_2(t) = u(t)$	$\frac{Y_i(s)}{U(s)} = 1$
Integration		$\dot{x}(t) = u(t)$ $y(t) = x(t)$	$\frac{Y(s)}{U(s)} = \frac{1}{s}$
Gain		$\dot{x}(t) = 0$ $y(t) = ku(t)$	$\frac{Y(s)}{U(s)} = k$
Summation		$\dot{x}(t) = 0$ $y(t) = \sum_{i=1}^{n} u_i(t)$	$Y(s) = \sum_{i=1}^{n} U_i(s)$

Figure 2.2.1: Primitive components of continuous time linear I/O systems represented as a block diagram, state space equations, and frequency space equations.

2.3 Construction of a Chemical Reaction Network from a Linear I/O System Specification

In this section I describe an *intermediate representation* of Linear I/O systems that uses simple, idealized chemical reactions. This level of abstraction requires that notions of signals, integration, gain, and summation blocks be instantiated. This intermediate representa-

tion may then be implemented by a variety of biomolecular engineering frameworks. In the sequel, one possible implementation is discussed where ideal chemical reactions are approximated by enzyme-free DNA devices. Throughout this chapter the *PI controller* [73] illustrated in Fig. 2.1.1a is used both as a design objective and as a running example. However, is through be clear that the approach allows for the implementation of *any* finite dimensional linear system with chemical reactions (and with DNA devices).

2.3.1 Signals Represented as Chemical Concentrations

A natural representation of a signal within a block diagram might be via the time-varying concentration of a particular chemical species. However, concentrations can only be non-negative, whereas signals in arbitrary linear systems take on positive and negative values. Therefore, a signal u is represented by the difference in concentration between two particular chemical species. Specifically, for each signal u there are corresponding chemical species u^+ and u^-.

Remark. I overloaded the symbols u^+ and u^- so that they represent both time varying concentrations and the names of a particular chemical species. It should be clear from the context which usage is intended.

The species u^+ and u^- are referred to as the *positive* and *negative* components of the signal u, respectively, and the actual value of u equals the difference between its components,

$$u \quad = \quad u^+ - u^-. \tag{2.3.1}$$

One implication of this scheme is that no signal value has a unique representation. For example, $u^+ = 100$ M and $u^- = 101$ M represents the same signal value as $u^+ = 0$ M and $u^- = 1$ M.

Definition. The *minimal representation* of u is defined here as the representation where $u^+ = 0$ M or $u^- = 0$ M.

Forcing signals to be represented minimally would be more efficient in an actual implementation of this scheme. To accomplish this forcing, the implementations below include reactions in which the positive and negative components of a signal annihilate each other, as in Equation (2.3.4).

To allow for blocks to be "wired" arbitrarily, it should be that a block has no retroactive effect on its input signals [74]. For example, in electrical implementations of I/O systems, devices are required to draw almost no current from their input signal sources, so that the meaning of a signal is not changed by the devices using it. One way to satisfy this notion is to require that signals act as catalysts for the reactions implementing blocks to which they are wired. In the sequel integration, gain, and summation blocks are shown to be approximated using a minimal set of reaction types: *catalysis*, *degradation*, and *annihilation* reactions, given in (2.3.2), (2.3.3), (2.3.4) respectively.

$$u \rightharpoonup u + y \tag{2.3.2}$$

$$u \rightharpoonup \emptyset \tag{2.3.3}$$

$$u^+ + u^- \rightharpoonup \emptyset. \tag{2.3.4}$$

2.3.2 Integration

An integration block takes as input a signal $u(t)$ and produces the output signal $y(t) = \int_0^t u(\tau)d\tau + y(0)$ with $t \in \mathbb{R}$. The transfer function of an integration block is $1/s$, as shown in Fig. 2.2.1. A chemical reaction network that implements this block is as follows,

$$u^\pm \xrightarrow{\alpha} u^\pm + y^\pm \tag{2.3.5}$$

$$y^+ + y^- \xrightarrow{\eta} \emptyset. \tag{2.3.6}$$

Where $\alpha, \gamma, \eta \in \mathbb{R}^+$. The block consists of two catalysis reactions (2.3.5) and an annihilation reaction (2.3.6).

Remark. I collapsed two reactions, $u^+ \xrightarrow{\alpha} u^+ + y^+$ and $u^- \xrightarrow{\alpha} u^- + y^-$, into reaction (2.3.5). I will use this notation with both $\pm$ and $\mp$ superscripts for brevity.

The catalysis of u^+ and u^- at rate α and annihilation of y^+ and y^- results in the following mass action equations,

$$\dot{u^+} = \dot{u^-} = 0 \tag{2.3.7}$$

$$\dot{y^+} = \alpha u^+ - \eta y^+ y^- \tag{2.3.8}$$

$$\dot{y^-} = \alpha u^- - \eta y^+ y^- \tag{2.3.9}$$

$$\dot{y} = \dot{y^+} - \dot{y^-} = \alpha u. \tag{2.3.10}$$

Note that the bimolecular annihilation reaction drives the concentration of chemical species y^+ and y^- toward the minimal representation of the signal y, and causes the dynamics of y^+ and y^- to be nonlinear. However, the signal dynamics, $y = y^+ - y^-$, remain linear due to the symmetry between $\dot{y}^+$ and $\dot{y}^-$.

2.3.3 Gain and Summation

Gain and summation blocks produce output signals that are linear combinations of their inputs. A gain block takes as input a single signal $u(t)$ and produces the output signal $y(t) = ku(t)$ where $k \in \mathbb{R}$. A summation block takes as input the signals $\{u_i(t)\}_{i=1}^n$, and produces the output signal $y(t) = \sum_{i=1}^n u_i(t)$. The transfer functions for gain and summation are $\frac{Y(s)}{U(s)} = k$ and $Y(s) = \sum_{j=1}^n U_j(s)$ respectively. The following chemical reaction network that outputs a linear combination of its input signals implements both gain and summation,

$$u_i^\pm \xrightarrow{\gamma k_i} u_i^\pm + y^\pm \tag{2.3.11}$$

$$y^\pm \xrightarrow{\gamma} \emptyset \tag{2.3.12}$$

$$y^+ + y^- \xrightarrow{\eta} \emptyset. \tag{2.3.13}$$

Where $k_i, \gamma, \eta \in \mathbb{R}^+$ for $i \in \{1, 2, \ldots, n\}$. In the special case $n = 1$, this chemical representation approximates the gain block in Fig. 2.2.1 for $k \geq 0$. For $n > 1$ this chemical representation approximates the summation block in Fig. 2.2.1. The chemical reaction network consists of $2n$ catalysis reactions (2.3.11), two degradation reactions (2.3.12), and one annihilation reaction (2.3.13). The chemical reaction network gives us the following mass action equations,

$$\dot{u_i^+} = \dot{u_i^-} = 0 \tag{2.3.14}$$

$$\dot{y^+} = \gamma \left(\sum_{i=1}^{n} k_i u_i^+ - y^+ \right) - \eta y^+ y^- \tag{2.3.15}$$

$$\dot{y^-} = \gamma \left(\sum_{i=1}^{n} k_i u_i^- - y^- \right) - \eta y^+ y^- \tag{2.3.16}$$

$$\dot{y} = \gamma \left(\sum_{i=1}^{n} k_i u_i - y \right). \tag{2.3.17}$$

For a constant inputs u_i, the steady state value of y is,

$$\lim_{t \to \infty} y(t) = \sum_{i=1}^{n} k_i u_i. \tag{2.3.18}$$

The chemical representation can be extended to allow negative multiplicative weights. For $k_i < 0$, the catalysis reactions (2.3.11) are replaced with

$$u^{\pm} \xrightarrow{\gamma k_i} u^{\pm} + y^{\mp}. \tag{2.3.19}$$

As before, the annihilation reaction drives the concentration of chemical species y^+ and y^- toward a minimal representation of the signal y without affecting the dynamics of the signal y.

2.3.4 Any Linear I/O System can be Approximated with Ideal Chemical Reactions

Although the dynamics of the chemical representations are not equivalent to the dynamics of integration, gain, and summation, the chemical representations can be used to approximate

the dynamics of integration, gain, and summation. As shown in the prequel, the signal dynamics of the chemical representations are linear systems, which means that they can be compared the I/O behavior of the chemical implementations to ideal integration, gain, and summation, through their transfer functions. In particular, increasing the rate parameter γ for gain and summation results in a closer time-response to ideal gain and summation given a step input. This result is discussed in more detail in Section 2.4.1. I regard a chemical representation of a linear I/O system as an approximation of an ideal linear I/O system if the transfer functions of the two systems can be made equivalent in the limit as the rate parameter γ goes to infinity. The dynamics of the chemical representation of integration, gain, and summation are contrasted with the dynamics of the linear block models of these components in Figure 2.3.1.

Theorem 2.3.1. *All finite-dimensional continuous time linear systems can be approximated with catalysis and degradation reactions.*

Proof. Equation (2.3.10) shows that in the chemical implementation of integration, the output signal y is the integral of αu. This results in the transfer function,

$$\frac{Y(s)}{U(s)} = \frac{\alpha}{s}. \tag{2.3.20}$$

To produce the integration dynamics described in Fig. 2.2.1, set the rate $\alpha = 1$ or compose this system in series with a chemical implementation of gain where $k = \alpha^{-1}$. The equivalent dynamics of the chemical representation where $\alpha = 1$ and an integration block are illustrated in Fig. 2.3.1a.

The transfer functions for the chemical representations of gain and summation are computed from Equation (2.3.17),

$$Y(s) = \frac{\gamma}{s+\gamma} \sum_{i=1}^{n} k_i U_i(s). \tag{2.3.21}$$

The result is a first order stable linear system. Shown in Fig. 2.3.1b and Fig. 2.3.1c, the chemical representation tracks the behavior of the gain block ($n = 1$) illustrated

in Fig. 2.2.1 and weighted summation ($n > 1$) with zero steady-state error for step input signal $u(t)$ and square wave input signals $u_1(t)$ and $u_2(t)$. Increasing γ the I/O behavior of this system can be made arbitrarily close to the I/O behavior of a weighted summation,

$$\lim_{\gamma \to \infty} Y(s) \quad = \quad \sum_{i=1}^{n} k_i U_i(s).$$

To produce the summation dynamics described in Fig. 2.2.1, set $k_i = 1$ for $i = 1, \ldots, n$. A similar result follows for the negative weight extension where reactions (2.3.11) are replaced with reactions (2.3.19) for some set $i \in I \subseteq \{1, \ldots, n\}$.

Because any finite dimensional linear system can be decomposed into integration, gain, and summation blocks the chemical reaction representation of linear I/O systems can be used to approximate any finite dimensional linear I/O system. It follows directly that as γ goes to infinity, the transfer function of an implemented chemical system approaches that of the ideal specification. □

Two details of the ideal chemical reaction model may be particularly concerning when implementing with biomolecules: First, the rate γ will be physically constrained to some (very large) finite value. Second, it may be impossible to precisely match reaction rates between chemical reactions as specified, for example, by the reaction pairs (2.3.11) and (2.3.12). In simulation, realistic values of γ can produce a time-response close to the specified system for step inputs, as illustrated in Fig. 2.3.1 and Fig. 2.1.1c for integration, gain, and summation components as well as the more complicated PI controller. If reaction rates involving positive and negative components do not match, then for fast annihilation reaction rates, η, the signal dynamics of the resulting system can be shown to be close to those of a related linear switch system. These observations and results are detailed in Section 2.4 where the robustness and sensitivity of the construction to different modeling parameters and chemical disturbances are discussed.

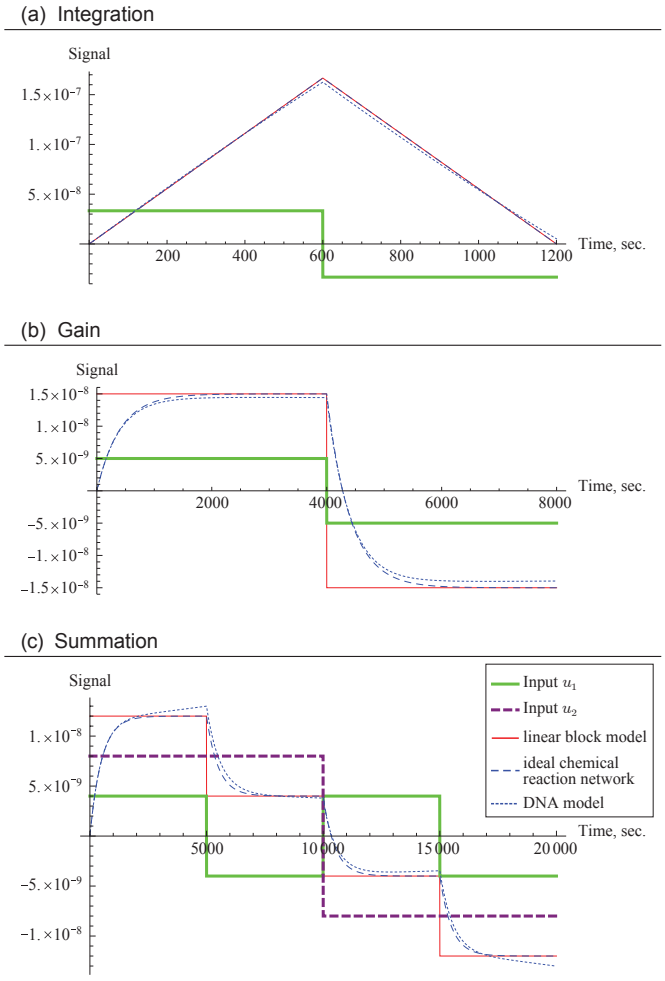


Figure 2.3.1: Step response of the linear block model, chemical reaction representation, and DNA model of integration, gain, and summation blocks. For each system the input u_1 is a square wave. (a) Integration block. The linear block model follows the trajectory $y(t) = \int_0^t u_1(t)$. The ideal chemical reaction representation follows this trajectory precisely. The DNA model drifts from the ideal chemical reaction trajectory as molecular fuel species are consumed. (b) Gain block. The linear block model follows the trajectory $y(t) = 3u_1(t)$. The chemical reaction representation produces the correct steady-state output. As with integration, the DNA model closely follows the ideal chemical reaction trajectory, but drifts as fuel species are consumed. (c) Summation block. The linear block model follows the trajectory $y(t) = u_1(t) + u_2(t)$. Given inputs u_1 and u_2 the output should consist of four monotonically decreasing steps. The chemical reaction representation follows each step in steady-state. As before, the DNA model drifts from the ideal chemical reaction representation as fuel species are consumed.

2.3.5 A Simple Optimization: Weighted Integration and Summation

The approximation of a linear system by a chemical reaction network can be made arbitrarily good by tuning the rate parameter γ. However, as mentioned, in practice it is not possible to assign arbitrary values to γ. A time response closer to the ideal system may be obtained by exploiting other parameters and decreasing the order of the chemical reaction model. For example, a gain composed with an integrator as illustrated in Fig. 2.3.2a is a first order system. The unoptimized chemical representation (Fig. 2.3.2b) is a second order system, however this system can be implemented precisely using an optimized first order weighted integration reaction (Fig. 2.3.2c). Illustrated in Fig. 2.3.2d, given a step input, the unoptimized representation of this system results in steady-state error (which decreases as γ increases), while the optimized representation replicates the dynamics of the ideal I/O system exactly.

In general two other optimizations can be made in the chemical representation: First, the weighted summation of n signals may be approximated by the second order representation of n summation and gain reaction sets, or a first order weighted summation reaction network. Second, the weighted or unweighted summation of n integrated signals may be approximated by second or third order representation of integration composed with gain (in the weighted case) and summation reaction sets, or implemented precisely by n weighted integration reaction sets, all of which share the same output species. In addition to reducing the overall order and output error of the approximation, these optimizations minimize the total number of species needed to implement a linear I/O system with catalysis, degradation, and annihilation reactions. These optimizations are used in the following example.

2.3.6 Example: Ideal Chemical Reaction Network Implementation of a PI Controller

To illustrate the method, the PI controller shown in Fig. 2.1.1a is constructed here. The PI controller is an example of a closed loop linear system designed to drive the output of the plant P to a desired set point u. The key feature of the PI controller is that it tracks any step input u with zero steady-state error for a large class of plants. Such a device could

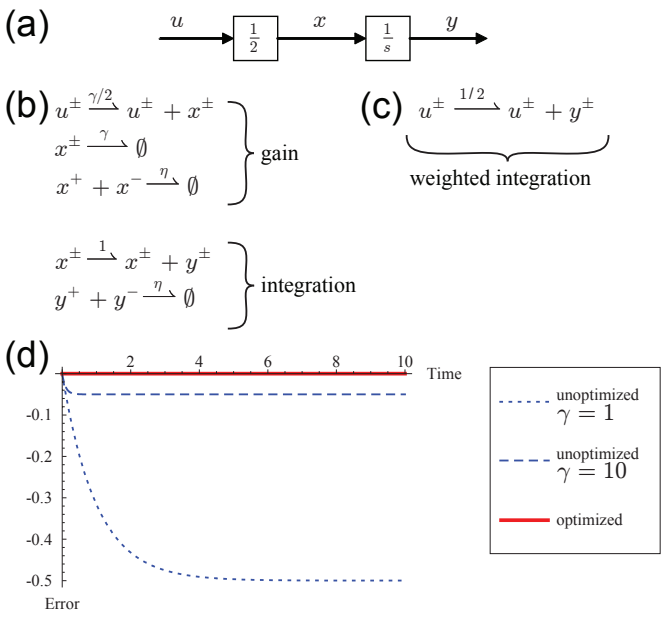


Figure 2.3.2: Approximation error for optimized and unoptimized ideal chemical reaction representations of the I/O system $\dot{x} = \frac{1}{2}u$, $y = x$. (a) Block diagram representing the ideal weighted integration system. (b) Unoptimized chemical reaction representation. This representation consists of three pairs of signal species, a gain block and an integration block. The signal dynamics resulting from mass action kinetics is a second order linear system. (c) Optimized chemical reaction representation. This representation consists of two pairs of signal species and a single weighted integration block. The signal dynamics resulting from mass action kinetics is a first order linear system that matches the dynamics of the ideal system. (d) Error trajectory for the signal y given $u(0) = u^+(0) - u^-(0) = 1$ and $x(0) = y(0) = 0$. The unoptimized chemical reaction representation of the weighted integration system results in some nonzero steady-state error which decreases monotonically as γ increases. The optimized chemical reaction representation results in zero steady state error.

be useful, for example, in regulating a fuel species driving a variety of downstream devices. Suppose the plant P is realized by leaky expression and chemical devices that produce and consume the signal species $x_5^{\pm}$ according to the reactions,

$$\emptyset \xrightarrow{\gamma\delta_1} x_5^{\pm} \tag{2.3.22}$$

$$x_5^{\pm} \xrightarrow{\gamma\delta_2} \emptyset, \tag{2.3.23}$$

for $\delta_1, \delta_2 \in \mathbb{R}^+$. These disturbances can be used to model leaky expression of signal molecules, or the retroactive effect of molecular devices the engineer wishes to control. For now, I claim that the effect of Equations (2.3.22–2.3.23) are captured by the plant $P(s) = (1 + \delta_2)^{-1}$ (this claim is later supported in Section 2.4.3). Note that δ_1 does not appear in this expression as a result of the symmetric effect Equation (2.3.22) has on molecular species x_5^+ and x_5^-. It is a well known result from control theory that for a step input $u(t)$, the output of the PI controller is robust to multiplicative plants $P(s)$ [73], meaning that for any multiplicative plant the controller should track the input signal $u(t)$ with zero steady-state error.

In general, there are two steps to compiling a biochemical controller from a block diagram. First, signals in the block diagram are enumerated, and it is determined which signals correspond to chemical species. Second, primitive blocks are instantiated by sets of chemical reactions. The PI controller in Fig. 2.1.1a consists of signals u, y, x_1,..., x_5, x_6 where u is the input signal and $y = x_6$ is the output signal. Using optimizations from the prequel, PI controller is approximated using species $u^{\pm}$, $x_1^{\pm}$, $x_4^{\pm}$, $x_5^{\pm}$ in the chemical reactions outlined in Fig. 2.3.3. The signal dynamics of the chemical realization is a second order linear approximation of the first order PI controller with the following signal dynamics,

$$\dot{x}_1 = \gamma(u - x_5 - x_1) \tag{2.3.24}$$

$$\dot{x}_4 = k_I x_1 \tag{2.3.25}$$

$$\dot{x}_5 = \gamma((k_P x_1 + x_4 + \delta_1) - (1 + \delta_2)x_5). \tag{2.3.26}$$

Demonstrated in Fig. 2.1.1, this controller produces the stable output $\lim_{t\to\infty} y(t) = u(t)$ for any step input $u(t)$ when $\delta_1, \delta_2 \in \mathbb{R}^+$. In fact, the chemical realization of the PI controller with a production and degradation disturbance can track a square wave input with zero steady-state error. Additionally, as $\gamma \to \infty$, the transient dynamics of the chemical controller approach those of the ideal PI controller.

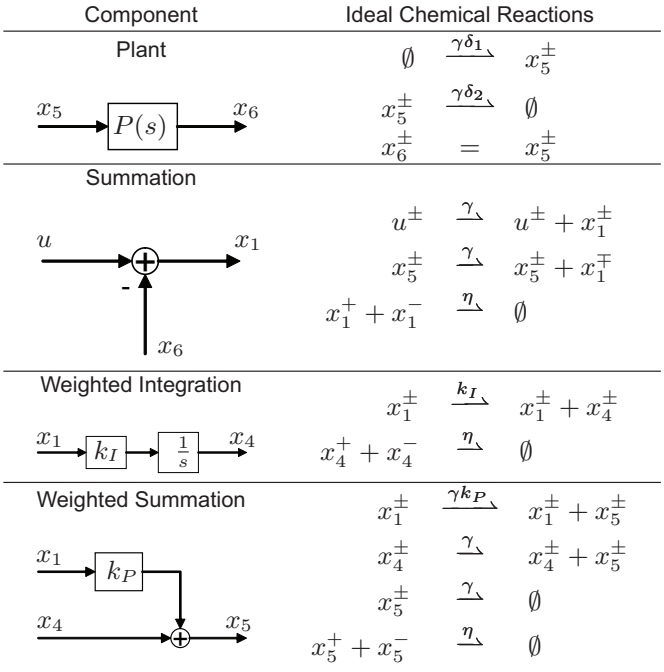


Figure 2.3.3: PI controller from Fig. 2.1.1a implemented in ideal chemical reactions.

2.4 Robustness and Sensitivity of Modeling Parameters and Disturbances in the Ideal Chemical Reaction Model

2.4.1 The Role of γ in the Time Domain

In practice the rate γ in summation and gain (Equations (2.3.11–2.3.13)) is bounded by physical constraints such as binding affinity, reaction temperature, and saturation point. The steady-state value for gain and summation given a step input is invariant to γ. However,

increasing γ for summation and gain given a step input has the effect of driving the system towards steady-state faster.

Claim 2.4.1. *For step input u, the time it takes for the reaction network representing summation or gain to reach the steady-state value y^* scales as the inverse of γ.*

Proof. Let $\{u_i\}_{i=1}^n$ be a set of constant inputs to a weighted summation chemical reaction network with weights $\{k_i\}$. The state space equations for this systems are,

$$\begin{aligned}
\dot{u_i^+} &= \dot{u_i^-} = 0 \\
\dot{y^+} &= \gamma\left(\sum_{i=1}^{n} k_i u_i^+ - y^+\right) - \eta y^+ y^- \\
\dot{y^-} &= \gamma\left(\sum_{i=1}^{n} k_i u_i^- - y^-\right) - \eta y^+ y^- \\
\dot{y} &= \gamma\left(\sum_{i=1}^{n} k_i u_i - y\right).
\end{aligned}$$

Let y^* be the steady-state value of y. The the dynamics of y have an analytic solution,

$$y(t) = e^{-t\gamma}\left(y(0) - y^*\right) + y^*.$$

Let $w \in (0, 100)$ denote the percent progress of $y(t)$ to steady state from initial conditions $y(0)$ Given constant set of inputs $\{u_i\}$,

$$\begin{aligned}
y(t) - y(0) &= \frac{w}{100}\left(y^* - y(0)\right) \\
0 &= \left(y(0) - y^*\right)\left(e^{-t\gamma} + \frac{w}{100} - 1\right) \\
t &= \frac{1}{\gamma}\ln\frac{100}{100 - w}.
\end{aligned}$$

Summation and gain blocks are special cases of weighted summation, therefore, for step input u, the time it takes to reach $w\%$ of the steady-state value y^* scales as the inverse of γ. □

2.4.2 Fast Annihilation and Imperfect Rate Matching in the Chemical Realization of Integration, Gain, and Summation

One potential pitfall of treating each signal as the difference in concentration between two molecular species is the need to match rate parameters between chemical reactions. For example, the chemical realization for integration (Equations (2.3.5–2.3.6)) relies on the rate parameter α to be the same for two separate chemical reactions. Worse yet, the chemical realization for summation and gain (Equations (2.3.11–2.3.13)) requires both γ and k_i to be the same for multiple reactions, and any difference results in nonlinear signal dynamics. In practice it may only be possible to guarantee close reaction rates. One solution to this problem is to require fast annihilation rates $\eta \gg \alpha\gamma, k_i$ and annihilation reactions for all inputs $u^{\pm}$. With these requirements the chemical realization for integration is written,

$$u^+ + u^- \xrightarrow{\eta} \emptyset \tag{2.4.1}$$

$$u^{\pm} \xrightarrow{\alpha^{\pm}} u^{\pm} + y^{\pm} \tag{2.4.2}$$

$$y^+ + y^- \xrightarrow{\eta} \emptyset. \tag{2.4.3}$$

Assuming $\eta \gg \alpha$, the signal dynamics can be approximated as a switched linear system [76] utilizing time-scale separation [77],

$$\dot{u_i^+} = \dot{u_i^-} = -\eta u^+ u^- \approx 0 \tag{2.4.4}$$

$$\dot{y} = \dot{y^+} - \dot{y^-} \approx \alpha^+ u^+ - \alpha^- u^- \tag{2.4.5}$$

$$\approx \begin{cases} \alpha^+ u, & u > 0 \\ \alpha^- u, & u \leq 0. \end{cases} \tag{2.4.6}$$

Similarly, for summation and gain, the full chemical realization with annihilation is written,

$$u^+ + u^- \xrightarrow{\eta} \emptyset \tag{2.4.7}$$

$$u_i^\pm \xrightarrow{\gamma^\pm k_i^\pm} u_i^\pm + y^\pm \tag{2.4.8}$$

$$y^\pm \xrightarrow{\gamma^\pm} \emptyset \tag{2.4.9}$$

$$y^+ + y^- \xrightarrow{\eta} \emptyset, \tag{2.4.10}$$

for $i = 1, 2, \ldots, n$. Assuming $\eta \gg \gamma, k_i$, the signal dynamics can again be approximated as a switched linear system. For compactness, consider the case $n = 1$ for the ideal chemical representation of a gain block,

$$\dot{u_i^+} = \dot{u_i^-} = -\eta u^+ u^- \approx 0 \tag{2.4.11}$$

$$\dot{y^+} = \gamma^+ \left(k^+ u^+ - y^+\right) - \eta y^+ y^- \tag{2.4.12}$$

$$\dot{y^-} = \gamma^- \left(k^- u^- - y^-\right) - \eta y^+ y^- \tag{2.4.13}$$

$$-\eta y^+ y^- \approx 0 \tag{2.4.14}$$

$$\dot{y} \approx \begin{cases} \gamma^+(uk^+ - y) & u > 0 \wedge y > 0 \\ \gamma^+ uk^+ - \gamma^- y & u > 0 \wedge y \leq 0 \\ \gamma^-(uk^- - y) & u \leq 0 \wedge y > 0 \\ \gamma^- uk^- - \gamma^+ y & u \leq 0 \wedge y \leq 0. \end{cases} \tag{2.4.15}$$

In general linear switched systems are more difficult to analyze than non-switched linear systems. However, for many stable systems, it is possible to compute bounds the behavior of the switched linear system in terms of the ideal non-switched linear system. Illustrated in Fig. 2.4.1, in simulation, the PI controller performs well even under $\pm 10\%$ variation in reaction rates.

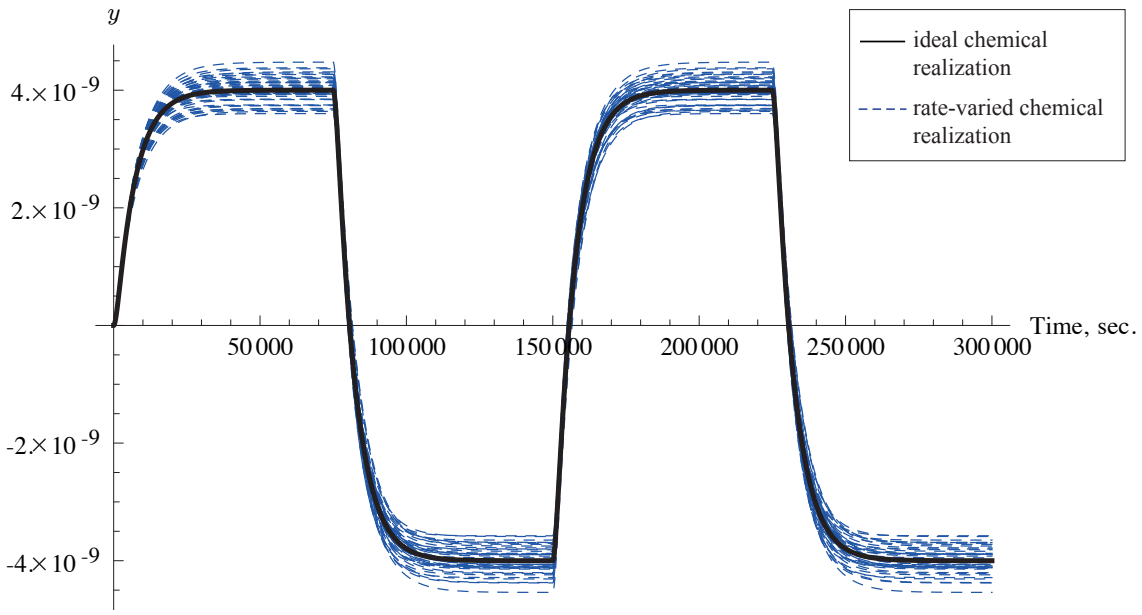


Figure 2.4.1: Output trajectories from chemical realizations of the PI Controller from Figure 2.1.1. The ideal chemical realization matches reaction rates between pairs of reactions. Rate-varied chemical realization output trajectories were obtained by varying the reaction rates $\pm 10\%$ from the ideal reaction rates randomly with a uniform distribution over 50 simulations.

2.4.3 The Effect of Production and Degradation of Signal Species on the Chemical Realization of a Linear I/O System

Production and degradation disturbances (described in Equations (2.3.22) and (2.3.23) respectively) model the effect of unregulated chemical devices or leaky expression on a chemical realization of a linear I/O system. This section discusses the effect of these disturbances on the output signals for the chemical realizations of integration and weighted summation, which generalize to weighted integration, summation, and gain reaction networks.

Claim 2.4.2. *The effect of a chemical disturbance on the output of an ideal chemical reaction implementation of integration or weighted integration is equivalent to replacing the integration with a gain, or weighted integration with a weighted summation.*

Proof. The chemical reaction network describing integration with input u and a production- and degradation-disturbed output y is as follows,

$$u^{\pm} \xrightarrow{\alpha} y^{\pm} \tag{2.4.16}$$

$$u^{+} + u^{-} \xrightarrow{\eta} \emptyset \tag{2.4.17}$$

$$y^{+} + y^{-} \xrightarrow{\eta} \emptyset \tag{2.4.18}$$

$$\emptyset \xrightarrow{\gamma\delta_1} y^{\pm} \tag{2.4.19}$$

$$y^{\pm} \xrightarrow{\gamma\delta_2} \emptyset. \tag{2.4.20}$$

Letting $\alpha = k\gamma$, the mass action kinetics of the system can then be written,

$$\dot{u^{+}} = \dot{u^{-}} = 0 \tag{2.4.21}$$

$$\dot{y^{+}} = \gamma(ku^{+} + \delta_1 - \delta_2 y^{+}) - \eta y^{+} y^{-} \tag{2.4.22}$$

$$\dot{y^{-}} = \gamma(ku^{-} + \delta_1 - \delta_2 y^{-}) - \eta y^{+} y^{-} \tag{2.4.23}$$

$$\dot{y} = \dot{y^{+}} - \dot{y^{-}} = \gamma(ku - \delta_2 y). \tag{2.4.24}$$

The transfer function of this system as $\gamma \to \infty$ is then,

$$\lim_{\gamma \to \infty} \frac{Y(s)}{U(s)} = \frac{k}{\delta_2}. \tag{2.4.25}$$

The resulting transfer function is equivalent to the transfer function for a gain block. For weighted integration, this result generalizes to the transfer function of weighted summation. □

Claim 2.4.3. *The effect of a chemical disturbance on the output of an ideal chemical reaction implementation of gain or weighted summation is to change the value of the gain or the value of all of the weights by a factor of* $(1 + \delta_2)^{-1}$

Proof. The chemical reaction network describing weighted summation with inputs u_i and weights k_i (for $i = 1, 2, \ldots, n$), and a production- and degradation-disturbed output y is as follows,

$$u_i^{\pm} \xrightarrow{\gamma k_i} y^{\pm} \tag{2.4.26}$$

$$y^{\pm} \xrightarrow{\gamma} \emptyset \tag{2.4.27}$$

$$u^+ + u^- \xrightarrow{\eta} \emptyset \tag{2.4.28}$$

$$y^+ + y^- \xrightarrow{\eta} \emptyset \tag{2.4.29}$$

$$\emptyset \xrightarrow{\gamma \delta_1} y^{\pm} \tag{2.4.30}$$

$$y^{\pm} \xrightarrow{\gamma \delta_2} \emptyset. \tag{2.4.31}$$

The mass action kinetics of the system can then be written,

$$\dot{u_i^+} = \dot{u_i^-} = 0 \tag{2.4.32}$$

$$\dot{y^+} = \gamma\left(\sum_{i=1}^{n} k_i u_i^+ - (1+\delta_2)y^+ + \delta_2\right) - \eta y^+ y^- \tag{2.4.33}$$

$$\dot{y^-} = \gamma\left(\sum_{i=1}^{n} k_i u_i^- - (1+\delta_2)y^- + \delta_2\right) - \eta y^+ y^- \tag{2.4.34}$$

$$\dot{y} = \gamma\left(\sum_{i=1}^{n} k_i u_i - (1+\delta_2)y\right). \tag{2.4.35}$$

The transfer function of this system as $\gamma \to \infty$ is then,

$$\lim_{\gamma\to\infty} Y(s) = (1+\delta_2)^{-1}\sum_{i=1}^{n} U_i(s)k_i. \tag{2.4.36}$$

The resulting transfer function is equivalent to the transfer function for weighed summation where all gains k_i are decreased by a factor of $(1+\delta_2)^{-1}$. The result applies to gain blocks as well, since the ideal chemical representation of a gain block is simply a weighted summation where $n = 1$. □

2.5 Mapping Integration, Summation, and Gain to DNA Strand Displacement Reactions

The ideal chemical reactions representation of arbitrary linear systems is a useful template which can be used to guide the implementation of arbitrary linear systems in physical substrates. Choosing a particular biomolecular implementation forces us to consider physical constraints such as limited supplies of energy-storing fuel molecules and finite maximum reaction rates.

In enzyme-free DNA reactions, energy can be stored in metastable DNA molecules [1,3,4,78]. Chemical reactions are driven by the transformation of metastable fuel into nonreactive waste products, and the reaction pathways are programmed through the sequence of nucleic acids and the availability of single-stranded binding sites, or "toeholds". Toehold-mediated

branch migration and strand displacement reactions are quantitatively well-understood [7–9], and many complex devices have been constructed around this design principle [1–4]. Recently it was shown that a DNA-based schema built on these design principles may approximate a large class of unimolecular and bimolecular chemical reactions [10]. Here, I illustrate and simulate a DNA-based implementation of catalysis, degradation, and annihilation (Fig. 2.5.1), as well as the PI controller introduced in Example I, implemented using the schema of Soloveichik et al. Details of the implementation, simulations, and conditions under which the implementation may be considered valid are given in Appendix Sections A.1 and A.2.

Following the notation from Soloveichik et al. [10], in this implementation of chemical reaction primitives for linear I/O systems set q_{max} to be the maximum strand displacement rate and assume $q_i \ll q_{max}$. All reactions are entropy-driven with potential energy stored in fuel species G_i, T_i, L_i, B_i, LS_i, and BS_i. Fuel species are assumed to appear in initial concentration C_{max}. Numbers label *domains*, which are unique sequences of nucleotides. In this parameterizing scheme, two labeled domains are complementary if and only if their labels are x and x^* respectively. Of note is the domain 1^*_q, which denotes a subsequence of the domain 1^* with length tuned to the reaction rate q_i.

Signal molecules in a given linear system correspond to single stranded DNA made uniquely addressable via a sequence of three domains on the $5'$ end. Signal molecules catalyze the degradation of fuel species into noninteracting waste molecules. As with any entropy-driven system the trivial steady-state occurs where the fuel species have degraded into nonreactive waste. However, in order to approximate the ideal chemical reactions, the operating regime is limited to where the concentration of fuel species is much greater than the concentration of signal molecules. As shown in Fig. 2.5.1, the dynamics of the DNA system approach those of the ideal chemical reaction as C_{max} increases.

Fig. 2.5.1a illustrates the production of signal molecule y and degradation of fuels G_i and T_i catalyzed by the signal molecule u. For large C_{max} the concentration $y(t)$ is approximately the integral of the concentration $u(t)$. Fig. 2.5.1b shows the degradation of fuel species G_i driven by the presence of signal molecule u. Again, for large C_{max} the concentration tra-

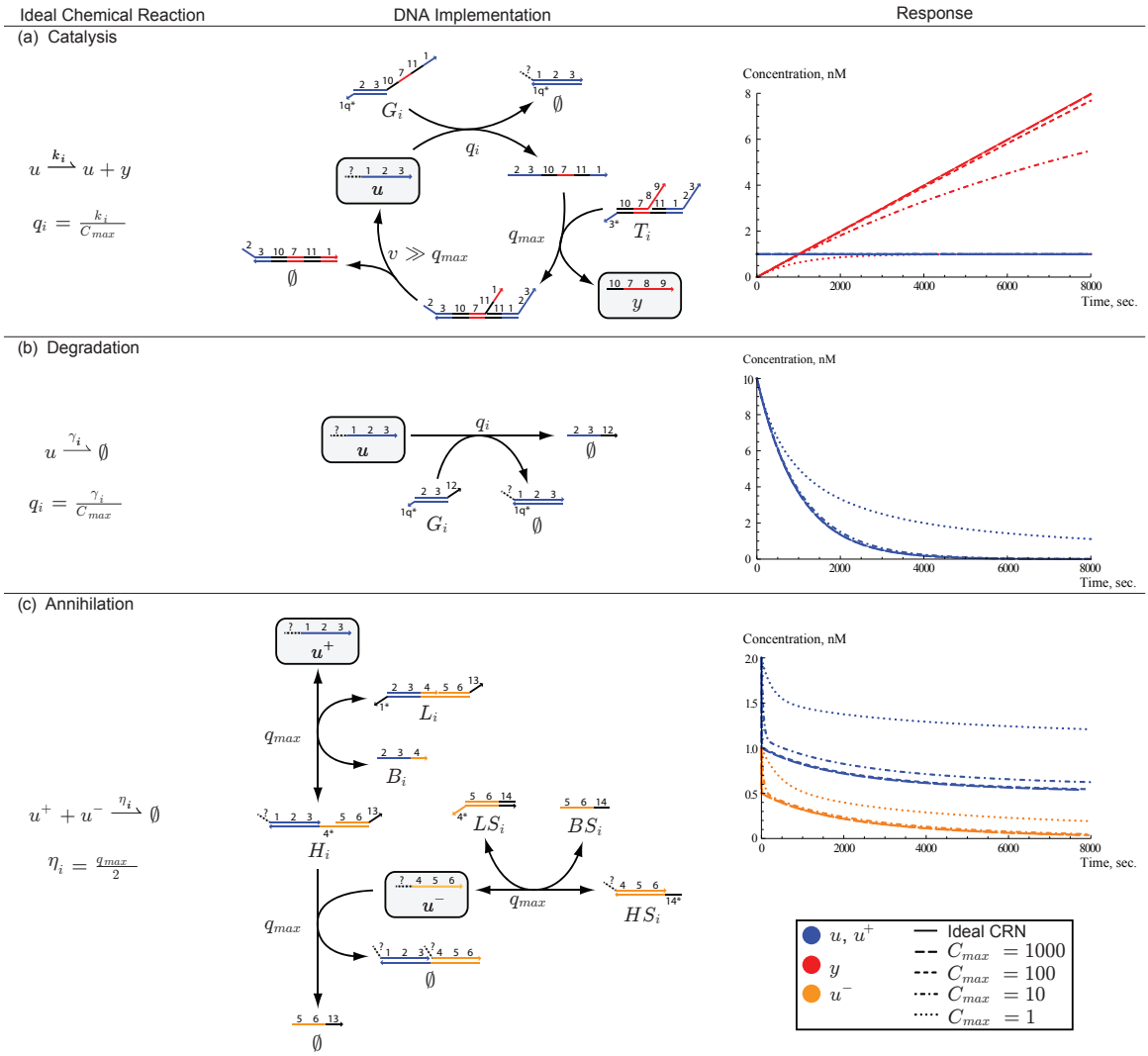


Figure 2.5.1: Ideal chemical reaction, DNA implementation, and signal response for (a) catalysis, (b) degradation, and (c) annihilation reactions. The domain 1_q is a subset of the domain 1. The initial concentration of fuel species G_i, T_i, L_i, B_i, LS_i, and BS_i are set to C_{max} = 1 nM, 10 nM, 100 nM, 1000 nM. For the catalysis and degradation response, $u(0) = 1$ nM. For the annihilation response, $u^+(0) = 1\zeta$ nM, $u^-(0) = 0.5\zeta$ nM where $\zeta = 2$ is a scaling factor that attenuates for the initial fast transient where u^+ and u^- are sequestered in intermediate species. All other initial concentrations are set to zero.

jectory $u(t)$ is approximately exponential decay. Lastly, Fig. 2.5.1c shows the degradation of fuel species L_i and LS_i driven by the presence of both u^+ and u^- molecules. Note that the dynamics of this reaction can be separated into fast dynamics where u^+ and u^- are sequestered in intermediate species through their reactions with L_i and LS_i respectively, and slow dynamics where u^- degrades into waste through its interaction with the intermediate species that sequestered u^+. Since it is the slow dynamics that approximate the ideal annihilation reaction, the initial concentration of u^+ and u^- must be scaled to attenuate for sequestering effect of the fast dynamics. That is, the initial concentration of all unregulated signal molecules must be scaled by a factor of two in order to approximate the ideal chemical reaction network. Again, the approximation of ideal annihilation improves for large C_{max}. Simulations of integration, gain, and summation blocks scaled to realistic parameters are given in Fig. 2.3.1. Details of the implementation and simulations are given in Appendix Section A.2.

2.5.1 Example: DNA PI Controller

Here, the PI controller illustrated in Fig. 2.1.1a is implemented with the DNA instantiation of catalysis, degradation, and annihilation. As with the ideal chemical reaction representation, the plant P is realized by the chemical reactions (2.3.22–2.3.23). Note that as before, this disturbance models leaky expression of a signal molecule, and downstream load on the output signal. This realization is based on the optimized chemical representation shown in Fig. 2.3.3. The step response of the DNA implementation of the PI controller in the face of the plant P are shown in Fig. 2.1.1c. Note that as discussed earlier, the unregulated input u was scaled in order to attenuate the sequestering effect of the annihilation reaction. Note that the output trajectory of the DNA implementation of the PI controller closely matches the output trajectory of the ideal chemical reaction implementation discussed in Example I at the beginning of the simulation, tracking the input u with near zero steady-state error. The drift away from zero steady-state error as time increases is due to the consumption of finite fuel molecules modeled in the DNA implementation. Details of the implementation and simulation are given in Appendix Section A.3.

2.6 Discussion

I demonstrated a method for realizing arbitrary linear I/O systems at two levels of abstraction: 1. an intermediate chemical reaction representation, and 2. a proposed DNA implementation. The intermediate chemical reaction representation of linear I/O systems provides a template for implementation using arbitrary biomolecules. Notably our construction of linear I/O systems relies on only three types of chemical reactions: catalysis, degradation, and annihilation.

Although an implementation of linear I/O systems using DNA has been explored in some depth, one could imagine an implementation of these reactions using a variety of different substrates: gene regulatory networks, MAPK cascades, or some combination of these systems [19, 20, 79–82]. The DNA implementation provides a specific avenue for composing existing DNA devices, and case study with which to examine experimental considerations such as finite chemical concentrations, realistic reaction rates, unmodeled chemical reactions, and data collection.

2.7 Methods

All simulations are performed in *Mathematica*, Version 7.0.1.0 [83], with numerical solver *NDSolve*. *Mathematica* files are available upon request. Specific details of the ideal chemical realization and DNA implementation of linear I/O primitives as well as the PI controller, including chemical reaction network models and reaction rates, are given in Appendix A.

Chapter 3

A FRAMEWORK FOR IMPLEMENTING FINITE STATE MACHINES IN GENE REGULATORY NETWORKS

An engineering framework for *finite state machines* (FSMs) in living cells is crucial scientifically and practically. Naturally occurring finite state machines control how cells switch states from stem cells to tissue specific cells according to chemical, mechanical, and logical cues from their local environments [84–87]. However, many questions remain unanswered: How is state encoded by patterns of gene expression? How are states stabilized to avoid spontaneous switching? How do state-specific signals reliably cause transitions between states? Most importantly, can a synthetic biologists build novel finite state machines in synthetic cells that control the process of differentiation and development? Here, I illustrate a general approach for designing and building finite state machines in living cells. The design method is rooted in the rich theory of finite state machines and sequential logic from computer engineering [29, 46]. I apply the method to a variety of examples modeled at different levels of detail and abstraction: as a Boolean network representation of a *gene regulatory network* (GRN), as *delay differential equations* (DDEs) modeling a biomolecular implementation of the GRN, and finally as a 2D multicellular simulation illustrating a complex microcolony edge detection behavior implemented from a high-level finite state machine specification.

3.1 Finite State Machines

Finite state machines are a well understood and intuitive model of state control and computation. A finite state machine is specified by the tuple,

$$M = (Q, \Sigma, \delta, q_0, F), \tag{3.1.1}$$

where Q and Σ are finite sets, $\delta : Q \times \Sigma \to Q$, $q_0 \in Q$, and $F \subseteq Q$. Q is a set of *states*, Σ is a set of *input symbols*, q_0 is the unique *start state*, δ is a *state transition function*, and F is a (possibly empty) set of *accepting states*. Finite state machines are typically represented as a directed graph. For example, Figure 3.1.1A depicts a two-state FSM. The set of states $Q = \{0, 1\}$ are represented by labeled vertexes. The start state $q_0 = 0$ is denoted by the "start" arrow, and the set of accepting states $F = \{1\}$ are illustrated as a double-circled vertex. Edge labels denote the full set of inputs $\Sigma = \{a, b\}$. The state transition map δ is represented by edges labeled with a list of input symbols that move the FSM between states.

The semantics of a finite state machine is then a set of rules defining its operation specified by M. The FSM takes as input a sequence of symbols $w \in \Sigma^*$ (where Σ^* is the Kleene operator applied to Σ), and determines weather or not the sequence is in a set of accepted sequences through a series of state transitions. The FSM begins in the initial state q_0. Given a sequence of inputs $w = \sigma_1\sigma_2 \ldots \sigma_n$ with $\sigma_i \in \Sigma$ for $i = 1, 2, \ldots, n$, the first state the machine transitions to is $q_1 = \delta(q_0, \sigma_1)$. The k^{th} state the FSM transitions to is then $q_k = \delta(q_{k-1}, \sigma_k)$. The w is an *accepted sequence* if $q_n \in F$. If $q_n \notin F$ or if for some k, $\delta(q_{k-1}, \sigma_k)$ is not defined, the FSM does not accept w. In other words, w is accepted if and only if there is a path through the graph induced by M that begins at q_0 and ends at $q_n \in F$ where consecutive edges in the path have labels that contain $\sigma_1, \sigma_2, \ldots, \sigma_n$.

In the sequel, I present a general method for constructing a GRN with a small number of component types to realize an arbitrary FSM specification. The method is illustrated through example, using the two-state machine in Figure 3.1.1, and show how the Boolean network model of the GRN implements the two-state FSM. Additionally, I show that a biomolecular realization of the GRN modeled by DDEs closely approximates the Boolean network dynamics for a wide range of physically realistic parameters. Finally, I demonstrate how finite state machines can be used to engineer complex multicellular behaviors with a bacterial microcolony edge detection example simulated in `gro` [42].

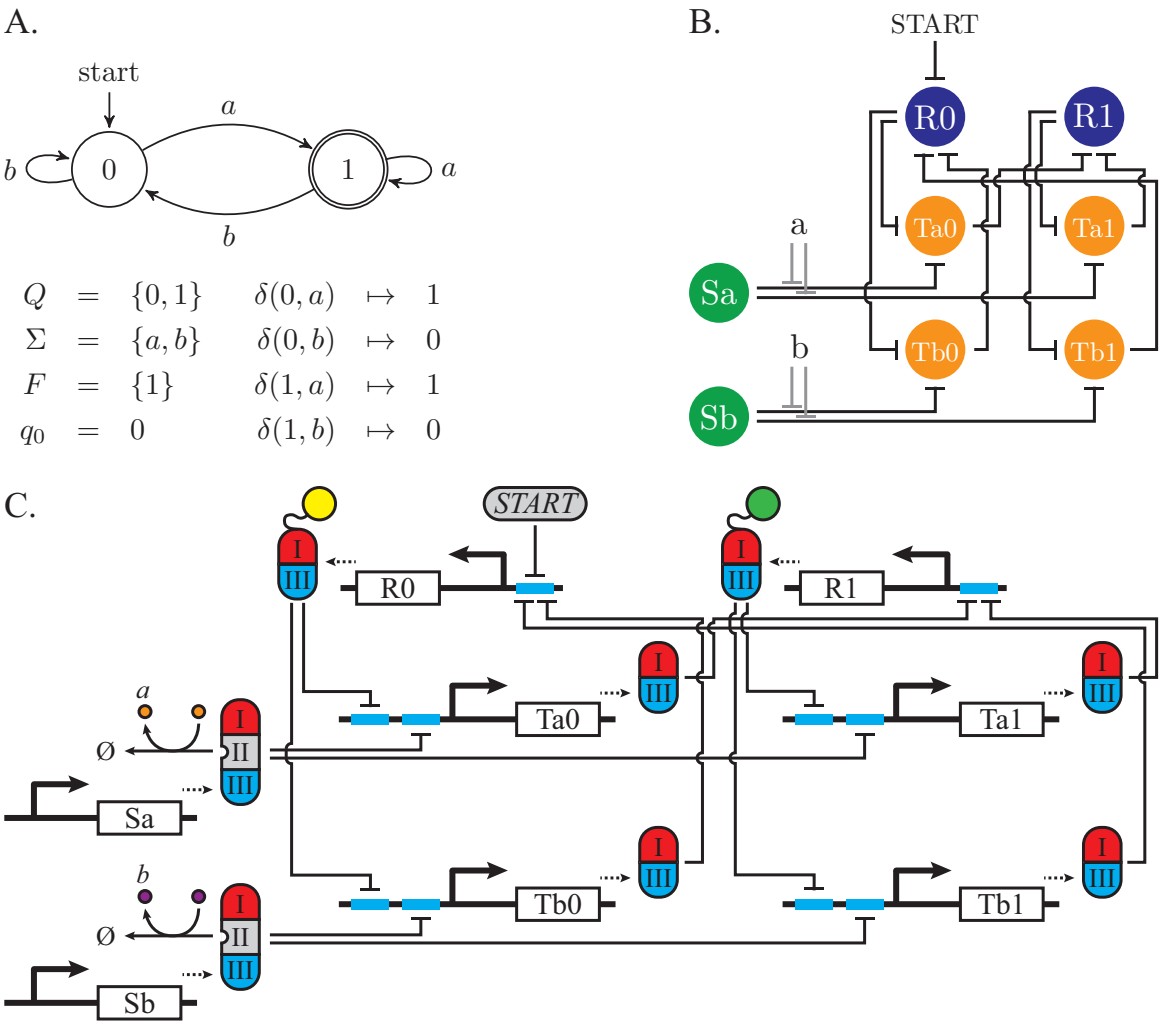


Figure 3.1.1: Simple two-state machine described as (A) a directed graph representation of a finite state machine, (B) a gene regulatoy network made of repressing transcription factors and inducers, and (C) a biomolecular realization of the same GRN using the parts described in Figure 3.2.1. In the GRN representation orange circles denote transition species, purple circles denote state species, and green circles denote sensor species. In the GRN and biomolecular realization, the gene network is in state i when species Ri is at a low level expression, and following transition $\delta(q,\sigma)$ when transition species Tσq is at a high level of expression.

3.2 Modeling Gene Regulatory Networks

Gene regulatory networks are specified by the tuple,

$$G = (G_V, G_U) \tag{3.2.1}$$

$$G_V = (V, E_r, E_a) \tag{3.2.2}$$

$$G_U = (U, I_r, I_a), \tag{3.2.3}$$

where G_V is a internal gene network graph, and G_U is a signal graph. In G_V, V is a finite set of gene products, $E_r \subseteq V \times V$ is a repression relation, and $E_a \subseteq V \times V$ is an activation relation. In G_U, U is a finite set of input signals, $I_r \subseteq U \times (V \cup E_r \cup E_a)$ is a signal repression relation, and $I_a \subseteq U \times (V \cup E_r \cup E_a)$ is a signal activation relation. Gene regulatory networks are typically represented as a directed graph, where V and U are sets of vertexes, E_r and E_a are directed repression and activation edges connecting nodes in V, and I_r and I_a are directed signal repression and signal activation edges connecting nodes in U to nodes or edges in G_V. For example, in Figure 3.1.1B, green, orange, and purple circles denote gene products, and START, a, and b denote input signals to the network. Repression edges that connect nodes denote repression relations between those gene products. For example, the edge connecting R0 to Ta0 indicates that R0 represses the production of Ta0. Some signal repression edges connect nodes to other edges, for example, a signal repression edge connects a to the edge between Sa and Ta0. This denotes a biomolecular reaction where signal a prevents Sa from repressing Ta0. These directed graphs provide a high level description of the relationship between genes and input signals in a regulatory network.

Remark. For simplicity of interpretation and implementation, I am mainly concerned with gene networks made of only repressing relations, and will specify these networks as

$$G = (V, E_r, U, I_r), \tag{3.2.4}$$

where E_a and I_a are empty and therefore not shown.

3.2.1 Biomolecular Parts

Specific biomolecular parts can be used to implement high level GRN specifications. Figure 3.2.1 illustrates a class of parts for transcription regulation and small molecule sensing that will be used throughout this paper. Our goal here is simply to show what a minimal set of parts can do. It will become apparent that the parts used could be replaced with similar parts, or that the designs I propose could be made more robust or efficient.

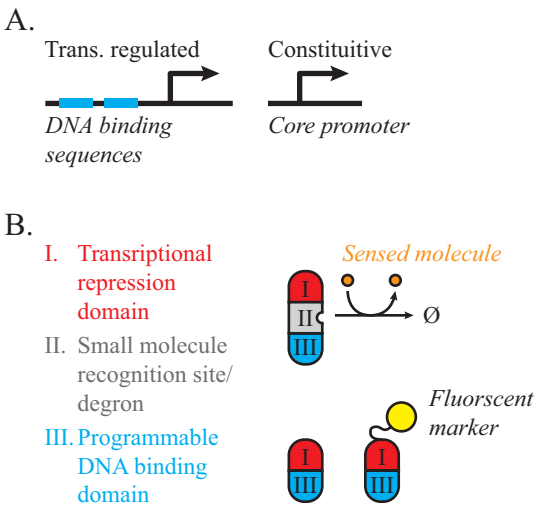


Figure 3.2.1: Biomolecular parts for realizing finite state machines: (A) Transcriptionally regulated and unregulated promoters. Transcriptionally regulated promoters have specific DNA sequences illustrated as blue bars that may be bound by domain III on a repressing transcription factors. Transcriptionally unregulated promoter is nominally "on". (B) Transcription factors are made of three primary components: a transcriptional repression domain; an optional signaling molecule binding site; and a programmable DNA binding domain. Sensed molecules are small diffusable signaling molecules that bind to recognition sites (i.e. a degron) in programmable transcription factors, and catalyze degradation of the transcription factor. Also illustrated is a fluorescently labeled transcription factor, which we use to distinguish "state" proteins from "transition" proteins.

I consider gene networks made of promoters, small diffusable signal molecules, and repressing transcription factors with programmable DNA binding domains. Figure 3.2.1a shows two varieties of promoters: *constitutive* and *transcriptionally regulated*. Constitutive pro-

moters are always "on", expressing their associated gene product at some nominal level. Transcriptionally regulated promoters contain one or more specific DNA binding sequences upstream of a core promoter. I consider transcriptionally regulated promoters that are nominally "on" unless bound by one or more repressing transcription factors.

Figure 3.2.1b shows several transcription factors made by fusing two or three functional domains: a transcriptional repression domain (I); a degron domain (II); a DNA binding domain (III). The construction of such transcription factors is now routine [52, 65, 66, 88, 89]. All transcription factors considered here contain domains I and III, and special signal sensing transcription factors contain domain II. Transcription factors might be fluorescently labeled, and for clarity, this labeling is used to distinguish "state" transcription factors from "transition" transcription factors. Small signal molecules (represented as small circles) bind to and catalyze the degradation of specific "sensor" transcription factors. Signals may be inducers or cell-cell signaling molecules that can be enzymatically produced, and exported by the cell and other cells or supplied exogenously. In this way, signals may be applied exogenously as input to a GRN or produced internally as an intercellular communication medium in multicellular systems.

In Figure 3.2.2 I depict biomolecular realizations and corresponding gene regulatory networks for the four types of components that are interconnected to construct finite state machines: the *transcriptionally unregulated gene*, the *singly regulated gene*, the *doubly regulated gene*, and the *small molecule sensor*. The first three components represent motifs for gene regulation via repressing transcription factors, while the fourth component will be used to sense a small input signal molecule. The biomolecular parts shown in Figure 3.2.1 and networks made from the composition of these parts can be modeled as Boolean networks, as illustrated in Figure 3.2.2. This is just one possible way to implement gene regulatory networks, and there are certainly others. For example, a similar implementation can be imagined with CRISPR transcription factors [60]. Now that I have defined the syntax of gene regulatory networks and a set of biomolecular parts to model, I will use Boolean network dynamics and delay differential equations to provide semantic interpretation as dynamical systems.

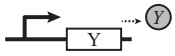
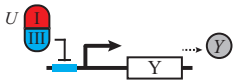
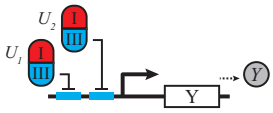
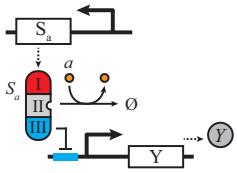
Component Type	Biomolecular Realization	GRN	Boolean Network Equations	Delay Differential Equations
Transcriptionally Unregulated Gene		Y	$Y^t = on$	$\frac{d}{dt}Y(t) = V_{max} - \beta Y(t)$
Singly Regulated Gene		U ⊣ Y	$Y^{t+1} = \neg U^t$	$\frac{d}{dt}Y(t) = \frac{V_{max}}{1+\left(\frac{U(t-\tau)}{k_{1/2}}\right)^n} - \beta Y(t)$
Doubly Regulated Gene		U_2 ⊣ Y, U_1 ⊣ Y	$Y^{t+1} = \neg(U_1^t \vee U_2^t)$	$\frac{d}{dt}Y(t) = \frac{V_{max}}{1+\left(\frac{U_1(t-\tau)+U_2(t-\tau)}{k_{1/2}}\right)^n} - \beta Y(t)$
Small Molecule Sensor		a ⊣ S_a, S_a ⊣ Y	$S_a^t = \neg a^t$ $Y^{t+1} = \neg S_a^t$	$\frac{d}{dt}S_a(t) = V_{max} - (\beta + k_p a(t))S_a(t)$ $\frac{d}{dt}Y(t) = \frac{V_{max}}{1+\left(\frac{S_a(t-\tau)}{k_{1/2}}\right)^n} - \beta Y(t)$

Figure 3.2.2: Components of a biomolecular realization of finite state machines, represented as a GRN, Boolean network equations, and delay differential equations. In the delay differential equation representation, τ denotes the delay associated with transcription and translation, β is the rate of dilution associated with cell growth, and V_{max} is the maximum rate of production of a gene product. The delay differential equations follow a Michaeles-Menten form where $k_{1/2}$ and n are the Michaeles constant and Hill coefficient respectively.

3.2.2 Boolean Network Model

The *Boolean network* model of gene regulation is a discrete time dynamical system used to model course-grained dynamics of gene expression [90, 91]. Boolean networks were first introduced in this application to investigate the dynamics of randomly generated gene regulatory networks, and have been used successfully as a descriptive tool, and more recently (and perhaps surprisingly) as a predictive model of gene regulatory network dynamics in natural and synthetic systems [92–96]. In a Boolean network, the expression level of each gene product or input to the network is in one of two states: *on* and *off*, (or *true* and *false*) denoting high or low expression level respectively. For brevity, define the domain $\mathbb{B} \triangleq \{on, off\}$. Boolean states are updated at discrete times, and the expression level of a gene product at time $t+1$ is a Boolean function of the expression level of gene products that affect it at time t. In general, for gene products $Y_1, Y_2, \ldots, Y_n$ and inputs $U_1, U_2, \ldots, U_m$, I denote the state of the gene products and inputs at time t as $Y_1^t, Y_2^t, \ldots, Y_n^t \in \mathbb{B}$ and $U_1^t, U_2^t, \ldots, U_m^t \in \mathbb{B}$ respectively. The dynamics of this network can then be written,

$$Y_1^{t+1} = f_1(Y_1^t, Y_2^t, \ldots, Y_n^t, U_1^t, U_2^t, \ldots, U_m^t) \tag{3.2.5}$$

$$Y_2^{t+1} = f_2(Y_1^t, Y_2^t, \ldots, Y_n^t, U_1^t, U_2^t, \ldots, U_m^t) \tag{3.2.6}$$

$$\vdots \tag{3.2.7}$$

$$Y_n^{t+1} = f_n(Y_1^t, Y_2^t, \ldots, Y_n^t, U_1^t, U_2^t, \ldots, U_m^t), \tag{3.2.8}$$

where $f_i : \mathbb{B}^{n+m} \to \mathbb{B}$ is some Boolean update function for $i = 1, 2, \ldots, n$.

Remark. For simplicity, I use the same symbol for the name of the gene product and the time-varying state of the gene product. Additionally, I name genes without subscripts, and gene products and time-varying state with subscripts. In general, the meaning of these symbols should be clear from context.

In the Boolean network model, the update function f_i typically corresponds to a mechanistic model of transcriptional regulation. In the networks considered in this text, all transcription factors are repressing, and all promoters are nominally "on" unless bound by some repressing

transcription factor. This means the transcriptionally unregulated gene motif in Figure 3.2.2 always expresses its gene product. The Boolean network dynamics for the transcriptionally unregulated gene encodes the sequential logic for *true*,

$$Y^{t+1} = on. \tag{3.2.9}$$

In the singly regulated gene shown in Figure 3.2.2, the input U is a repressing transcription factor with a DNA binding domain that binds to and represses transcription of gene Y, preventing expression of gene product Y. The Boolean network dynamics for the singly regulated gene encodes the sequential logic for NOT,

$$Y^{t+1} = \neg U^t. \tag{3.2.10}$$

In other words, gene product Y is *on* unless the input U was present at the previous time step. Similarly, the doubly regulated gene in Figure 3.2.2, has two inputs U_1 and U_2 that may each bind to and repress transcription of gene Y. The Boolean network dynamics for the doubly regulated gene encodes the sequential logic for NOR,

$$Y^{t+1} = \neg \left(U_1^t \vee U_2^t \right), \tag{3.2.11}$$

meaning the output Y is *on* unless input U_1 or input U_2 was present at the previous time step. Finally, the small molecule sensor in Figure 3.2.2 is a two-gene component that makes use of two additional parts from Figure 3.2.1: a signal molecule a, and a transcription factor S_a that contains the degron domain (II) that is sensitive to a. The transcription factor S_a is transcriptionally unregulated, and in the absence of signal a, S_a binds to the upstream binding sequence of gene Y and prevents expression of gene product Y (same as the singly regulated gene). However, in the presence of a, the signal molecule quickly binds to S_a and catalyzes its degradation through fast protein-protein interactions. The resulting absence of S_a allows gene product Y to be expressed. This interaction is denoted in the GRN in Figure 3.2.2 by a repression arrow pointing from a to the repression arrow connecting S_a to Y. Here, a^t is said to be an input to the network. Since the interaction between S_a and

a is much faster than gene expression, the state S_a^t can be approximated as a function of a^t,

$$S_a^t = \neg a^t. \tag{3.2.12}$$

The Boolean network dynamics of the small molecule sensor component are then,

$$Y^{t+1} = \neg S^t \tag{3.2.13}$$

$$= a^t. \tag{3.2.14}$$

The advantage of a Boolean network model is that it allows for the complete enumeration of the state space, which is used later to show that for any FSM a Boolean network can be constructed from the components in Figure 3.2.2 such that the dynamics of the Boolean network are isomorphic to the transition function of the FSM. The disadvantage of such a model is that it may make unrealistic simplifying assumptions, and lacks the fidelity of a continuous system such as a network modeled by Hill functions or chemical reaction networks. For this reason, I also consider a delay differential equation model of gene expression.

3.2.3 Delay Differential Equation Model

There are some questions that cannot be addressed assuming the Boolean network model. Gene expression levels are in general not binary, and depend on factors such as binding affinities of DNA binding domains, and dilution as a result of cell growth and division. Ultimately any model of a gene regulatory network depends on the specific biomolecular components used to implement the system. However it is useful to consider a simple continuous model to examine how well a motif for implementing finite state machines approximates the discrete Boolean network model under various kinetic parameters. To this end, DDEs and Hill equations [93, 97] are used to model the dynamics of gene product concentrations in the components outlined in Figure 3.2.2. Notably, I will make the following simplifying assumptions about the gene network: all regulated and unregulated genes are built around

the same core promoter, all gene products have the same nominal rates of translation and transcription, all proteins have approximately the same rate of nominal dilution and degradation, and all transcription factors have similar binding affinities and cooperativity. These are reasonable design assumptions that greatly reduce the number of parameters in the model.

The DDE for the transcriptionally unregulated gene in Figure 3.2.2 is,

$$\frac{d}{dt}Y(t) = V_{max} - \beta Y(t), \tag{3.2.15}$$

where $Y(t) \in \mathbb{R}$ is the time-varying concentration of gene product Y, and $V_{max} \in \mathbb{R}$ and $\beta \in \mathbb{R}$ are tunable parameters to the model. V_{max} is the rate of transcription and translation of gene Y, and β denotes the cumulative rate dilution due to cell growth and nominal protein degradation. In steady state,

$$\lim_{t\to\infty} Y(t) = \frac{V_{max}}{\beta}. \tag{3.2.16}$$

The DDE for describing the dynamics of the singly regulated gene component in Figure 3.2.2 is,

$$\frac{d}{dt}Y(t) = \frac{V_{max}}{1 + \left(\frac{U(t-\tau)}{k_{1/2}}\right)^n} - \beta Y(t). \tag{3.2.17}$$

The first term in this DDE takes the form of a Hill function. As before, V_{max} is the maximum unregulated rate of transcription, and β is a cumulative diffusion and nominal protein degradation rate. In addition, $U(t) \in \mathbb{R}$ is the time-varying concentration of repressing transcription factor U, and $k_{1/2} \in \mathbb{R}$, $n \in \mathbb{R}$ and $\tau \in \mathbb{R}$ are three new tunable parameters. $k_{1/2}$ is the concentration of $U(t)$ required to half the rate of transcription of gene Y, and n is the Hill coefficient for repression. τ denotes the time delay from transcription of a gene to translation of the gene product. Similarly, the DDE for the doubly regulated gene

component is,

$$\frac{d}{dt}Y(t) = \frac{V_{max}}{1 + \left(\frac{U_1(t-\tau)+U_2(t-\tau)}{k_{1/2}}\right)^n} - \beta Y(t). \tag{3.2.18}$$

This is the same as Eq. (3.2.17) for the singly regulated gene, except that here the time-delayed concentration of both repressing transcription factors U_1 and U_2 are summed in the Hill function. This assumes that both U_1 and U_2 bind to the same DNA binding sequence, and have identical effects on the transcription of gene Y. Finally, two equations model the dynamics of the small molecule sensor in Figure 3.2.2,

$$\frac{d}{dt}S_a(t) = V_{max} - (\beta + k_p a(t))\, S_a(t) \tag{3.2.19}$$

$$\frac{d}{dt}Y(t) = \frac{V_{max}}{1 + \left(\frac{S_a(t-\tau)}{k_{1/2}}\right)^n} - \beta Y(t). \tag{3.2.20}$$

Eqs. (3.2.15) and (3.2.19) captures the dynamics of production and degradation of sensor molecule S_a via signal molecule a. As before, V_{max} is the maximum rate of production of S_a and β is the cumulative rate of diffusion and nominal degradation. Additionally, S_a is degraded through fast protein-protein interactions catalyzed by signal a. The kinetics of these reactions are lumped into a single rate parameter k_p. Eq. (3.2.20) then models the production of output molecule Y regulated by repressing transcription factor S_a just as with the singly regulated gene in Eq. (3.2.17).

3.3 General Construction of a GRN from an FSM Specification

To implement an FSM with a GRN, I define states and input symbols in terms of the parts and components illustrated in Figures 3.2.1 and 3.2.2. A natural choice is to represent a state in the FSM with a particular gene expression level or set of gene expression levels, and represent input symbols with the presence or absence of signal molecules that alter the gene expression level. A sequence of input symbols could then be encoded as a trajectory over signal concentrations. The GRN implementation of a FSM then acts as a sequence recognizer, and the notion of an accepting state encodes the logic for recognizing a particular

set of trajectories over input signals. Specifically, beginning with the FSM,

$$M = (Q, \Sigma, \delta, q_0, F), \tag{3.3.1}$$

the gene regulatory network

$$(V, E_r, U, I_r) = g(M), \tag{3.3.2}$$

is generated where V is the set of gene products in the network, E_r describes how the proteins in V are "wired" together as a GRN, U is the set of input signals to the network, and I_r describes how the signal molecules affect transcription of the genes in V.

Inputs to the network are a set of signal molecules Σ and an inducible repressing transcription factor START,

$$U = \Sigma \cup \{\text{START}\}. \tag{3.3.3}$$

Every gene in the network encodes a repressing transcription factor, and these can be broken into three categories: *state genes*, *transition genes*, and *sensor genes*. For clarity, I name all the state genes "Rq" where q is the name of state in the FSM. The state genes are defined as a set of singly repressed genes,

$$V_R = \{\text{R}q \mid \forall q \in Q\}. \tag{3.3.4}$$

Similarly, I prefix the names of all transitions genes with a "T". The transition genes are a set of doubly repressed genes,

$$V_T = \{\text{T}\sigma q \mid \forall q \in Q, \sigma \in \Sigma \text{ s.t. } \exists \delta(q, \sigma)\}. \tag{3.3.5}$$

For $q, q' \in Q$ and $\sigma \in \Sigma$, I consider the GRN to be following transition $\delta(q, \sigma) \mapsto q'$ when Tσq is at a *high* level of expression and all other transition genes are at a *low* level of expression.

Finally, all the sensor gene names are prefixed with a "S". Sensor genes are a set of transcriptionally unregulated genes,

$$V_S = \{\mathrm{S}\sigma \mid \forall \sigma \in \Sigma\}, \tag{3.3.6}$$

where each signal molecule σ binds uniquely to the sensor transcription factor $S\sigma$. For $\sigma \in \Sigma$, I consider the symbol σ to be applied to the machine when the signal molecule σ is present.

The repression edges E_r are then made up of edges between genes. Specifically, let $E_{R,T}$ be the set of repression edges from state genes to transition genes. For each state $q \in Q$, state gene $\mathrm{R}q$ represses transition genes $\mathrm{T}\sigma q$ for all $\sigma \in \Sigma$.

$$E_{R,T} = \{(\mathrm{R}q, \mathrm{T}\sigma q) \mid \forall \mathrm{R}q \in \mathrm{V}_R, \mathrm{T}\sigma q \in \mathrm{V}_T\}. \tag{3.3.7}$$

Let $E_{T,R}$ be the set of repression edges from transition genes to state genes. To encode the transition function, for all $q, q' \in Q$ and $\sigma \in \Sigma$ where $\delta(q, \sigma) \mapsto q'$, transition gene $\mathrm{T}\sigma q$ should repress state gene $\mathrm{R}q'$.

$$E_{T,R} = \{(\mathrm{T}\sigma q, \mathrm{R}q) \mid \forall \mathrm{T}\sigma q \in \mathrm{V}_T, \mathrm{R}q \in \mathrm{V}_R\}. \tag{3.3.8}$$

Let $E_{S,T}$ be the set of repression edges from sensor genes to transition genes. For each input symbol $\sigma \in \Sigma$, sensor gene $\mathrm{S}\sigma$ represses transition genes $\mathrm{T}\sigma q$ for all $q \in Q$.

$$E_{S,T} = \{(\mathrm{S}\sigma, \mathrm{T}\sigma q) \mid \forall \mathrm{S}\sigma \in \mathrm{V}_S, \mathrm{T}\sigma q \in \mathrm{V}_T\}. \tag{3.3.9}$$

Let $I_{\Sigma,S}$ be the set of repression edges from signal molecules in Σ to the edges in $E_{S,T}$,

$$I_{\Sigma,S} = \{(\sigma,\ (\mathrm{S}\sigma, T\sigma q)) \mid \forall \sigma \in \Sigma, (\mathrm{S}\sigma, T\sigma q) \in E_{S,T}\}. \tag{3.3.10}$$

Finally, encoding the start state q_0, START should repress start gene Rq_0. Then for $(V, E_r, U, I_r) = g(M)$,

$$V = V_R \cup V_T \cup V_S \tag{3.3.11}$$

$$E_r = E_{R,T} \cup E_{T,R} \cup E_{S,T} \tag{3.3.12}$$

$$U = \Sigma \cup \{\text{START}\} \tag{3.3.13}$$

$$I_r = I_{\Sigma,S} \cup \{(\text{START}, \text{R}q_0)\}. \tag{3.3.14}$$

3.3.1 Boolean Network Model of the General Construction

In general, the Boolean network equations for a sensor gene Sσ and a transition gene Tσq are,

$$S_\sigma^t = \neg\sigma^t \tag{3.3.15}$$

$$T_{\sigma,q}^{t+1} = \neg\left(R_q^t \vee S_\sigma^t\right), \tag{3.3.16}$$

where S_σ^t, $T_{\sigma,q}^t$, and R_q^t denote the Boolean value of the sensor gene, transition gene, and state gene respectively at time t. There are two forms of the equation describing the dynamics of each state gene Rq. When $q \neq q_0$,

$$R_q^{t+1} = \neg \bigvee_{\{(q',\sigma)|\delta(q',\sigma)\mapsto q\}} T_{\sigma,q'}^t \tag{3.3.17}$$

and when $q = q_0$,

$$R_q^{t+1} = \neg\left(\left(\bigvee_{\{(q',\sigma)|\delta(q',\sigma)\mapsto q\}} T_{\sigma,q'}^t\right) \vee \text{START}^t\right). \tag{3.3.18}$$

In the absence of any inputs $\sigma \in \Sigma$ and a low expression of START, the network reaches steady-state in two time steps. The expression level at steady state is,

$$S_\sigma^t = on \tag{3.3.19}$$

$$T_{\sigma,q}^t = off \tag{3.3.20}$$

$$R_q^t = on, \tag{3.3.21}$$

for all $\sigma \in \Sigma$ and $q \in Q$. At steady state, the network is not in any state of the FSM, nor is it following any transitions.

In the Boolean network framework, an input sequence to the FSM is encoded as a trajectory over signal molecules and the START gene activity. Let the set of input symbols $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, and let $w = \sigma_{c_1}\sigma_{c_2}\ldots\sigma_{c_m} \in \Sigma^*$ be an input string to the FSM M where the index $c_i \in \{1, \ldots, n\}$ for $i = 1 \ldots m$. In the Boolean network model of the GRN $g(M)$, the sequence w specifies an input trajectory $u^t = h_{BN}(w, t)$ for $t \in \mathbb{N}$, defined as

$$h_{BN}(w,t) = \begin{bmatrix} START^t \\ \sigma_1^t \\ \sigma_2^t \\ \vdots \\ \sigma_n^t \end{bmatrix} \tag{3.3.22}$$

where,

$$START^t = \begin{cases} on, & t \in \{0, 1\} \\ off, & \text{otherwise,} \end{cases} \tag{3.3.23}$$

and for each σ_j^t with $j \in \{1, 2, \ldots, n\}$,

$$\sigma_j^t = \begin{cases} on, & \exists c_i \text{ s.t } j = c_i \text{ and } t \in \{2i, 2i+1\} \\ off, & \text{otherwise.} \end{cases} \tag{3.3.24}$$

Exactly one signal or the START gene is active at any particular time. The START gene is *on* for two time steps (i.e. from $t = 0$ through $t = 1$). Immediately following the START pulse, each signal σ_{c_i} (for $i = 1, 2, \ldots, m$) is *on* in sequence for two time steps. An example trajectory is shown in Figure 3.3.1, and discussed later in detail.

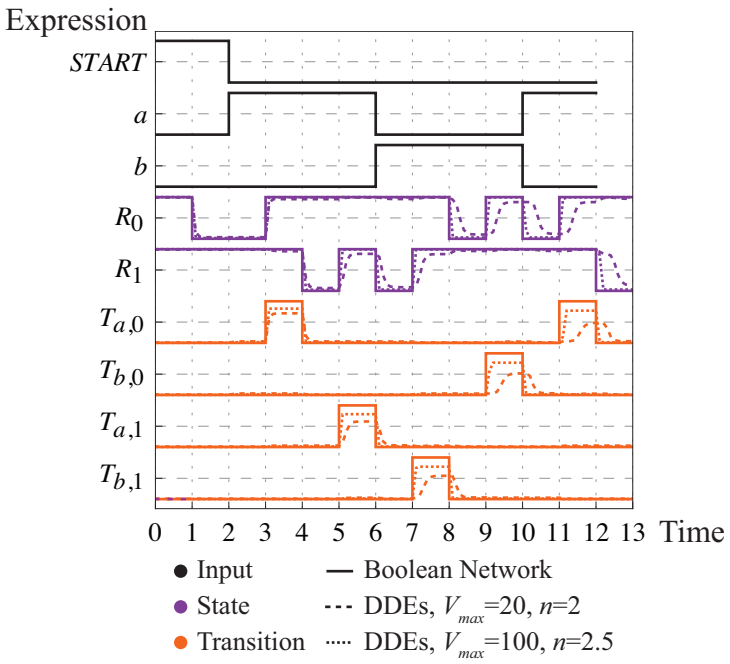


Figure 3.3.1: Boolean network and DDE trajectories for the two-state FSM from Figure 3.1.1. In this simulation, $\tau = 1$, $n = 2$, $V_{max} = \beta = 20$, $k_{1/2} = 0.2$, k_p=200. Solid lines denote Boolean network trajectories and dotted lines denote DDE trajectories. The top three plots show the input trajectories of a, b, and START that encode the sequence "aabba". The control input for the Boolean network and DDE model are identical. The middle two plots show to the expression level of state genes R0 and R1 where low expression of Ri corresponds to being in state i of the FSM. The bottom four plots illustrate the expression level of transition genes Ta0, Tb0, Ta1, and Tb1, where high expression of Tσq denotes the transition $\delta(q, \sigma)$ in the FSM.

3.3.2 DDE Model of the General Construction

Similarly, the dynamics of a biomolecular realization of sensor and transition genes can be written as a system of DDEs. Where $\sigma(t)$ is the concentration of signal molecule σ at time t, and $S_\sigma(t)$, $T_{\sigma,q}(t)$, $R_q(t)$, and $START(t)$ are the concentrations of sensor, transition, state, and "start" transcription factors respectively,

$$\frac{d}{dt}S_\sigma(t) = V_{max} - (\beta + k_p\sigma(t))\, S_\sigma(t) \tag{3.3.25}$$

$$\frac{d}{dt}T_{\sigma,q}(t) = \frac{V_{max}}{1 + \left(\frac{S_\sigma(t-\tau)+R_q(t-\tau)}{k_{1/2}}\right)^n} - \beta T_{\sigma,q}(t), \tag{3.3.26}$$

for each $\sigma \in \Sigma$ and $q \in Q$. As with the Boolean network representation, there are two forms of the DDE describing the dynamics of a state gene. When $q \neq q_0$,

$$\frac{d}{dt}R_q(t) = \frac{V_{max}}{1 + \left(\frac{START(t-\tau)+\sum_{(q',\sigma)|\delta(q',\sigma)\mapsto q} T_{\sigma,q'}(t-\tau)}{k_{1/2}}\right)^n} - \beta R_q(t). \tag{3.3.27}$$

When $q = q_0$,

$$\frac{d}{dt}R_q(t) = \frac{V_{max}}{1 + \left(\frac{\sum_{(q',\sigma)|\delta(q',\sigma)\mapsto q} T_{\sigma,q'}(t-\tau)}{k_{1/2}}\right)^n} - \beta R_q(t). \tag{3.3.28}$$

In general, the steady-state of this system of DDEs can be solved for numerically given the control inputs $S_\sigma(t) = 0\ \forall\sigma \in \Sigma$ and $START(t) = 0$. The steady state of the DDEs qualitatively reflect the steady state of the Boolean network for a large range of parameters.

As with the Boolean network model, an input sequence to the FSM is encoded as an input trajectory $u(t) = h_{DDE}(w, \Delta t, t)$ for $t \in \mathbb{R}$ and *pulse width* $\Delta t \in \mathbb{R}$, defined as,

$$h_{DDE}(w,t) = \begin{bmatrix} START(t/\Delta t) \\ \sigma_1(t/\Delta t) \\ \sigma_2(t/\Delta t) \\ \vdots \\ \sigma_n(t/\Delta t) \end{bmatrix} \tag{3.3.29}$$

(3.3.30)

where,

$$START(t) = \begin{cases} 1, & t \in [0,1) \\ 0, & \text{otherwise} \end{cases} \tag{3.3.31}$$

(3.3.32)

and for each σ_j with $j \in \{1, 2, \ldots, n\}$

$$\sigma_j(t) = \begin{cases} 1, & \exists c_i \text{ s.t. } j = c_i \text{ and } t \in [2i, 2i+1) \\ 0, & \text{otherwise.} \end{cases} \tag{3.3.33}$$

As with the Boolean network model, exactly one signal or the START gene is active at any given time. Here, Δt controls for the input signal pulse width, typically taken to be $\Delta t \geq \tau$. In 4, the pulse width $\Delta t = \tau$, making the sample input trajectory for the DDE model match the input to the Boolean network.

3.3.3 Example: Two-state Machine as a Boolean Network

As an example, Figure 3.1.1A depicts a simple two-state FSM represented as a directed graph, along with a GRN implementation and a biomolecular realization of the same ma-

chine. This machine has two states, $Q = \{0, 1\}$. The FSM begins in state $q_0 = 0$, and from here two transitions a possible. If the next input symbol is an "a", then the machine moves to state 1; however, if the next input symbol is a "b", then the machine stays in state 0. Conversely, from state 1, an "a" leaves the machine in state 1, while a "b" moves the machine back to state 0. Since the set of accepting states F consists only of state 1, this FSM accepts all sequences of "a" and "b" that end in an "a".

In the GRN implementation in Figure 3.1.1B, the state 0 is represented by the low expression of gene R0, and the state 1 is represented by the low expression of gene R1. Modeled as a Boolean network, the equations describing the dynamics of this GRN are,

$$R_0^{t+1} = \neg \left(T_{b,0}^t \vee T_{b,1}^t \vee \mathrm{START}^t \right) \tag{3.3.34}$$

$$R_1^{t+1} = \neg \left(T_{a,0}^t \vee T_{a,1}^t \right) \tag{3.3.35}$$

$$T_{a,0}^{t+1} = a^t \wedge \neg R_0^t \tag{3.3.36}$$

$$T_{b,0}^{t+1} = b^t \wedge \neg R_0^t \tag{3.3.37}$$

$$T_{a,1}^{t+1} = a^t \wedge \neg R_1^t \tag{3.3.38}$$

$$T_{b,1}^{t+1} = b^t \wedge \neg R_1^t. \tag{3.3.39}$$

Figure 3.3.1 shows the Boolean network and DDE trajectories for this GRN given the input sequence "aabba". In the Boolean network, the initial high expression of START at time $t = 0$ through $t = 1$ results in a low expression of R0 during time $t = 1$ through $t = 2$, denoting the state $q_0 = 0$ in the FSM. Since every transition gene Tσq is repressed by both state gene Rq and sensor gene Sσ, low expression of Rq at time t means that the expression level of Tσq at time $t + 1$ is sensitive to signal σ at time t. Specifically, from Eqs. (3.3.34–3.3.39), when R0 is *off*,

$$T_{a,0}^{t+1} = a^t \tag{3.3.40}$$

$$T_{b,0}^{t+1} = b^t. \tag{3.3.41}$$

When the signal a is present at time $t = 2$, repression is temporarily relieved on transition gene Ta0 at time $t = 3$. The high expression of Ta0 indicates transition $\delta(0, a)$ in the FSM. In general, the presence of an input signal coinciding with the repression of a state gene determines which transition gene will be active at the following time step. Transition gene Ta0 represses state gene R1 (state 1 in the FSM) at time $t = 4$, leaving transition genes Ta1 and Tb1 sensitive to signal molecule a. Since signal a is still active at time $t = 4$, Ta1 is expressed at time $t = 5$ and R1 is again repressed at time $t = 6$ (in the FSM, $\delta(1, a) \mapsto 1$). This process is repeated for the subsequent long pulse of signal b and short pulse of signal a (encode two "b" input symbols followed by an "a"). After the final pulse of signal a is supplied, the GRN arrives in the final accepting state, and R1 is repressed at time $t = 12$. In this example the transition function δ was defined for all combinations of input symbols and states; however, as will be shown in Section 3.4, if the simulation were to continue absent of input signals, or if an input symbol was provided that did not correspond to a valid transition, the Boolean network model of this GRN would return to its initial steady state in two time steps.

3.3.4 Example: Two-state Machine as a System of DDEs

As mentioned previously, gene expression levels are, in general, not binary. In order to assess continuous effects such as production, dilution, and degradation rates, and binding affinities, the GRN can be modeled as a system of delay differential equations. The equations describing the continuous dynamics of the biomolecular realization in Figure 3.1.1C

are,

$$\frac{d}{dt}S_a(t) = V_{max} - (\beta + k_p a(t))\, S_a(t) \tag{3.3.42}$$

$$\frac{d}{dt}S_b(t) = V_{max} - (\beta + k_p b(t))\, S_b(t) \tag{3.3.43}$$

$$\frac{d}{dt}T_{a,0}(t) = \frac{V_{max}}{1 + \left(\frac{S_a(t-\tau)+R_0(t-\tau)}{k_{1/2}}\right)} - \beta T_{a,0}(t) \tag{3.3.44}$$

$$\frac{d}{dt}T_{b,0}(t) = \frac{V_{max}}{1 + \left(\frac{S_b(t-\tau)+R_0(t-\tau)}{k_{1/2}}\right)} - \beta T_{b,0}(t) \tag{3.3.45}$$

$$\frac{d}{dt}T_{a,1}(t) = \frac{V_{max}}{1 + \left(\frac{S_a(t-\tau)+R_1(t-\tau)}{k_{1/2}}\right)} - \beta T_{a,1}(t) \tag{3.3.46}$$

$$\frac{d}{dt}T_{b,1}(t) = \frac{V_{max}}{1 + \left(\frac{S_b(t-\tau)+R_1(t-\tau)}{k_{1/2}}\right)} - \beta T_{b,1}(t). \tag{3.3.47}$$

Here the same values for the time delay τ, rate parameters V_{max}, β, and k_p, and Hill parameters $k_{1/2}$ and n, are used throughout the system. This is a sensible choice assuming all genes use the same core promoter, and all transcription factors use the same repression domain and carefully tuned DNA binding domains. A sample trajectory of this system is shown in Figure 3.3.1. Choosing $\tau = 1$ allows direct comparison to the Boolean network trajectory with the same control input. In an experimental system, the stepwise constant input could be approximated by washing media across cells in a microfluidic device. For the other parameters, $k_{1/2} = 0.2$, $k_p = 200$, and V_{max}, β, and n were varied.

In Figure 3.3.1 the DDE simulations qualitatively track the trajectories of the Boolean network model. Initially both DDE simulations track quite well. However by time $t = 6$ the behavior of the DDE model for $n = 2$ and $V_{max} = \beta = 20$ appears to lag compared to the ideal Boolean network model, and by time $t = 12$, the DDE model appears to be lagging by nearly a half time unit. Additionally, the maximum amplitude of the gene expression levels decreases over time, and this is particularly evident with the transition genes. By comparison, the DDE model for $V_{max} = \beta = 100$ and $n = 2.5$ tracks the Boolean network model without significant lag or change in maximum amplitude of expression. There are many methods for improving the behavior of the DDE model. One possible solution is

to re-engineer the finite state machine. Other FSMs may accept the same language, and result in a different GRN that is more robust to the same input. In general, one would like to know, given an FSM and GRN implementation of that FSM, how robust is the GRN implementation to changes in the parameters V_{max}, β, n, $k_{1/2}$, and τ.

3.4 GRNs are Computationally Equivalent to FSMs

In the prequel, assuming a Boolean network model, the set of input sequences to the two-state FSM are exactly the set of input sequences that end in an accepting state in the GRN realization of the two-state FSM. In computer science, given two models of computation, if behavior of any machine constructed in one model can be recapitulated with a machine constructed in the other model, we say that one model is *simulated* by the other. In fact, given any FSM the general construction can be used to design a GRN that recapitulates the behavior of the FSM. In other words, assuming a Boolean network model, GRNs simulate FSMs. Additionally, because any Boolean network can be simulated by an FSM, *any* gene regulatory network modeled as a Boolean network is no more or less capable than a finite state machine—assuming a Boolean network model, GRNs are computationally equivalent to FSMs.

Theorem 3.4.1. *Given a finite state machine $M = (Q, \Sigma, \delta, q_0, F)$, the gene regulatory network $(V, E_r, U, I_r) = g(M)$ simulates M when modeled as a Boolean network.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an FSM, and let $g(M)$ be the general construction of a GRN based on M. Let $Q = \{1, 2, \ldots, n\}$ and $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, then the sets of state genes and transition genes in $g(M)$ are $\{\mathrm{R}1, \mathrm{R}2, \ldots, \mathrm{R}n\}$ and $\{\mathrm{T}\sigma_1 1, \ldots, \mathrm{T}\sigma_m n\}$. Let $Y \in \mathbb{B}^{n(m+1)}$ be a *state vector* and $U \in \mathbb{B}^m$ be an *input vector* to the Boolean

network model of $g(M)$, where

$$Y = \begin{bmatrix} R_1 \\ \vdots \\ R_n \\ T_{\sigma_1,1} \\ \vdots \\ T_{\sigma_m,n} \end{bmatrix} \tag{3.4.1}$$

$$U = \begin{bmatrix} START \\ \sigma_1 \\ \vdots \\ \sigma_m \end{bmatrix}. \tag{3.4.2}$$

Note that in the Boolean network model of $g(M)$, the state of the sensor gene $\mathrm{S}\sigma_i$ at time t depends only on the state of the corresponding input signal σ_i at time t (Eq. (3.3.15)). The update function for the Boolean network (Eqs. (3.2.5–3.2.8)) can be written,

$$Y^{t+1} = f(Y^t, U^t). \tag{3.4.3}$$

Let $\mathcal{R}_Q$ and $\mathcal{R}_\Sigma$ be equivalence relations between states $q \in Q$ and states in the Boolean network, and between input symbols $\sigma \in \Sigma$ and control inputs to the Boolean network respectively,

$$\mathcal{R}_Q \in Q \times \mathbb{B}^{n(m+1)} \tag{3.4.4}$$

$$\mathcal{R}_\Sigma \in \Sigma \times \mathbb{B}^{m+1}. \tag{3.4.5}$$

Additionally, let $F_{BN} \subseteq \mathbb{B}^{n(m+1)}$ be the set of accepting states F projected into $\mathbb{B}^{n+nm}$ by $\mathcal{R}_Q$,

$$F_{BN} = \{Y \in \mathbb{B}^{n(m+1)} \mid \exists q \in F \text{ s.t. } q \sim_{\mathcal{R}_Q} Y\}. \tag{3.4.6}$$

The Boolean network model of $g(M)$ *simulates* M, if and only if there exists equivalence relations $\mathcal{R}_Q$ and $\mathcal{R}_\Sigma$ such that a sequence $w \in \Sigma^*$ is accepted by the FSM if and only if the sequence of control inputs related to w by $\mathcal{R}_\Sigma$ allows the Boolean network to reach a state in the set F_{BN}.

Let $\phi : Q \to \mathbb{B}^{n(m+1)}$ map from an FSM state to a Boolean network state vector, where for $q \in Q$, $\phi(q) = Y$ and,

$$R_i = \begin{cases} on, & i \neq q \\ off, & \text{otherwise} \end{cases} \tag{3.4.7}$$

$$T_{\sigma_j i} = off. \tag{3.4.8}$$

Let $\psi : \Sigma \to \mathbb{B}^{m+1}$ map from FSM input symbols to Boolean network input vectors, where for $\sigma_j \in \Sigma$, $\psi(\sigma_j) = U$ and,

$$START = off \tag{3.4.9}$$

$$\sigma_i = \begin{cases} on, & i = j \\ off, & \text{otherwise.} \end{cases} \tag{3.4.10}$$

Note that ϕ and ψ are injective functions that define equivalence relations on FSM and Boolean network states and inputs,

$$\mathcal{R}_Q = \{(q, \phi(q)) \mid \forall q \in Q\} \tag{3.4.11}$$

$$\mathcal{R}_\Sigma = \{(\sigma, \phi(\sigma)) \mid \forall \sigma \in \Sigma\}. \tag{3.4.12}$$

Finally, let $\gamma : \mathbb{B}^{n(m+1)} \times \mathbb{B}^{m+1} \to \mathbb{B}^{n(m+1)}$ be the *two-step update function* where for state vector $Y \in \mathbb{B}^{n(m+1)}$ and input vector $U \in \mathbb{B}^{m+1}$,

$$\gamma(Y, U) = f(f(Y, U), U). \tag{3.4.13}$$

Here, γ computes the effect of applying control input U to the Boolean network model of $g(M)$ for two time steps, beginning in state Y.

Suppose that $q, q' \in Q$ and $\sigma \in \Sigma$ such that $\delta(q, \sigma) \mapsto q'$, and let $Y^t = \phi(q) \in \mathbb{B}^{n(m+1)}$ and $U^t = U^{t+1} = \psi(\sigma) \in \mathbb{B}^{m+1}$ with,

$$Y^{t+1} = f(Y^t, U^t) \tag{3.4.14}$$

$$Y^{t+2} = f(Y^{t+1}, U^{t+1}) \tag{3.4.15}$$

$$= \gamma(Y^t, \psi(\sigma)). \tag{3.4.16}$$

At time t, state gene Rq is at a low level of expression, and all other state genes are at a high level of expression. All transition genes Tσi for all $i \in Q$ are at a low level of expression. Signal molecule σ is at a high level of expression at time t, and the START gene is at a low level of expression. Solving for Eqs. (37) to (39), only the expression levels of the state gene Rq, and the transition gene Tσq change at time $t + 1$,

$$R_q^{t+1} = \neg \bigvee_{\{(p,s)|\delta(p,s)\to q\}} T_{s,p}^t \tag{3.4.17}$$

$$= on \tag{3.4.18}$$

$$T_{\sigma,q}^{t+1} = \sigma^t \wedge \neg R_q^t \tag{3.4.19}$$

$$= on. \tag{3.4.20}$$

At the following time step, only the expression levels of the state gene Rq' and transition gene Tσq change,

$$R_{q'}^{t+2} = \neg \bigvee_{\{(p,s)|\delta(p,s)\to q\}} T_{s,p}^{t+1} \tag{3.4.21}$$

$$= \mathit{off} \tag{3.4.22}$$

$$T_{\sigma,q}^{t+2} = \sigma^t \wedge \neg R_q^{t+1} \tag{3.4.23}$$

$$= \mathit{off}. \tag{3.4.24}$$

The state vector of the Boolean network at time $t+2$ is then,

$$Y^{t+2} = \gamma(\phi(q), \psi(\sigma)) \tag{3.4.25}$$

$$= \phi(q'). \tag{3.4.26}$$

Now suppose that $q \in Q$ and $\sigma \in \Sigma$ such that $\delta(q,\sigma)$ does not map to anything. Again, let $Y^t = \phi(q)$ and $U^t = U^{t+1} = \psi(\sigma)$. As before, solving for Eqs. (37) to (39), only the expression levels of the state gene Rq, and the transition gene Tσq change from time t to $t+1$,

$$Y^{t+1} = f(Y^t, U^t) \tag{3.4.27}$$

$$R_q^{t+1} = \neg \bigvee_{\{(p,s)|\delta(p,s)\to q\}} T_{s,p}^{t} \tag{3.4.28}$$

$$= \mathit{on} \tag{3.4.29}$$

$$T_{\sigma,q}^{t+1} = \sigma^t \wedge \neg R_q^t \tag{3.4.30}$$

$$= \mathit{on}. \tag{3.4.31}$$

However, unlike the previous example, at the following time step, only the expression level of the transition gene Tσq changes,

$$Y^{t+2} = f(Y^{t+1}, U^{t+1}) \tag{3.4.32}$$

$$= \gamma(\phi(q), \psi(\sigma)) \tag{3.4.33}$$

$$R_i^{t+2} = \neg \bigvee_{\{(p,s)|\delta(p,s)\to q\}} T_{s,p}^{t+1} \tag{3.4.34}$$

$$= on \tag{3.4.35}$$

$$T_{\sigma,q}^{t+2} = \sigma^t \wedge \neg R_q^{t+1} \tag{3.4.36}$$

$$= off. \tag{3.4.37}$$

There is no state $q' \in Q$ such that $\phi(q) = Y^{t+2}$. Taken together, this means that for any $q, q' \in Q$ and $\sigma \in \Sigma$,

$$\delta(q,\sigma) \mapsto q' \iff \gamma(\phi(q), \psi(\sigma)) \mapsto \phi(q'). \tag{3.4.38}$$

Now, given an input sequence $w \in \Sigma^*$ with $w = \sigma_{c_1}\sigma_{c_2}\ldots\sigma_{c_l}$,

$$q_1 = \delta(q_0, \sigma_{c_1}) \iff \phi(q_1) = \gamma(\phi(q_0), \psi(\sigma_{c_1})) \tag{3.4.39}$$

$$q_2 = \delta(q_1, \sigma_{c_2}) \iff \phi(q_2) = \gamma(\phi(q_1), \psi(\sigma_{c_2})) \tag{3.4.40}$$

$$\vdots \tag{3.4.41}$$

$$q_l = \delta(q_{l-1}, \sigma_{c_l}) \iff \phi(q_l) = \gamma(\phi(q_{l-1}), \psi(\sigma_{c_l})) \tag{3.4.42}$$

$$q_l \in F \iff \phi(q_l) \in F_{BN}. \tag{3.4.43}$$

Finally, note that from any state, applying the control input that corresponds to expressing the START gene for two time steps pushes the state of the Boolean network into the state $Y = \phi(q_0)$. This means that a sequence w is accepted by M if and only if the Boolean network is in state $Y^{t=2(|w|+1)+1} \in F_{BN}$ after applying control

input $h_{BN}(w,t)$ (Eq. (3.3.22)) from time $t = 0$ to $t = 2(|w| + 1)$. Thus, as proved by induction, the Boolean network model of $g(M)$ simulates M. □

Corollary 3.4.2. *Modeled as a Boolean network, gene regulatory networks are computationally equivalent to finite state machines.*

Proof. Two frameworks for computation are equivalent if and only if any construction in one framework can be simulated by a construction in the other framework. As shown in Theorem 1, given an FSM M, the Boolean network model of the gene regulatory network $g(M)$ simulates M. Now, given a GRN $G = (V, E_r, E_a)$ modeled as a Boolean network, it is easy to see that because there are a finite number of species in the Boolean network model, and each species may be in only one of two states, there must also be finite number of states in the Boolean network model. It is therefore always possible to construct an FSM $M = (Q, \Sigma, \delta, q_0, F)$ that simulates G. For example, let the set of states to the FSM be the set of states to the Boolean network, $Q = \mathbb{B}^{|V|}$, and let the set of input symbols be a subset $\Sigma \subseteq \mathbb{B}^{|V|}$. For every state vector $Y, Y' \in \mathbb{B}^{|V|}$ and for every input vector U, if $Y' = f(Y, U)$ in the Boolean network model, add the mapping $\delta(Y, U) \mapsto Y'$ to the transition function in the FSM. Since the Boolean network model of any GRN can be simulated by an FSM, GRNs are computationally equivalent to FSMs. □

3.5 DDEs Approximate the Behavior of the Boolean Network

One method for examining how well the behavior of the DDE model approximates the ideal Boolean network model is to simply compare the state of each model after applying a prescribed input. To illustrate this method, a new example is introduced, the modulo-two pulse counter. The finite state machine encoding the modulo-two pulse counter is shown in Figure 3.5.1A. There are two input symbols to this machine, $\Sigma = \{a, \epsilon\}$, and the machine accepts all sequences of input symbols that end in an ϵ. Viewed another way, any accepted input sequence can be split into a sequence of contiguous a symbols interrupted by sequences

of contiguous "ϵ" symbols. In this way, an accepted input sequence consisting of an even number of contiguous a symbols should end in state 0, while an accepted input sequence consisting of an odd number of continuous a sequences should end in state 2. The GRN implementation of this machine is shown in Figure 3.5.1B. If a biomolecular implementation is imagined where signal molecule ϵ is available except in the presence of the input signal a, this machine counts discrete pulses of a modulo two.

The Boolean network model of the GRN in Figure 3.5.1B simulates the FSM shown in Figure 3.5.1A. Initially, input signal a is absent and signal ϵ is present, all state genes are *on*, and all transition genes are *off*. At time $t = 0$, the START gene is *on*, leading to the repression of R_0 at time $t = 1$. The machine then transitions between states 0 through 3 according to the prescribed pulses of input signal a. Figure 3.5.1C illustrates a few sample trajectories of the DDE model against the trajectory produced by the ideal Boolean network model for an input of five pulses of equal duration of signal molecule a followed by signal molecule ϵ. In these trajectories, $\beta = V_{max}$, $\tau = 1$, $k_{1/2} = 0.2$, and V_{max} and n were varied.

Intuitively, for a fixed τ, increasing V_{max} and β will improve the dynamic response of the DDE model, and increasing the Hill coefficient n results in a sharper sigmoidal response. This is reflected in Figure 3.5.1C where the DDE model for $V_{max} = \beta = 100$ and $n = 2.5$ track the ideal Boolean network trajectory more closely than the model where $V_{max} = \beta = 20$ and $n = 2$. Additionally, varying $k_{1/2}$ affects the sensitivity of expression to upstream transcription factors. The duration of the input pulse may also be increased relative to τ. For example, the control input shown in Figure 3.5.1C applies $START$ followed by five pulses of the input signal a according to a square wave with pulse width of $\Delta t = 2$. A precise value for τ may not be known, and in networks like the modulo-two pulse counter, more reliable performance may be achieved by allowing the network to settle into a periodic orbit before changing inputs.

To quantitatively measure the effects of these parameters, the following two metrics compare the state of gene expression at a fixed time after applying a control input in a DDE model to the ideal Boolean network. Where $\hat{R}_q(t)$ and $R_q(t)$ are the values of expression for gene

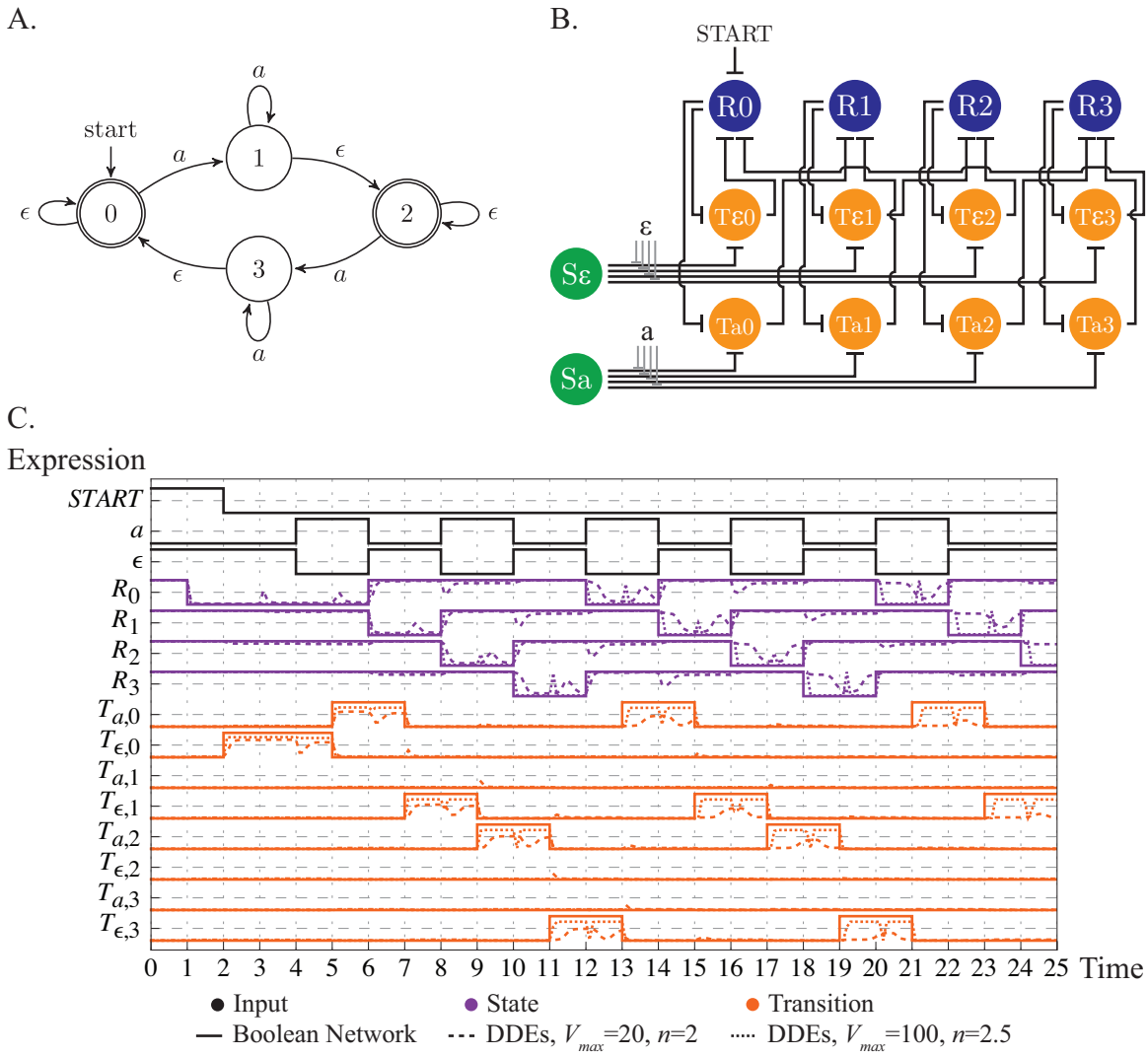


Figure 3.5.1: Modulo-two pulse counter machine described as (A) a directed graph representation of a finite state machine, (B) a gene regulatory network. In the GRN representation, an unspecified mechanism makes the ϵ signal available except in the presense of the a signal. (C) Boolean network and DDE trajectories for the modulo-two pulse counter machine specified by the GRN. In this simulation, $\tau = 1$, $k_{1/2} = 0.2$, and $\beta = V_{max}$. V_{max} and n are varied, with larger values of V_{max} and n resulting in trajectories that more closely follow the ideal Boolean network trajectories.

Rq in the DDE model and Boolean network model respectively,

$$e_{avg} = max_{q \in Q} \frac{2}{\Delta t} \int_{12\Delta t}^{12.5\Delta t} |R_q(t) - \hat{R}_q(t)| dt \tag{3.5.1}$$

$$e_{thresh} = \begin{cases} 0, & e_{avg} < 1/2, \\ 1, & e_{avg} \geq 1/2 \end{cases}. \tag{3.5.2}$$

Eq. (3.5.1) describes error e_{avg}, which is the L_∞ norm on the average error Eq. (3.5.1) takes the maximum average error e_{avg}, between the DDE model and Boolean network model for the gene products R_q for all $q \in Q$ for half a pulse width following the five pulses of the input signal a. In Figure 3.5.1, e_{avg} the maximum average difference between the DDE model and Boolean network model for gene products R_0, R_1, R_2, and R_3 on the time interval $t = 24$ to $t = 25$. Eq. (3.5.2) digitizes this error with a threshold of $\frac{1}{2}$ as e_{thresh}. Figure 3.5.2 shows the average error and thresholded error over a range of parameters for the pulse counter machine given $V_{max} = \beta$ and $\tau = 1$, and nominal values of $\Delta t = 20$, $V_{max} = \beta = 10$, and $k_{1/2} = 0.3$. The figure shows that for this machine, long pulse widths and larger values of V_{max} and β result in less error for all values of n, and that for smaller values of n, there is an optimal value of $k_{1/2}$ near 0.3.

3.6 The FSM Framework in a Cellular Information Processing Context

In the previous two examples, finite state machines were shown as complete circuits. Many synthetic and naturally occurring biomolecular circuits have been developed and characterized, and in the larger context of cellular information processing, finite state machines may be used to specify the logical control that wires together a broad array of biomolecular sensor and effector circuits. This strategy may be an intuitive way to design multicellular behaviors. To illustrate this concept, a specification is presented for a microcolony edge detection circuit built around finite state machine control logic, and implemented as a GRN.

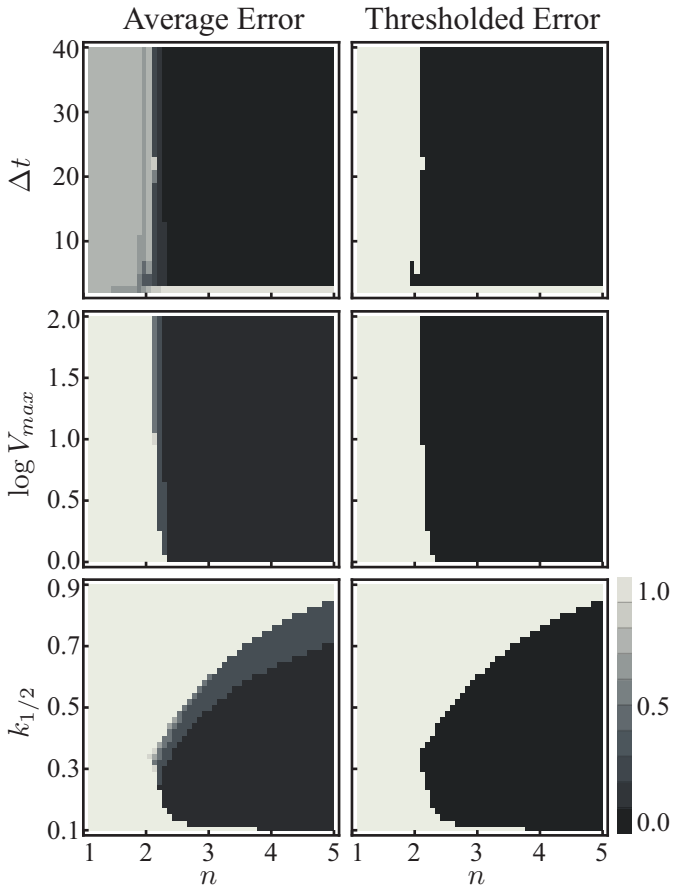


Figure 3.5.2: Average error and threshold error for the modulo-two pulse counter illustrated in Figure 3.5.1. Error metrics (Eqs. (3.5.1–3.5.2)) compare the state of the DDE model to the state of the ideal Boolean network model. Error is computed following a control input of five pulses of signaling molecule a. The heat map shows the error for varying Hill coefficient n over a range of values for $k_{1/2}$, $\log V_{max}$, and input pulsewidth Δt, given $\tau = 1$ and $\beta = V_{max}$. Nominally $\tau = 1$, $\Delta t = 20$, $V_{max} = \beta = 10$, and $k_{1/2} = 0.3$. Zero error means that the DDE model and Boolean network model end in the same state, while an error of one means there is maximal error between models. In this example, increasing pulse width Δt, the rate of production and degradation dynamics V_{max} and β, or the Hill coefficient n, improves performance.

3.6.1 Example: Bacterial Microcolony Edge Detection Circuit

Figure 3.6.1A depicts a specification for a microcolony edge detection circuit. The figure shows the information flow between the control logic modules (labeled "wave generator", "edge detection", and "toggle switch"), and biomolecular sensors and effectors (labeled "stochastic pulse generator", "band pass filter", and "timer"). Each control module is specified by a finite state machine. Here the notion of an FSM $A = (Q, \Sigma, \delta, q_0, F)$ is amended to include a finite set of output symbols Λ and a multivalued output function,

$$\gamma : Q \times \Sigma \to \Lambda. \tag{3.6.1}$$

On the arrival of each input symbol σ, when the FSM is in state $q \in Q$, in addition to following transition $\delta(\sigma, q) \mapsto q'$ the machine emits output symbol $\lambda = \gamma(\sigma, q)$ with $q' \in Q$ and $\lambda \in \Lambda$. This is similar to the operation of a type of a finite state machine called a *Mealy machine* [98]. As with input symbols, the biomolecular realization of an output symbol is the upregulation of a transcription factor, or enzymatic production of an inducer or signaling molecule that may be exported from the cell. In this way, output symbols are a medium for intercellular or intracellular communication.

For each control module in Figure 3.6.1, the state transition map $\delta(\sigma, q) \mapsto q'$ and output map $\gamma(\sigma, q) \mapsto \lambda$ are represented as a set of directed edges labeled $\frac{\sigma}{\lambda}$ connecting node q to node q'. If there is no output symbol $\lambda \in \Lambda$ associated with a state transition $\delta(\sigma, q) \mapsto q'$, then the edge is simply labeled σ. For brevity, if multiple input symbols, say σ_1 and $\sigma_2 \in \Sigma$ result in the same transition from state q to q', the input symbols are enumerated as $\sigma_1|\sigma_2$ in the edge label. Similarly, if the output function $\gamma(q, \sigma)$ is multivalued, say $\gamma(q, \sigma) \mapsto \lambda_1$ and $\gamma(q, \sigma) \mapsto \lambda_2$, the output symbols are enumerated as λ_1 & λ_2. For example, in the wave generation module of Figure 3.6.1A the edge labeled $\epsilon^0|\frac{k}{\texttt{emit}}$ specifies the transition function,

$$\delta(Q_0^0, \epsilon^0) \mapsto Q_0^0 \tag{3.6.2}$$

$$\delta(Q_0^0, k) \mapsto Q_0^0, \tag{3.6.3}$$

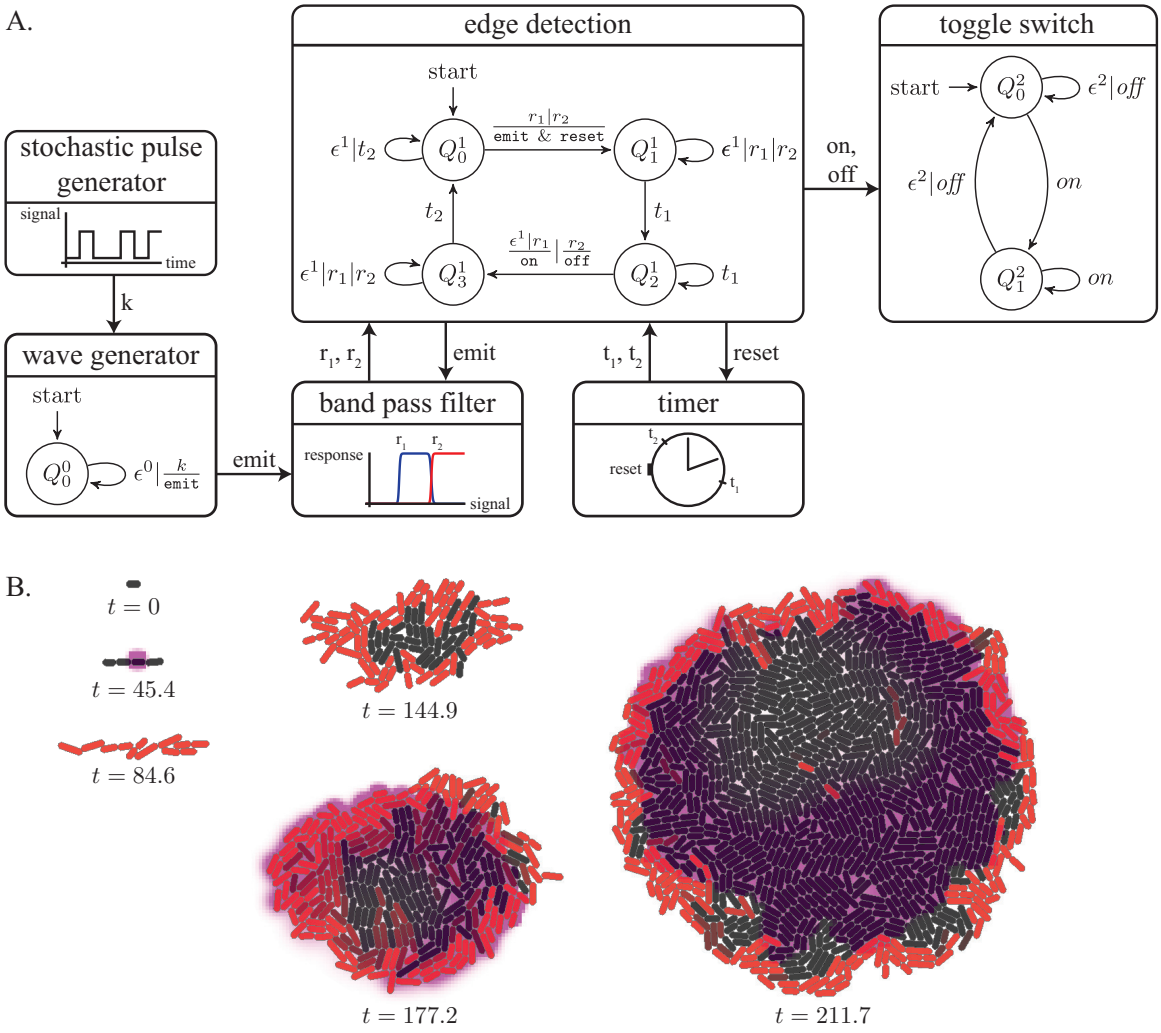


Figure 3.6.1: Finite state machine specification and snapshots from a `gro` simulation of the bacterial microcolony edge detection circuit. (A) The edge detection circuit consists of three asynchronous and parallel finite state machines and three sensor/effector modules based on existing biomolecular circuits. The stochastic pulse generator is a source of genetic noise, the band pass filter responds to middle and high concentrations of a diffusable *emit* signal, and the timer emits an intercellular signal at times t_1 and t_2 after receiving a pulse of the *reset* signal. (B) A homogeneous microcolony grows from a single cell. As the microcolony grows, cells stochastically begin a "wave". Cells relay the wave and measure the local concentration of the diffusable *emit* signal after a short refractory period, to determine whether a cell is in the middle or on the edge of a microcolony. Cells on the edge move to a RFP producing state, while cells in the middle relax to a non-RFP producing state.

and output function γ is defined as,

$$\gamma(Q_0^0, k) \mapsto \texttt{emit}. \tag{3.6.4}$$

The biomolecular realization of this specification is that signals ϵ^0 and k both leave the system in state Q_0^0, however a transition on signal k results in the upregulation of the *emit* signaling molecule.

In operation, the wave generator, edge detection, and toggle switch control logic modules in Figure 3.6.1A act in parallel with a stochastic pulse generator, a concentration band pass filter, and a timer through named communication channels. Consider the *emit* output symbol to be an intercellular communication channel, and all other output symbols are intracellular. In a biomolecular realization, the *emit* symbol can be implemented as a diffusable signaling molecule that is exported from the cell. The intuition behind this design is that at a stochastic rate, any cell in the microcolony can emit a pulse of a diffusable signaling molecule that is sensed and relayed by neighboring cells. By sensing the local concentration of signaling molecules a short time after relaying a wave, a cell can determine whether or not it is on the edge of a microcolony. Cells that detect a high local concentration of signaling molecules have many close neighbors that relayed a wave recently, and thus not on the edge of the microcolony. Alternatively, cells that detect a relatively low concentration of signaling molecules have fewer neighbors, indicating that the cell is on the edge of the microcolony. When a cell detects that is it on the edge of a microcolony, it goes into a RFP producing state. When the cell no longer senses that it is on the edge of a microcolony, it stops production of RFP. This high-level specification was implemented in the 2D simulation environment `gro`, robustly producing the red ring colony phenotype illustrated in Figure 3.6.1. More detailed discussion of the specification can be found in Appendix B.

The high level specification in Figure 3.6.1 can be refined by replacing finite state machines with gene regulatory networks, and specifying sensors and effectors as input/output modules. Figure 3.6.2 illustrates how the GRN implementations of the logical control modules

are interconnected with sensors and effectors. The wave generator, edge detection, and toggle switch FSMs are modular components that realized as three distinct GRNs. The stochastic pulse generator, band pass filter, and timer sensors and effectors are biomolecular modules that take a signaling molecule or transcription factor as input, and produce a signaling molecule or transcription factor as output. Transition genes in the FSMs are used as input to sensors and effectors, and the output of sensors and effectors are then wired to the production of signals in the GRN realizations of the FSMs.

3.7 Discussion

I showed that any finite state machine M can be implemented as a gene regulatory network $g(M)$ made entirely of nominally "on" repressing transcription factors. I presented this result in the context of a Boolean network model of gene regulatory networks, where finite state machines are computationally equivalent to gene regulatory networks. Furthermore, I described a construction for arbitrary finite state machines with a set of simple biomolecular parts. I presented a delay differential equation model for the biomolecular construction, and showed through example how the DDE model can be compared to the ideal Boolean network model using a metric induced by the L_∞ norm. Additionally, I showed that the behavior of the DDE model is close to the behavior of the Boolean network model over a range of physically relevant parameters. Finally, I applied the framework to a bacterial microcolony edge detection example, using a `gro` simulation to show that our approach can be used in a more general cellular information processing context to implement asynchronous logical control interfacing with a set of biomolecular sensors and actuators.

3.8 Methods

All Boolean network and DDE simulations are performed in Mathematica, Version 8.0 [83], with numerical solver NDSolve. The microcolony edge detection example was simulated in `gro`, Version beta.4 [42]. Mathematica and `gro` files are available upon request.

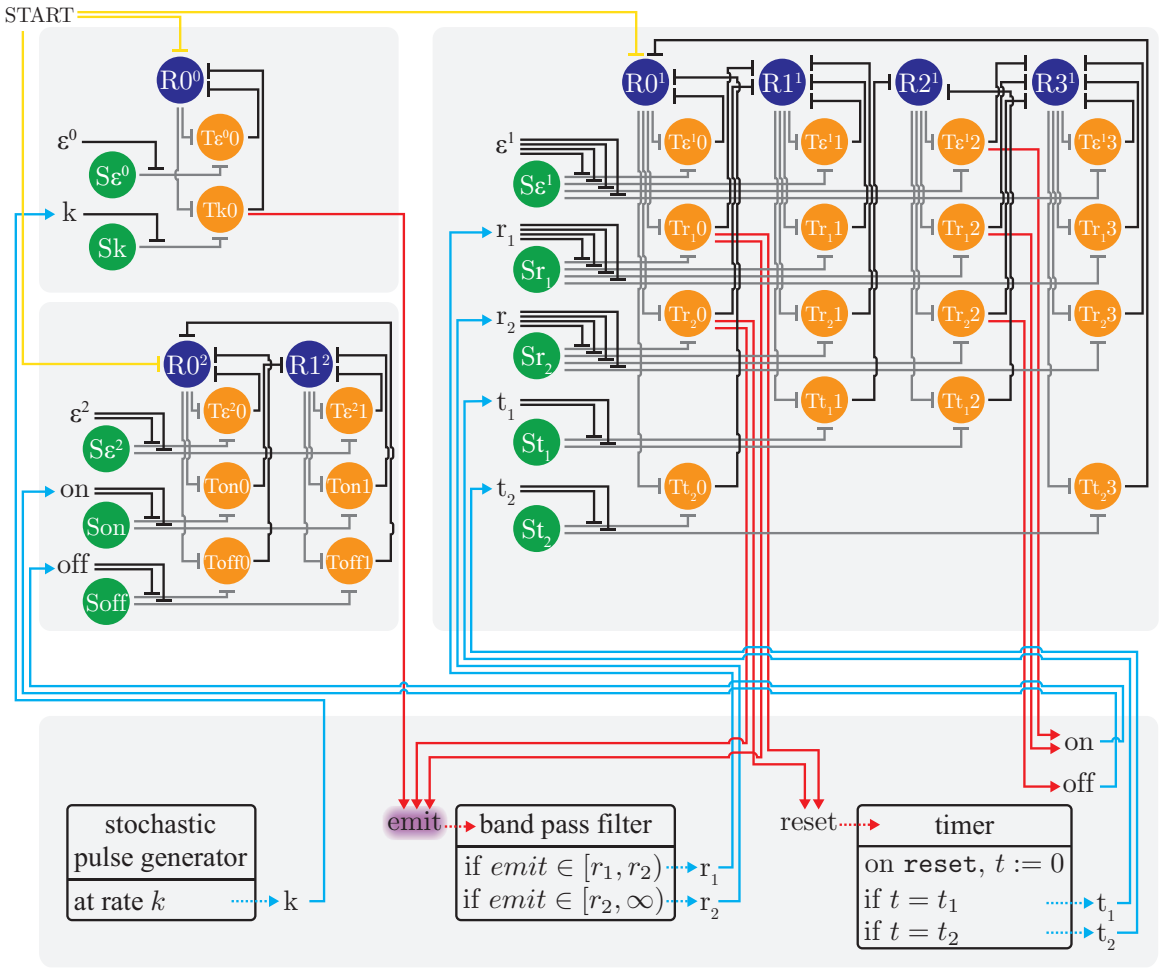


Figure 3.6.2: Gene regulatory network realization of the bacterial microcolony edge detection circuit depicted in Figure 3.6.1. Finite state machine modules are separated by gray modules. The upper left module encodes the wave generator, the module below it encodes the toggle switch, and the module in the upper right encodes the edge detection FSM. The bottom module contains specifications for the unrealized sensors and effectors. Sensor and effector specifications consist of an optional input signal, output signal, and a description of the behavior of the module. Red lines depict how genes in the finite state machines are wired to sensors, and blue lines depict how sensors are wired to signals. The purple cloud around *emit* indicates that *emit* is an external diffusable cell-cell signaling molecule; all other signals are considered internal.

Chapter 4

CONCLUSIONS

For this dissertation, I designed two tool chains for compiling high-level specifications into biomolecular implementations. I successfully demonstrated that by carefully choosing representations of mathematical objects (i.e. dual rail representation of signals), the tools and results of control theory and computer science can be applied to the design of biomolecular reaction networks and genetic circuits. In Chapter 2 I showed how the framework of Linear I/O systems can be used to design robust, scalable, and composable, biochemical reaction networks. In Chapter 3 I illustrated a method for compiling finite state machines into a gene regulatory network, and showed mathematically that a well-accepted Boolean network model of gene regulation is computationally equivalent to finite state machines. Additionally, I demonstrated quantitatively that a more realistic delay differential equation model of gene regulation can approximate the Boolean network model for a physically realistic set of parameters. Furthermore, I demonstrated how finite state machines and linear I/O systems might provide a framework for specifying and analyzing complex behaviors in a cellular information processing framework. Specifically, this was shown in the context of a bacterial microcolony edge detection circuit this is specified from the input/output behavior of circuits that have been implemented in living cells.

Our gene regulatory network and biomolecular constructions were optimized for clarity and simplicity of parts, and not for performance. Other network topologies and biomolecular implementations may perform more robustly or be better suited for a particular task. For example, a chemical system incorporating seesaw gates [12] or transcription enzymes might be more robust for a particular experimental condition or linear I/O system specification. A FSM system utilizing recombinase and genome editing may result in greater state stability [16]. Incorporating new parts, like activators, analog sensors, or molecular insulation devices

[74, 99], could also improve performance. Additionally, many different finite state machines may encode the same control logic or recognize the same language over input symbols. It is certainly the case that given a desired high level behavior, some finite state machines result in a more robust biomolecular implementation that others. Our metric for comparing the behavior of a continuous time and continuous space system to an ideal Boolean network model can be used to explore and optimize over the space of possible machines. More detailed analysis of the low level models (DDEs for GRNs, and the DNA model for the scheme developed by Soloveichik et al.) could be done, but the issues addressed in such an analysis really depend on the actual implementation.

The results of this work are representative of a new focus on abstraction in synthetic biological systems. In building a design theory for chemistry, chemical reactions networks are usually the most natural intermediate representation—the middle of the "hourglass" [100]. Many different high level languages and formalisms have been and can likely be compiled to chemical reactions, and chemical reactions themselves (as an abstract specification) can be implemented with a variety of low level molecular mechanisms. Similarly, gene regulatory networks are usually the most natural intermediate representation for synthetic gene circuits in living cells, and gene regulatory networks themselves may be implemented with a variety of biomolecular mechanisms. This framework enables the synthetic biologist to ask and answer new and specific questions about synthetic biomolecular mechanisms. For example, one may ask specifically how the CRISPR system and small molecule sensing might be used to implement a eukaryotic finite state machine [60, 89]. I have outlined one approach to answering this question in Appendix Section D.1, which may be a fruitful direction of research.

Additionally, this work has implications to understanding the biology of cells, possibly suggesting a new way to relate cell state to observed patterns of gene expression, and more generally, hinting at the computational power and limitations of cells. A single cell can recognize any sequence of molecular inputs that can be represented as a regular expression, however no arrangement of a gene regulatory network can enable a single cell to recognize more complex sets of sequences such as those specified by context-free or recursively

enumerable languages. This means, for example, that a single cell can be programmed to compute the sum of two integers represented by signal pulses, however a single cell could not, without some extra internal memory of variable size, compute the product of two arbitrary integers, or recognize when an arbitrary sequence consists of an equal number of pulses of two chemical inducers.

This observation suggests that multicellular organisms may have evolved precisely because doing so enables a more powerful form of computation. Indeed, given single cells that implement finite state machines, it is a small theoretical step to arrive at multicellular systems that are capable of more complex computation. L-systems, for example, are a cell-based model for plant development in which cells have finite state that when taken together is computationally equivalent to pushdown automata [27]. Another rich area of research may be in developing more complex models of computation based on growing, dividing, cells. In Appendix Section D.2, I outline one way in which a linear arrangement of cells such as cyanobacteria could hypothetically implement a Turing tape machine. Future work in this area might more carefully explore the computational power and limitations of multicellular systems, and the relation between computational complexity and functional morphology, in more detail.

BIBLIOGRAPHY

[1] Seelig G, Soloveichik D, Zhang DY, Winfree E (2006) Enzyme-free nucleic acid logic circuits. *Science* 314:1585–8.

[2] Kim J, White KS, Winfree E (2006) Construction of an in vitro bistable circuit from synthetic transcriptional switches. *Mol Syst Biol* 2:68.

[3] Zhang DY, Turberfield AJ, Yurke B, Winfree E (2007) Engineering entropy-driven reactions and networks catalyzed by DNA. *Science* 318:1121–5.

[4] Yurke B, Turberfield AJ, Mills AP, Simmel FC, Neumann JL (2000) A DNA-fuelled molecular machine made of DNA. *Nature* 406:605–8.

[5] Qian L, Soloveichik D, Winfree E (2011) Efficient turing-universal computation with DNA polymers. *DNA computing and molecular programming* pp 123–140.

[6] Soloveichik D, Cook M, Winfree E, Bruck J (2008) Computation with finite stochastic chemical reaction networks. *Natural Computing* 7:615–633.

[7] Zhang DY, Winfree E (2009) Control of DNA strand displacement kinetics using toehold exchange. *J Am Chem Soc* 131:17303–14.

[8] Green C, Tibbetts C (1981) Reassociation rate limited displacement of DNA strands by branch migration. *Nucleic Acids Res* 9:1905–18.

[9] Panyutin IG, Hsieh P (1993) Formation of a single base mismatch impedes spontaneous DNA branch migration. *J Mol Biol* 230:413–24.

[10] Soloveichik D, Seelig G, Winfree E (2010) DNA as a universal substrate for chemical kinetics. *Proc Natl Acad Sci USA* 107:5393–8.

[11] Schulman R, Winfree E (2010) Programmable control of nucleation for algorithmic self-assembly. *SIAM Journal on Computing* 39:1581–1616.

[12] Qian L, Winfree E (2011) A simple DNA gate motif for synthesizing large-scale circuits. *J R Soc Interface* 8:1281–97.

[13] Costa Santini C, Bath J, Tyrrell AM, Turberfield AJ (2013) A clocked finite state machine built from DNA. *Chem Commun (Camb)* 49:237–9.

[14] Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in Escherichia coli. *Nature* 403:339–42.

[15] Ellis T, Wang X, Collins JJ (2009) Diversity-based, model-guided construction of synthetic gene networks with predicted functions. *Nat Biotechnol* 27:465–71.

[16] Friedland AE, et al. (2009) Synthetic gene networks that count. *Science* 324:1199–202.

[17] Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. *Nature* 403:335–8.

[18] Miyamoto T, Razavi S, DeRose R, Inoue T (2012) Synthesizing biomolecule-based boolean logic gates. *ACS Synth Biol.*

[19] Basu S, Gerchman Y, Collins CH, Arnold FH, Weiss R (2005) A synthetic multicellular system for programmed pattern formation. *Nature* 434:1130–4.

[20] Sohka T, et al. (2009) An externally tunable bacterial band-pass filter. *Proc Natl Acad Sci USA* 106:10135–40.

[21] Danino T, Mondragón-Palomino O, Tsimring L, Hasty J (2010) A synchronized quorum of genetic clocks. *Nature* 463:326–30.

[22] You L, Cox RS, Weiss R, Arnold FH (2004) Programmed population control by cell-cell communication and regulated killing. *Nature* 428:868–71.

[23] Regot S, et al. (2011) Distributed biological computation with multicellular engineered networks. *Nature* 469:207–11.

[24] Tamsir A, Tabor JJ, Voigt CA (2011) Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. *Nature* 469:212–5.

[25] Liu C, et al. (2011) Sequential establishment of stripe patterns in an expanding cell population. *Science* 334:238–41.

[26] Lindemayer A (1968) Mathematical models for cellular interaction in development I and II. *J. Theoret. Biol.(18)* 280:315.

[27] Prusinkiewicz P, Lindenmayer A (1996) *The Algorithmic Beauty of Plants* (Springer).

[28] Sipser M (2013) *Introduction to the theory of computation* (Cengage Learning, Boston, MA).

[29] Hopcroft JE, Motwani R, Ullman JD (2007) *Introduction to automata theory, languages, and computation* (Pearson/Addison Wesley, Boston).

[30] Burks AW, Von Neumann J (1966) *Theory of self-reproducing automata* (University of Illinois Press).

[31] Turing AM (1937) *On Computable Numbers, with an Application to the Entscheidungsproblem: A Correction* (C.F. Hodgson & Son), p 3.

[32] Gardner M (1970) Mathematical games: The fantastic combinations of John Conway's new solitaire game life. *Scientific American* 223:120–123.

[33] Wolfram S (1983) Statistical mechanics of cellular automata. *Reviews of modern physics* 55:601.

[34] Wolfram S (2002) *A new kind of science* (Wolfram Media, Champaign, IL).

[35] Abelson H, et al. (2000) Amorphous computing. *Commun. ACM* 43:74–82.

[36] Coore D, Nagpal R, Weiss R (1997) Paradigms for structure in an amorphous computer., Technical report.

[37] Coore DN (1999) Ph.D. thesis (Massachusetts Institute of Technology).

[38] Nagpal R (2002) *Programmable self-assembly using biologically-inspired multiagent control* (ACM), pp 418–425.

[39] Nagpal R, Kondacs A, Chang C (2003) *Programming methodology for biologically-inspired self-assembling systems.*

[40] Stoy K, Nagpal R (2007) Self-reconfiguration using directed growth. *Distributed Autonomous Robotic Systems 6* pp 3–12.

[41] Yamins D, Nagpal R (2008) *Automated global-to-local programming in 1-d spatial multi-agent systems* (International Foundation for Autonomous Agents and Multiagent Systems), pp 615–622.

[42] Jang SS, Oishi KT, Egbert RG, Klavins E (2012) Specification and simulation of synthetic multicelled behaviors. *ACS Synth Biol* 1:365–374.

[43] Chomsky N (1956) Three models for the description of language. *IEEE Transactions on Information Theory* 2:113–124.

[44] Moore EF (1956) Gedanken-experiments on sequential machines. *Automata studies* 34:129–153.

[45] Rabin MOO, Scott D (1959) Finite automata and their decision problems. *IBM Journal of Research and Development* 3:114–125.

[46] Minsky ML (1967) *Computation: Finite and Infinite Machines.*

[47] Burks AW, Wang H (1957) The logic of automata–part i. *J. ACM* 4:193–218.

[48] Minsky ML (1961) Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. *Annals of Mathematics* 74:437–455.

[49] McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology* 52:99 – 115.

[50] Kleene SC (1951) Representation of events in nerve nets and finite automata.

[51] Wieland M, Fussenegger M (2012) Engineering molecular circuits using synthetic biology in mammalian cells. *Annu Rev Chem Biomol Eng* 3:209–34.

[52] Lohmueller JJ, Armel TZ, Silver PA (2012) A tunable zinc finger-based framework for Boolean logic computation in mammalian cells. *Nucleic Acids Res.*

[53] Wang B, Kitney RI, Joly N, Buck M (2011) Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology. *Nat Commun* 2:508.

[54] Becskei A, Séraphin B, Serrano L (2001) Positive feedback in eukaryotic gene networks: cell differentiation by graded to binary response conversion. *EMBO J* 20:2528–35.

[55] Hasty J, McMillen D, Collins JJ (2002) Engineered gene circuits. *Nature* 420:224–230.

[56] Bonnet J, Subsoontorn P, Endy D (2012) Rewritable digital data storage in live cells via engineered control of recombination directionality. *Proc Natl Acad Sci USA* 109:8884–9.

[57] Siuti P, Yazbek J, Lu TK (2013) Synthetic circuits integrating logic and memory in living cells. *Nat Biotechnol* 31:448–52.

[58] Tabor JJ, et al. (2009) A synthetic genetic edge detection program. *Cell* 137:1272–81.

[59] Lu TK, Khalil AS, Collins JJ (2009) Next-generation synthetic gene networks. *Nat Biotechnol* 27:1139–50.

[60] Qi LS, et al. (2013) Repurposing CRISPR as an RNA-guided platform for sequence-specific control of gene expression. *Cell* 152:1173–83.

[61] Beerli RR, Barbas CF (2002) Engineering polydactyl zinc-finger transcription factors. *Nat Biotechnol* 20:135–41.

[62] Mandell JG, Barbas CF (2006) Zinc finger tools: custom DNA-binding domains for transcription factors and nucleases. *Nucleic Acids Res* 34:W516–23.

[63] Moscou MJ, Bogdanove AJ (2009) A simple cipher governs DNA recognition by TAL effectors. *Science* 326:1501.

[64] Li Y, Moore R, Guinn M, Bleris L (2012) Transcription activator-like effector hybrids for conditional control and rewiring of chromosomal transgene expression. *Sci Rep* 2:897.

[65] Khalil AS, et al. (2012) A synthetic biology framework for programming eukaryotic transcription functions. *Cell* 150:647–58.

[66] Havens KA, et al. (2012) A synthetic approach reveals extensive tunability of auxin signaling. *Plant Physiol* 160:135–42.

[67] Kämpf MM, et al. (2012) Rewiring and dosing of systems modules as a design approach for synthetic mammalian signaling networks. *Mol Biosyst.*

[68] Blount BA, Weenink T, Vasylechko S, Ellis T (2012) Rational diversification of a promoter providing fine-tuned expression and orthogonal regulation for synthetic biology. *PLoS One* 7:e33279.

[69] Egbert RG, Klavins E (2012) Fine-tuning gene networks using simple sequence repeats. *Proceedings of the National Academy of Sciences.*

[70] Brewster RC, Jones DL, Phillips R (2012) Tuning promoter strength through RNA polymerase binding site design in Escherichia coli. *PLoS Comput Biol* 8:e1002811.

[71] Bacchus W, Fussenegger M (2012) Engineering of synthetic intercellular communication systems. *Metab Eng.*

[72] Klonowski W (1983) Simplifying principles for chemical and enzyme reaction kinetics. *Biophys Chem* 18:73–87.

[73] Dorf RC, Bishop RH (2008) *Modern control systems* (Pearson Prentice Hall, Upper Saddle River, NJ).

[74] Del Vecchio D, Ninfa AJ, Sontag ED (2008) Modular cell biology: retroactivity and insulation. *Mol Syst Biol* 4:161.

[75] Dullerud GE, Paganini FG (2000) *A course in robust control theory : a convex approach* (Springer, New York).

[76] Sun Z (2005) *Switched linear systems : Control and design* (Springer, London).

[77] Khalil HK (2002) *Nonlinear systems* (Prentice Hall, Upper Saddle River, N.J.).

[78] Yin P, Choi HMT, Calvert CR, Pierce NA (2008) Programming biomolecular self-assembly pathways. *Nature* 451:318–22.

[79] Kim J (2007) Ph.D. thesis (California Institute of Technology).

[80] Park SHH, Zarrinpar A, Lim WA (2003) Rewiring MAP kinase pathways using alternative scaffold assembly mechanisms. *Science* 299:1061–4.

[81] Bashor CJ, Helman NC, Yan S, Lim WA (2008) Using engineered scaffold interactions to reshape MAP kinase pathway signaling dynamics. *Science* 319:1539–43.

[82] Grünberg R, Serrano L (2010) Strategies for protein synthetic biology. *Nucleic Acids Res* 38:2663–75.

[83] Wolfram Research, Inc. (2010) Mathematica. *Wolfram Research, Inc.* Version 8.0.

[84] Lawrence PA (1992) *The making of a fly: the genetics of animal design.* (Blackwell Scientific Publications Ltd).

[85] Bonasio R, Tu S, Reinberg D (2010) Molecular signals of epigenetic states. *Science* 330:612–6.

[86] Feng S, Jacobsen SE, Reik W (2010) Epigenetic reprogramming in plant and animal development. *Science* 330:622–7.

[87] Inbar-Feigenberg M, Choufani S, Butcher DT, Roifman M, Weksberg R (2013) Basic concepts of epigenetics. *Fertil Steril* 99:607–15.

[88] Garg A, Lohmueller JJ, Silver PA, Armel TZ (2012) Engineering synthetic TAL effectors with orthogonal target sites. *Nucleic Acids Res.*

[89] Gilbert LA, et al. (2013) CRISPR-mediated modular RNA-guided regulation of transcription in eukaryotes. *Cell* 154:442–51.

[90] Kauffman SA (1969) Metabolic stability and epigenesis in randomly constructed genetic nets. *J Theor Biol* 22:437–67.

[91] Thomas R (1973) Boolean formalization of genetic control circuits. *J Theor Biol* 42:563–85.

[92] Strogatz SH (2001) Exploring complex networks. *Nature* 410:268–276.

[93] de Jong H (2002) Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology* 9:67–103.

[94] Shen-Orr SS, Milo R, Mangan S, Alon U (2002) Network motifs in the transcriptional regulation network of Escherichia coli. *Nat Genet* 31:64–8.

[95] Bornholdt S (2008) Boolean network models of cellular regulation: prospects and limitations. *J R Soc Interface* 5 Suppl 1:S85–94.

[96] Sadot A, et al. (2013) Information-theoretic analysis of the dynamics of an executable biological model. *PLoS One* 8:e59303.

[97] Weiss JN (1997) The Hill equation revisited: uses and misuses. *FASEB J* 11:835–41.

[98] Mealy GH (1955) A method for synthesizing sequential circuits. *Bell System Technical Journal* 34:1045–1079.

[99] Daniel R, Rubens JR, Sarpeshkar R, Lu TK (2013) Synthetic analog computation in living cells. *Nature* 497:619–23.

[100] Doyle J, Csete M (2007) Rules of engagement. *Nature* 446:860.

[101] Horvath P, Barrangou R (2010) Crispr/cas, the immune system of bacteria and archaea. *Science* 327:167–70.

[102] Brouns SJJ, et al. (2008) Small crispr rnas guide antiviral defense in prokaryotes. *Science* 321:960–4.

[103] Ishino Y, Shinagawa H, Makino K, Amemura M, Nakata A (1987) Nucleotide sequence of the iap gene, responsible for alkaline phosphatase isozyme conversion in escherichia coli, and identification of the gene product. *J Bacteriol* 169:5429–33.

[104] Hou Z, et al. (2013) Efficient genome engineering in human pluripotent stem cells using cas9 from neisseria meningitidis. *Proc Natl Acad Sci U S A* 110:15644–9.

[105] Mali P, et al. (2013) Rna-guided human genome engineering via cas9. *Science* 339:823–6.

[106] Maeder ML, et al. (2013) Crispr rna-guided activation of endogenous human genes. *Nat Methods* 10:977–9.

[107] McIsaac RS, et al. (2011) Fast-acting and nearly gratuitous induction of gene expression and protein depletion in saccharomyces cerevisiae. *Mol Biol Cell* 22:4447–59.

Appendix A

IMPLEMENTATION AND SIMULATION DETAILS FOR THE CHEMICAL REALIZATION AND DNA IMPLEMENTATION OF LINEAR I/O SYSTEMS

Simulations were generated via mass action kinetics models in *Mathematica* [83]. DNA implementations were designed in the schema of Soloveichik et al. [10]. This section provides details of the chemical reaction networks, rate constants, and initial conditions used to produce simulations appearing in Chapter 2.

A.1 Catalysis, Degradation, Annihilation

Figure 2.5.1 shows simulated trajectories of the DNA implementation of catalysis, degradation, and annihilation reactions. For each initial fuel concentration C_{max}, the rates q_i and q_{max} were tuned in order to approximate a particular ideal chemical reaction. Rate constants q_i and q_{max} used to produce the simulated trajectories in Figure 2.5.1 are shown in Table A.1.1. Of note is that the rate parameters q_i and q_{max} for $C_{max} = 1$ μM are physically realizable for catalysis, degradation, and annihilation [7].

	Catalysis ($M^{-1}s^{-1}$)		Degradation ($M^{-1}s^{-1}$)		Annihilation ($M^{-1}s^{-1}$)	
C_{max} (nM)	q_i	q_{max}	q_i	q_{max}	q_i	q_{max}
1	10^6	10^9	10^6	–	–	10^6
10	10^5	10^8	10^5	–	–	10^6
100	10^4	10^7	10^4	–	–	10^6
1000	10^3	10^6	10^3	–	–	10^6

Table A.1.1: Rate parameters for the DNA implementation of catalysis, degradation, and annihilation, shown in Figure 2.5.1.

A.2 Integration, Gain, Summation

Integration, gain, and summation are the primitive blocks of any linear I/O system. These primitive blocks are approximated by ideal chemical reactions, and the ideal chemical reactions are implemented enzyme-free DNA reactions illustrated in Figure 2.5.1. Figure 2.3.1 illustrates the dynamics ideal chemical realization and DNA implementation of integration, gain, and summation, in response to a square wave input for physically realizable reaction rates and chemical concentrations.

A.2.1 Integration

The linear model, as well the ideal chemical realization, and DNA implementation of integration and the square wave input are illustrated in Figure A.2.1. Using the schema introduced by Soloveichik et al., integration is implemented with DNA using two catalysis and one annihilation reaction. The square wave input is implemented with one annihilation reaction, and two impulse signal species inputs. In practice such an impulse in accomplished by pipetting in a small volume of input species at high concentration. Unregulated inputs in the DNA model are added at twice their concentration in the ideal chemical realization in order to attenuate for the fast sequestration of u^+ and u^- in H_3 and HS_3 respectively. The result is the integration of a square wave input,

$$y(t) = \int_{\tau=0}^{t} \alpha u_1(\tau) dt \tag{A.2.1}$$

$$u_1(t) = \begin{cases} 3.33 \times 10^{-8} & t < 600 \\ -3.33 \times 10^{-8} & \text{otherwise} \end{cases} \tag{A.2.2}$$

$$\alpha \approx 0.00833. \tag{A.2.3}$$

Linear Model	Ideal Chemical Realization	DNA Implementaiton
(a)		
$y(t) = \int_{\tau=0}^{t} \alpha u_1(\tau) d\tau$	$u_1^{\pm} \xrightarrow{\alpha} u_1^{\pm} + y^{\pm}$	$u_1^{\pm} + G_1^{\pm} \xrightarrow{q_1} O_1^{\pm}$
		$O_1^{\pm} + T_1^{\pm} \xrightarrow{q_{max}} u_1^{\pm} + y^{\pm}$
		$y^+ + L_2 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} H_2 + B_2$
	$y^+ + y^- \xrightarrow{\eta} \emptyset$	$y^- + LS_2 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} HS_2 + BS_2$
		$y^- + H_2 \xrightarrow{q_{max}} \emptyset$
$u_1(t) = \begin{cases} 3.33 \times 10^{-8} & 0 \le t < 600 \\ -3.33 \times 10^{-8} & 600 \le t < 1200 \end{cases}$	$u_1^+ + u_1^- \xrightarrow{\eta} \emptyset$	$u_1^+ + L_3 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} H_3 + B_3$
		$u_1^- + LS_3 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} HS_3 + BS_3$
		$u_1^- + H_3 \xrightarrow{q_{max}} \emptyset$
	$u_1^+(0) = 33.33$ nM	$u_1^+(0) = 66.67$ nM
	$u_1^-(600) = 66.67$ nM	$u_1^-(600) = 133.33$ nM

(b)

$\alpha = \frac{1}{2} q_1 C_{max}$	$C_{max} = 1\ \mu$M
$\eta = \frac{1}{2} q_{max} C_{max}$	$q_1 = 1.67 \times 10^4$/M/s
	$q_{max} = 10^6$/M/s

Figure A.2.1: Linear model, ideal chemical realization, and DNA implementation for integration of a square wave input. (a) Integration is approximated by three ideal chemical reactions. The DNA implementation is modeled by eight reactions. The square wave input is implemented by a single annihilation reaction and two instantaneous additions of chemical species at time $t = 0$ and $t = 600$. (b) Rate and concentration parameters for the simulated trajectories that appear in Figure 3a. The initial concentration of fuel species $G_i^{\pm}$, $T_i^{\pm}$, L_i, B_i, LS_i, and BS_i, are set to C_{max}. All other initial concentrations are set to 0 nM unless otherwise specified.

A.2.2 Gain

The linear model, ideal chemical realization, and DNA implementation of gain and the square wave input are illustrated in Figure A.2.2. Gain is implemented in DNA using two catalysis, two degradation, and one annihilation reaction. Again, the square wave input is implemented with one annihilation reaction, and two impulse signal species inputs. As before unregulated inputs u^+ and u^- are added at twice their ideal concentration in order to attenuate for sequestration in the annihilation implementation. The result is a gain multiplied by a square wave input,

$$y(t) = ku_1(t) \tag{A.2.4}$$

$$u_1(t) = \begin{cases} 5 \times 10^{-9} & t < 4000 \\ -5 \times 10^{-9} & \text{otherwise} \end{cases} \tag{A.2.5}$$

$$k = 3. \tag{A.2.6}$$

A.2.3 Summation

Finally, the linear model, ideal chemical realization, and DNA implementation of summation and the two square wave inputs are shown in Figure A.2.3. Two-input summation is implemented in DNA using four catalysis, two degradation, and one annihilation reaction. Each square wave input is modeled with an annihilation reaction and a series of impulse signal species inputs. Again, the unregulated inputs u^+ and u^- are added at twice their ideal concentration to attenuate for sequestration. The result is the summation of two square wave inputs,

$$y(t) = u_1(t) + u_2(t) \tag{A.2.7}$$

$$u_1(t) = \begin{cases} 4 \times 10^{-9} & t \in [0, 5000) \cup [10000, 15000) \\ -4 \times 10^{-9} & \text{otherwise} \end{cases} \tag{A.2.8}$$

$$u_2(t) = \begin{cases} 8 \times 10^{-9} & t < 10000 \\ -8 \times 10^{-9} & \text{otherwise.} \end{cases} \tag{A.2.9}$$

(a)

Linear Model	Ideal Chemical Realization	DNA Implementaiton
$y(t) = ku_1(t)$	$u_1^{\pm} \xrightarrow{\gamma k} u_1^{\pm} + y^{\pm}$	$u_1^{\pm} + G_1^{\pm} \xrightarrow{q_1} O_1^{\pm}$ $O_1^{\pm} + T_1^{\pm} \xrightarrow{q_{max}} u_1^{\pm} + y^{\pm}$
	$y^{\pm} \xrightarrow{\gamma} \emptyset$	$y^{\pm} + G_2^{\pm} \xrightarrow{q_2} \emptyset$
	$y^{+} + y^{-} \xrightarrow{\eta} \emptyset$	$y^{+} + L_3 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} H_3 + B_3$ $y^{-} + LS_3 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} HS_3 + BS_3$ $y^{-} + H_3 \xrightarrow{q_{max}} \emptyset$
$u_1(t) = \begin{cases} 5 \times 10^{-9} & 0 \le t < 4000 \\ -5 \times 10^{-9} & 4000 \le t < 8000 \end{cases}$	$u_1^{+} + u_1^{-} \xrightarrow{\eta} \emptyset$	$u_1^{+} + L_4 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} H_4 + B_4$ $u_1^{-} + LS_4 \underset{q_{max}}{\overset{q_{max}}{\rightleftharpoons}} HS_4 + BS_4$ $u_1^{-} + H_4 \xrightarrow{q_{max}} \emptyset$
	$u_1^{+}(0) = 5$ nM $u_1^{-}(4000) = 10$ nM	$u_1^{+}(0) = 10$ nM $u_1^{-}(4000) = 20$ nM

(b)

$k = q_1/q_2$	$C_{max} = 1\ \mu$M
$\gamma = \frac{1}{2} q_2 C_{max}$	$q_1 = 1.5 \times 10^4$/M/s
$\eta = \frac{1}{2} q_{max} C_{max}$	$q_2 = 0.5 \times 10^4$/M/s
	$q_{max} = 10^6$/M/s

Figure A.2.2: Linear model, ideal chemical realization, and DNA implementation of a gain using a square wave input. (a) Gain is approximated with five ideal chemical reactions. The DNA implementation is modeled with nine reactions. The square wave input is modeled by an annihilation reaction and two instantaneous additions of chemical species at time $t = 0$ and $t = 4000$. (b) Rate and concentration parameters for the simulated trajectories that appear in Figure 3b. Initial concentration of fuel species are set to C_{max}. All other initial concentrations are set to 0 nM.

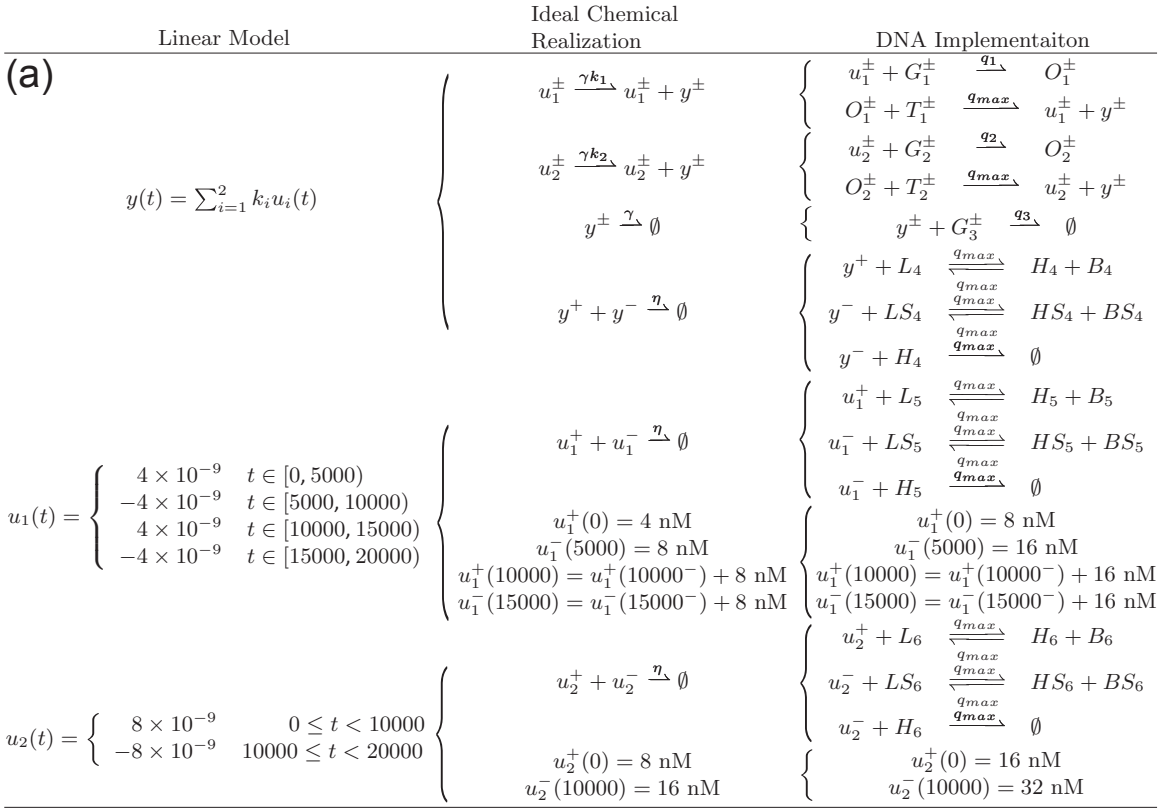


Figure A.2.3: Linear model, ideal chemical realization, and DNA implementation of summation using two square wave inputs. (a) Two-input summation is approximated with seven ideal chemical reactions. The DNA implementation is modeled with 16 reactions. The square wave inputs are modeled by an annihilation reaction for each input signal, as well as instantaneous additions of chemical species at times $t = 0$, 5000, 1000, 15000. (b) Rate and concentration parameters for the simulated trajectories that appear in Figure 3c. Initial concentration of fuel species are set to C_{max}. All other initial concentrations are set to 0 nM.

A.3 Ideal Chemical Reaction Network and DNA Implementation of the PI Controller

The linear model, ideal chemical realization, and DNA implementation of the PI controller with production and degradation disturbances is presented in Figure A.3.1. The optimized

PI controller consists of two weighted summations (four catalysis, two degradation, and one annihilation reaction each), and a weighted integrator (two catalysis reactions and an annihilation reaction). The step input, as before, is implemented with an annihilation reaction and a series of impulse signal species inputs. As mentioned before, in practice an impulse of signal species is generated by pipetting in a small volume of input species at high concentration. Again, in the DNA implementation the unregulated u^+ and u^- inputs are added at twice their ideal concentration to attenuate for sequestration in the annihilation implementation. Identical production and degradation reactions were composed with the ideal chemical realization and DNA model, resulting in an implementation of the PI controller in Figure 2.1.1 with $k_I = k_P = 1$, and $P(s) = \frac{1}{3}$.

(a)

Linear System	Ideal Chemical Realizaton	DNA Implementation
$x_1(t) = u(t) - x_5(t)$	$u^{\pm} \xrightarrow{\gamma} u^{\pm} + x_1^{\pm}$	$u^{\pm} + G_1^{\pm} \xrightarrow{q_1} O_1^{\pm}$ $O_1^{\pm} + T_1^{\pm} \xrightarrow{q_{max}} u^{\pm} + x_1^{\pm}$
	$x_5^{\pm} \xrightarrow{\gamma} x_5^{\pm} + x_1^{\mp}$	$x_5^{\pm} + G_2^{\pm} \xrightarrow{q_2} O_2^{\pm}$ $O_2^{\pm} + T_2^{\pm} \xrightarrow{q_{max}} x_5^{\pm} + x_1^{\mp}$
	$x_1^{\pm} \xrightarrow{\gamma} \emptyset$	$x_1^{\pm} + G_3^{\pm} \xrightarrow{q_3} \emptyset$
	$x_1^{+} + x_1^{-} \xrightarrow{\eta} \emptyset$	$x_1^{+} + L_4 \xrightleftharpoons[q_{max}]{q_{max}} H_4 + B_4$ $x_1^{-} + LS_4 \xrightleftharpoons[q_{max}]{q_{max}} HS_4 + BS_4$ $x_1^{-} + H_4 \xrightarrow{q_{max}} \emptyset$
$\dot{x_4}(t) = k_I x_1(t)$	$x_1^{\pm} \xrightarrow{k_I} x_1^{\pm} + x_4^{\pm}$	$x_1^{\pm} + G_5^{\pm} \xrightarrow{q_5} O_5^{\pm}$ $O_5^{\pm} + T_5^{\pm} \xrightarrow{q_{max}} x_1^{\pm} + x_4^{\pm}$
	$x_4^{+} + x_4^{-} \xrightarrow{\eta} \emptyset$	$x_4^{+} + L_5 \xrightleftharpoons[q_{max}]{q_{max}} H_5 + B_5$ $x_4^{-} + LS_5 \xrightleftharpoons[q_{max}]{q_{max}} HS_5 + BS_5$ $x_4^{-} + H_5 \xrightarrow{q_{max}} \emptyset$
$x_5(t) = k_P x_1 + x_4$ $y(t) = x_6(t) = P x_5(t)$	$x_1^{\pm} \xrightarrow{\gamma k_P} x_1^{\pm} + x_5^{\pm}$	$x_1^{\pm} + G_7^{\pm} \xrightarrow{q_7} O_7^{\pm}$ $O_7^{\pm} + T_7^{\pm} \xrightarrow{q_{max}} x_1^{\pm} + x_5^{\pm}$
	$x_4^{\pm} \xrightarrow{\gamma} x_4^{\pm} + x_5^{\pm}$	$x_4^{\pm} + G_8^{\pm} \xrightarrow{q_8} O_8^{\pm}$ $O_8^{\pm} + T_8^{\pm} \xrightarrow{q_{max}} x_4^{\pm} + x_5^{\pm}$
	$\emptyset \xrightarrow{\gamma\delta_1} x_5^{\pm}$	$\emptyset \xrightarrow{\gamma\delta_1} x_5^{\pm}$
	$x_5^{\pm} \xrightarrow{\gamma(1+\delta_2)} \emptyset$	$x_5^{\pm} + G_9^{\pm} \xrightarrow{q_9} \emptyset$ $x_5^{\pm} \xrightarrow{\gamma\delta_2} \emptyset$
	$y^{\pm} \triangleq x_6^{\pm} \triangleq x_5^{\pm}$	$y^{\pm} \triangleq x_6^{\pm} \triangleq x_5^{\pm}$
	$x_5^{+} + x_5^{-} \xrightarrow{\eta} \emptyset$	$x_5^{+} + L_{10} \xrightleftharpoons[q_{max}]{q_{max}} H_{10} + B_{10}$ $x_5^{-} + LS_{10} \xrightleftharpoons[q_{max}]{q_{max}} HS_{10} + BS_{10}$ $x_5^{-} + H_{10} \xrightarrow{q_{max}} \emptyset$
$u(t) = \begin{cases} 4\times10^{-9} & t \in [0, 75000) \\ -4\times10^{-9} & t \in [75000, 150000) \\ 4\times10^{-9} & t \in [150000, 225000) \\ -4\times10^{-9} & t \in [225000, 300000) \end{cases}$	$u^{+} + u^{-} \xrightarrow{\eta} \emptyset$	$u^{+} + L_{11} \xrightleftharpoons[q_{max}]{q_{max}} H_{11} + B_{11}$ $u^{-} + LS_{11} \xrightleftharpoons[q_{max}]{q_{max}} HS_{11} + BS_{11}$ $u^{-} + H_{11} \xrightarrow{q_{max}} \emptyset$
	$u^{+}(0) = 4$ nM $u^{-}(75000) = 8$ nM $u^{+}(150000) = u^{+}(150000^{-}) + 8$ nM $u^{-}(225000) = u^{-}(225000^{-}) + 8$ nM	$u^{+}(0) = 8$ nM $u^{-}(75000) = 16$ nM $u^{+}(150000) = u^{+}(150000^{-}) + 16$ nM $u^{-}(225000) = u^{-}(225000^{-}) + 16$ nM

(b)

$$k_I = \tfrac{1}{2} q_5 C_{max}$$
$$k_P = q_7/q_8$$
$$\gamma = \tfrac{1}{2} q_i C_{max},\ i \in \{1, 2, 3, 8, 9\}$$
$$\eta = \tfrac{1}{2} q_{max} C_{max}$$
$$P = (1 + \delta_2)^{-1}$$

$$C_{max} = 1\ \mu\text{M}$$
$$q_i = 800/\text{M/s},\ i \in \{1, 2, 3, 5, 7, 8, 9\}$$
$$q_{max} = 10^6/\text{M/s}$$
$$\delta_2 = 2$$

Figure A.3.1: PI Controller with production and degradation disturbances. (a) The PI controller is approximated with 17 ideal chemical reactions, or 19 reactions including the chemical disturbance. The DNA implementation is modeled with 30 reactions, or 33 reactions including the chemical disturbance. The square wave input is modeled by an annihilation reaction and four instantaneous additions of chemical species u^+ and u^- at times $t = 0$, 75000, 150000, 225000. (b) Rate and concentration parameters for the simulated trajectories that appear in Figure 1c. Fuel species $G_i^{\pm}$, $T_i^{\pm}$, L_i, B_i, LS_i, and BS_i, are set to C_{max}. All other species have initial concentration 0 nM.

Appendix B

BACTERIAL MICROCOLONY EDGE DETECTION

Section 3.6.1 discusses a possible implementation of a bacterial microcolony edge detection circuit designed from a finite state machine specification. A high level specification of the circuit is illustrated in Figure 3.6.1. In detail, the operation of the edge detection circuit begins with the stochastic pulse generator module. This effector module supplies the input symbol k to the wave generator at a stochastic rate. On receiving the symbol k, the wave generator upregulates the production of the diffusable signaling molecule *emit*, depicted in Figure 3.6.1 as a pink cloud. The band pass filter module responds to different concentrations of the *emit* signal, producing symbol r_1 for medium concentrations of signal, and r_2 for high concentrations of signal.

The edge detection FSM receives input from the band bass filter and timer modules. From the initial state Q_0^1 of the edge detection FSM, a cell responds to a medium or high local concentration of *emit* signal by upregulating its own production of *emit* and communicating the *reset* symbol to the timer module on the transition from state Q_0^1 to Q_1^1. This has the effect of rippling a wave of signaling molecules across the microcolony. After a short period of time, the timer module provides the input symbol t_1 back to the edge detection FSM, moving the machine to state Q_2^1. From this state, a high local concentration of *emit* signal results in the FSM sending an *off* symbol to the toggle switch FSM, and a lower local concentration of the signal results in the FSM sending an *on* symbol to the toggle switch FSM. The intuition here is that, a high local concentration of the *emit* signal indicates that the cell is surrounded by neighboring cells and detecting their signal wave. A lower local concentration of the *emit* signal indicates that the cell is on the edge of the microcolony.

Finally, the toggle switch FSM stores the boolean state of position of the cell, where state Q_0^2 indicates that a cell is not on the edge of a microcolony, and state Q_1^2 indicates that a cell is on the edge of a microcolony. In the biomolecular realization, transition to state Q_1^2 can be linked to the upregulation of RFP, resulting in a red cell phenotype.

Appendix C

BOOLEAN NETWORK MODEL AS IT RELATES TO NEURAL NETWORKS

Although Kauffman was the first to apply a Boolean network model to GRNs, the notion of a discrete time, discrete state, network had been explored previously. Notably, Minsky investigated the computational complexity of synchronous neural networks built from McClulloch-Pitts cells, and concluded that these networks can simulate any finite state machine, and vice versa [46]. For completeness, the relation between McCulloch-Pitts cells, neural networks, finite state machines, and gene regulatory networks is discussed here.

C.1 McClulloch-Pitts Cells

Inspired by the observation that clusters of simple neurons can be wired together to carry out complex function in the brain such as learning, and motivated to understand how to engineer similar function, McCulloch and Pitts formalized an early model of the neuron [46]. A McCulloch-Pitts neuron takes a vector of Boolean-valued inputs, calculates a piecewise linear combination of those inputs, and compares it to a threshold value to generate a Boolean output. More precisely, given threshold $h \in \mathbb{Z}$, *excitatory* inputs $x_1(t), \ldots, x_n(t) \in \mathbb{B}$, and *inhibitory* inputs $i_1(t), \ldots, i_m(t) \in \mathbb{B}$ at time $t \in \mathbb{N}$, the output of the neuron y at time $t+1$ is,

$$y(t+1) = \begin{cases} 1 & \text{if } \sum_k x_k(t) \geq h \text{ and } \sum_k i_k(t) = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{C.1.1}$$

Furthermore, neurons can be wired together, by setting one or more inputs of a neuron to be the outputs of one or more neurons. A neuron is said to *fire* at time t if its output $y(t) = 1$.

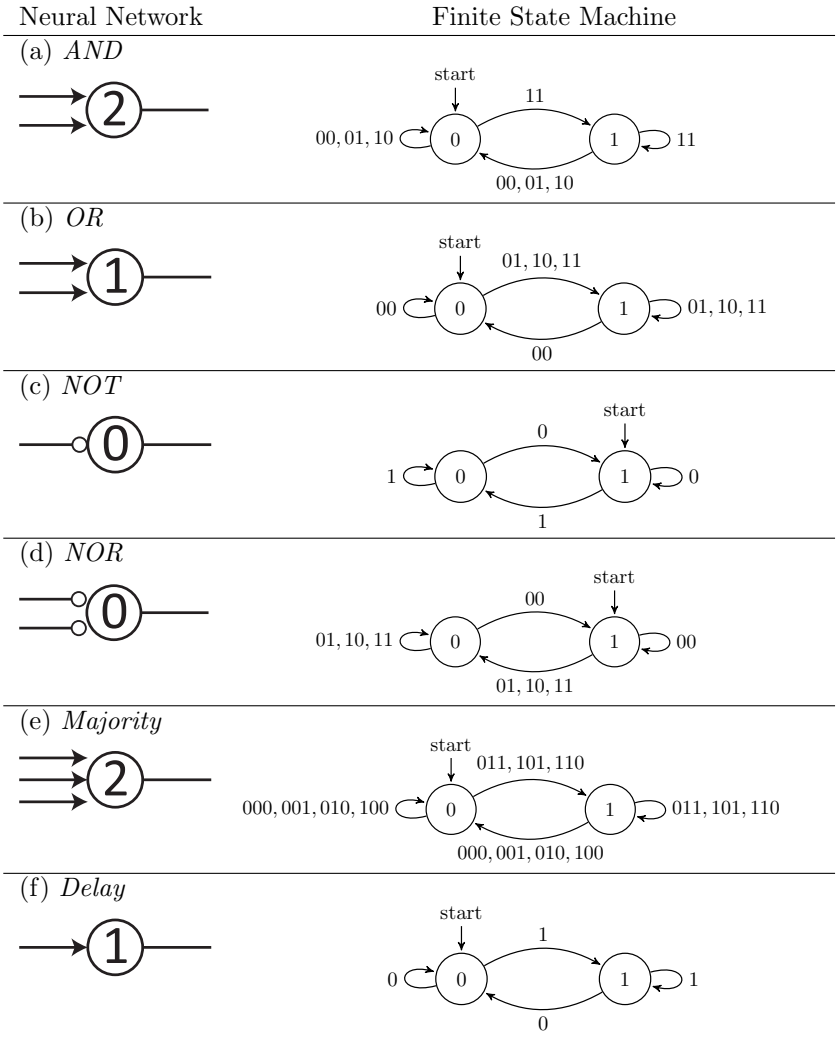


Figure C.1.1: Examples of neural networks made with a single McClulloch-Pitts cell, along with the finite state machine simulating the neuron starting from ground state (inputs set to 0). For each neuron, input fibers are on the left and a single *unlabeled* output fiber is on the right. Each neuron is labeled with the threshold value $h = 0, 1, 2$. Finite state machines have states 0 and 1 corresponding to the neuron firing or not firing. Input symbols are binary sequences corresponding to the state of the inputs fibers (0 or 1). Networks (a)-(d) compute the named Boolean logic operation on their input in a single time step. (e) The *Majority* cell will fire when a majority of the input fibers are set to 1 at the previous time step. (f) The *Delay* function relays the state of the input fiber with a unit delay. Note that by wiring together neurons (a)-(c) and (f), or (c)-(d) and (f), any Boolean function can be computed in a finite number of time steps with a non-recurrent network. As it turns out, (c)-(d) are sufficient to simulate any finite state machine.

This model has a natural graphical representation, and an example is illustrated in Figure C.1.1. In this representation a neuron is denoted by a vertex v_i in a directed graph $G = (V, E)$, with associated time varying Boolean state $x_i(t) \in \mathbb{B}$ and constant threshold $h_i \in \mathbb{Z}$. Typically the vertex is labeled by the threshold value h_i. The *input fibers* to v_i are represented by the set of edges $\cdot \to v_i \in E$, and *output fibers* represented by the set of edges $v_i \to \cdot \in E$. Input and output fibers are labeled *excitatory* or *inhibitory*, and the label is represented graphically with a pointed ($\longrightarrow$) or ball ($-\!\circ$) arrowhead, respectively.

C.2 Neural Networks

A neural network may also be an I/O systems. In this case the state of particular input fibers are specified at every time step. These input fibers appear as half-edges into cells, as in Figure C.1.1. Similarly, an output fiber that is not yet wired to an input cell may appear as a half-edge out of a cell. Half-edge outputs may be unlabeled, as the meaning of the edge label (excitatory or inhibitory) is arbitrary until wired to another cell.

Individual McClulloch-Pitts neurons can compute many different functions on their inputs. For example, as shown in Figure C.1.1, basic Boolean operators such as *AND*, *OR*, *NOR*, as well as more complex functions like *Majority* and *Delay* can be expressed as a single neuron. Furthermore, *any* discrete function can be computed by a network of McClulloch-Pitts neurons. Illustrated in Figure C.2.1, GRNs made only of nominally "on" repressing transcription factors are equivalent to networks of McClulloch-Pitts neurons where the threshold for all cells $h = 0$ and all connections are inhibitory. As discussed in Section 3.4, although this class of networks may seem restrictive, it is sufficient to implement all Boolean logic (shown in Figure C.2.1), and perhaps surprisingly, it is sufficient to simulate any finite state machine.

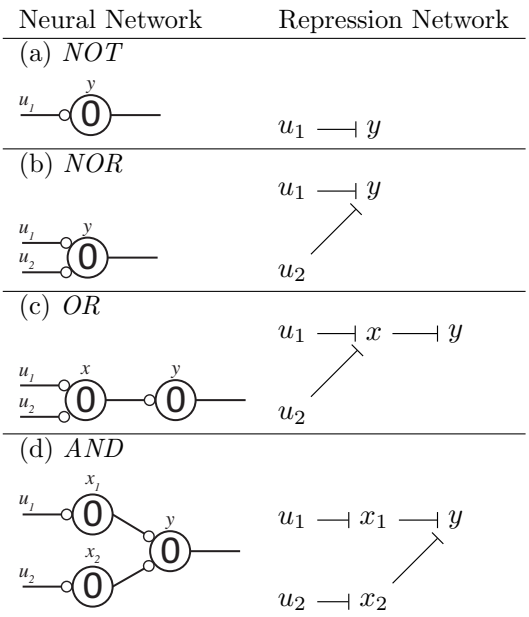


Figure C.2.1: Examples illustrating the equivalence of GRNs made only of nominally "on" repressing transcription factors, with neural networks where for each cell $h = 0$, and all connections are inhibitory. For clarity, input fibers (u_1, u_2) and neurons (x, x_1, x_2, y) have been named to correspond with molecular species in the GRN. In (a)-(d), each network computes the named Boolean logic operation on its inputs. (a)-(b) compute *NOT* and *NOR* in a single time step, where as (c)-(d) require two time steps to propagate the input to the output y for *AND* and *OR* functions. Here *AND* and *OR* networks are formed by composing the *NOT* and *NOR* networks. (a)-(b) are sufficient to simulate arbitrary finite state machines.

Appendix D

FUTURE DIRECTIONS

D.1 CRISPR-based Finite State Machines

As mentioned previously, the GRN representation of a FSM might be implemented by any number of biomolecular sensors and transcription regulation mechanisms. One promising technology for engineering novel transcription factors is the synthetic *Clustered Regularly Interspaced Short Palindromic Repeats* (CRISPR) system [101–103]. The CRISPR system can enable arbitrary "wirings" between transcription factors, a crucial aspect of engineering *de novo* gene regulation networks. The CRISPR mechanism is composed of two parts, the *single guide RNA* (sgRNA) and a *CRISPR-associated nuclease* (Cas9). The sgRNA contains a short guide sequence followed by short palindromic repeats that form a hairpin scaffold. The hairpin scaffold associates with Cas9, guiding the nuclease to a seuqnece of DNA targeted by the sgRNA guide sequence. [104, 105] A synthetic, catalytically inactive, mutation of Cas9 (dCas9) has been used successfully for steric repression in several prokaryotes and eukaryotes. [60] Recently, dCas9 has been further modified by fusing the KRAB repression domain as well as the VP64 and p65AD activation domains to the catalytically inactive protein. These fusion proteins have been shown to be effective chimeric transcription factors in E. coli, yeast, and human cells. [89, 106] Additionally, recent work in understanding the auxin signal-processing pathway in plants has resulted in a new small molecule sensing toolbox in yeast. [66] This toolbox, in conjunction with other small molecule systems [107], might be used as a source of modular and tunable input sensing in the FSM scheme. For example, biomolecular parts such as those illustrated in Figure D.1.1 that include transcription factors that are sensitive to auxin and β-estradiol might be sufficient to implement FSMs like the two-state machine discussed in Section 3.3 with a network like the one il-

lustrated in Figure D.1.2. Work such as this would represent an unprecedented top-down, specification-driven approach to designing complex cellular circuits.

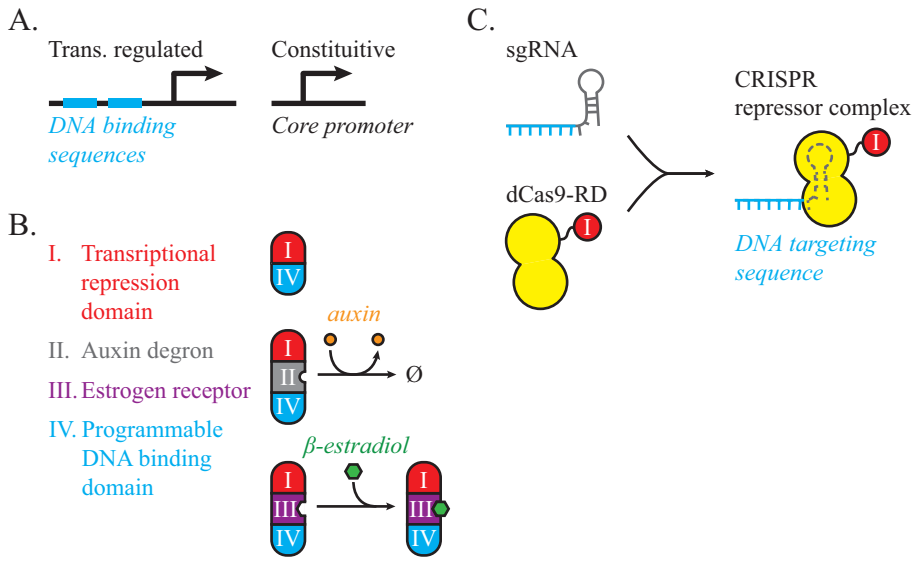


Figure D.1.1: Biomolecular components for realizing finite state machines: (A) Transcriptionally regulated and constitutive promoters. Transcriptionally regulated promoters have one or more specific DNA sequences designed to bind to repressing transcription factors. Binding sequences are illustrated as blue bars. Promoters are nominally "on" unless repressed by one or more transcription factors. (B) Transcription factors for sensing auxin and β-estradiol are made of four primary components: a transcriptional repression domain; a programmable DNA binding domain; and optional auxin degron or estrogen receptor. (C) CRISPR trascription factors are used to implement the finite state machine logic. The primary components of a CRISPR transcription factor are constitutively expressed nuclease deficient Cas9 (dCas9) fused to a repression domain (RD), and programmable single guide RNA (sgRNA) customized for specific DNA binding sequences. The scaffold section of the sgRNA binds to the dCas9-RD, enabling targeted repression of specific genes.

D.2 Streptobacilli Implementation of a Turing Tape Machine

The work presented in this thesis suggests that single cells can implement finite state machines, and that biomolecular engineering frameworks such as CRISPR-based transcription factors might offer a specific path for implementing novel state machines in cells. This

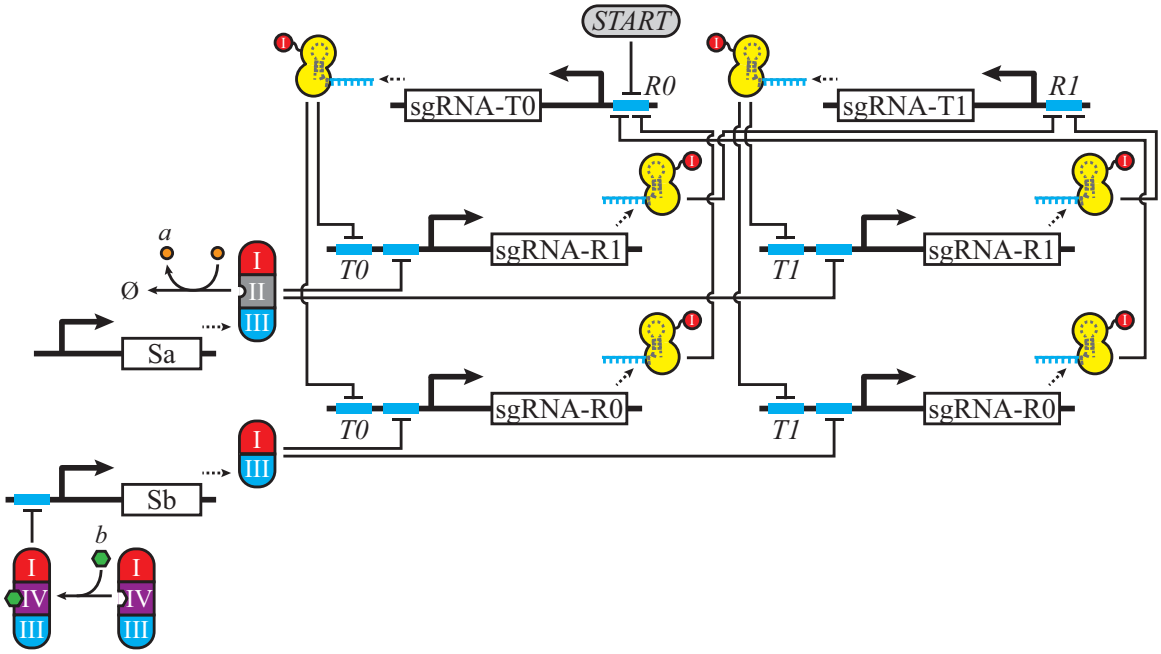


Figure D.1.2: Simple two-state machine described as a biomolecular realization of the gene regulatory network using parts from Figure D.1.1. Input symbols a and b are implemented as auxin and β-estradiol respectively. The gene network is arranged to mirror the layout of the gene regulatory network in (B). Sensor genes (Sa and Sb in the GRN) are implemented as auxin and β-estradiol sensors using transcription factors illustrated Figure D.1.1B. State and transition genes (Ri and T$q\sigma$ in the GRN) are implemented as CRISPR transcription factors formed by constitutive dCas9-RD and sgRNA sequences expressed under transcriptionally regulated promoters. For example, Ta0 in the GRN is implemented as sgRNA-R1 which is regulated by binding sites for sgRNA-T0 and transcription factor Sa. When sgRNA-R1 is in high concentration, a transcription factor is formed that binds to DNA binding sequence R1, repressing the expression of sgRNA-T1 (state gene R1 in the GRN).

framework offers a new way to relate to the computational power of observed patterns of gene expression in multicellular systems, as well as a step towards the implementation of more complex models of computation (i.e. a stack machine or Turing Machine) in a growing microcolony. Still many questions remain—How is data stored and read from from a microcolony? What are the considerations in terms of space-time complexity in designing and executing algorithms on multicelled systems? What are the limitations on computation imposed by cell division times, the number of genes and unique molecular species in a synthetic network, or the kinetics of small diffusing signal molecules?

The Church-Turing thesis hypothesizes that any general way to compute is equivalent to the Turing machine [29]. Illustrated in Figure D.2.1a, often a Turing machine is visualized as a *tape head* operating on an infinitely long *tape*. The tape is divided into discrete cells, any of which may store one of a finite number of states. The tape head implements a finite state machine that takes as input the state of the cell directly under the tape head, and can write a new state state to the cell, and move the tape one cell to the left or right. The tape serves as both the input and output to the Turing machine.

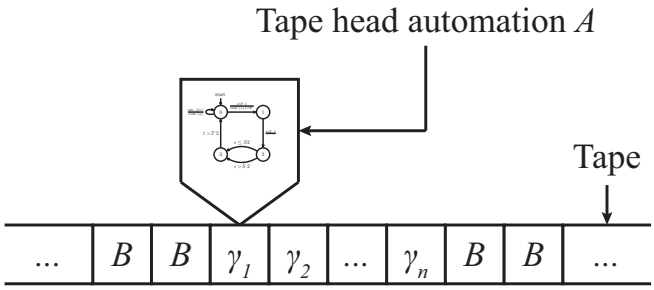
A Turing machine is specified by the tuple,

$$M \quad = \quad (Q, \Sigma, \Gamma, \delta, q_0, B, F). \tag{D.2.1}$$

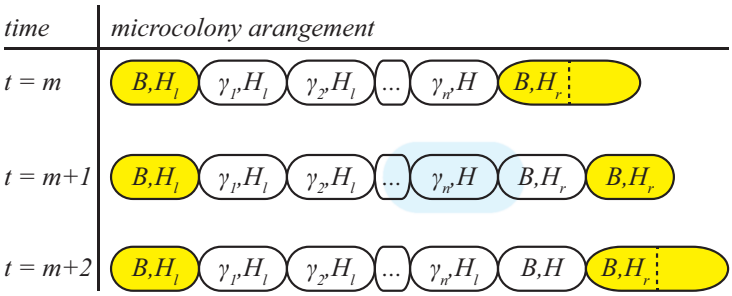
Where Γ is a finite set of tape symbols, and $B \in \Gamma$ is a special *blank* symbol. The input to a Turing machine is a finite length string of symbols $\gamma_1, \ldots, \gamma_n \in \Gamma$ placed on the tape. All other cells are marked blank with symbol B. The tape head automation is specified by the tuple

$$A \quad = \quad (Q, \Gamma, \delta, q_0, F). \tag{D.2.2}$$

Similar to the definition of a finite state automation in Section 3.1, Q is a finite set of states, Γ is a finite set of input symbols, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of accepting states. Unlike the previous definition, this transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ maps to a tuple (q, γ, d) where $q \in Q$ is the next state, $\gamma \in \Gamma$ is a symbol written to the



(a) Turing's tape machine



(b) Growing microcolony simulating a tape machine

Figure D.2.1: Turing's tape machine and a simulating microcolony. (a) Tape cells are represented by contiguous squares, and labeled with a tape symbols $B, \gamma_1, \ldots, \gamma_n \in \Gamma$. Initially the tape is seeded with the input $\gamma_1, \ldots, \gamma_n$, and the tape head begins over the first input cell. At each time step the tape head reads the current cell, updates the automation A, writes a new symbol from Γ to the tape, and moves the tape one cell to the left or right. (b) A growing line of cells simulate a Tape machine. Every cell runs a finite state automation, allowing a cell to behave as both the tape and the tape head. A microcolony edge detection program allows cells to tell if they are on the edge of the tape, and divide if the tape head is approaching. Cells shaded yellow are on the edge of the microcolony. At $t = m$, the cell labeled γ_n, H holds the tape head (denoted H), and is marked with the symbol γ_n. All other cells are labeled H_l or H_r to denote that they are left of the tape head or right of the tape head, respectively. At time $t = m + 1$, the cell marked as the tape head emits a signal (illustrated as a blue halo), to communicate to its immediate neighbors the current state at the tape head, and the intension to move the tape head one cell to the right. At time $t = m + 2$, the old tape head has relabeled itself H_l. The cell to the right of the old tape head, having the state H_r moves to the state H upon receiving the blue signal, becoming the new tape head. Simultaneously, the right edge cell, detecting the tape head approaching, begins dividing to guarantee the tape continues to extend beyond the tape head.

tape at the location of the tape head, $d \in \{L, R\}$ is the direction to move the tape head (L or R for "left" and "right" respectively). The tape head is positioned at one end of the cells. At each time step the tape head first scans the tape and changes state, then writes a tape symbol to the tape, then moves the tape left or right.

Informally, this machine might be simulated by a single cell growing into a line of cells, such as a string of cyanobacteria, or cells trapped in a narrow microfluidic chamber as illustrated in Figure D.2.1b. Intuitively, every cell could run the same automation $A' = (Q', \Gamma, \delta', q_0', F')$. A' is based on A in Equation D.2.2, but lifted to the state space $Q' = Q \times \Gamma \times \{L, R\} \times \{H, H_l, H_r\}$, where H, H_l, and H_r denote "tape head", "left of tape head", and "right of tape head" respectively. The state transition function $\delta' : Q' \times \Gamma \to Q'$, q_0', and F' are lifted as well. The finite state automation $A' = (Q', \Sigma, \delta', q_0, F')$ can be implemented with a repression network. Now, rather than moving the tape relative to the tape head, a cell can "pass" the tape head sub-state to its neighbor by emitting a local "move right" or "move left" signal as well as a signal communicating its current state $q \in Q'$. After signaling the tape head movement, the cell then updates its own tape head state from H to H_l or H_r. Since each cell of the tape is represented by a single cell microorganism, at any time the tape is of finite length. Thus the tape must grow to out pace the computation. A concurrent microcolony edge detection program allows edge cells to grow the tape in response to receiving the tape head, and dynamically allocate resources to the computation.

The difficult problem of programming colony morphology, such as stripe formation, may be naturally formalized as the problem of specifying a Turing machine that writes a desired pattern of tape symbols. Computational tools like the `gro` specification and simulation environment offer a compelling means to rapidly prototype, analyze, and refine models for programming cell morphology. This approach will provide useful data for this construction as a Boolean network, and possibly for more realistic models of gene transcription and translation.