2017

Devices for safety-critical molecular programmed systems

Samuel Jay Ellis
Iowa State University

Devices for safety-critical molecular programmed systems

by

Samuel Jay Ellis

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
James I. Lathrop, Co-major Professor
Robyn R. Lutz, Co-major Professor
Stephen B. Gilbert
Eric R. Henderson
Jack H. Lutz

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2017

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my Co-Major Professors, Dr. James I. Lathrop and Dr. Robyn R. Lutz. Their support and guidance enabled me to continue my research when I was unsure of which direction it should take. I thank them for their patience and understanding over my entire graduate career and especially while I wrote this thesis. I feel extremely lucky to have had them both as my Major Professors and mentors.

My thanks go to the members of the Laboratory for Molecular Programming (LAMP) in the Computer Science Department at Iowa State University for their friendship and for their informative and interesting discussions. I would like to thank Titus H. Klinge in particular for introducing me to this field of research and helping me get my foot in the door of the LAMP research group.

I would also like to thank my friends and family for their support while I pursued my degree. A special thank you goes to my mother, Cathy Ellis, and my brother, Matthew Ellis, for always being available when I needed to talk. They provided encouragement when I needed it most.

This research was supported in part by the National Science Foundation grants #1247051 and #1545028.

ABSTRACT

The behavior of matter at the molecular level can be programmed to create nanoscale molecular components that accomplish desired tasks. Many molecular components are developed with intended uses that are safety-critical, such as medical applications. Ensuring the correctness and fault tolerance of such devices is paramount. Techniques to develop robustly correct programs have been widely studied in software systems and many devices have been constructed to aid in the safe operation of systems. We seek to demonstrate the effectiveness of software and safety engineering techniques in the molecular programming domain.

In this thesis, we present the design of five new devices to aid in the development of safety-critical molecular programmed systems. We introduce a Runtime Fault Detection device (RFD) to robustly detect faults and initiate recovery actions in response to a failed system. We present the Concentration Monitor, a device that can detect changes, major and minor, in concentrations in real-time and demonstrate its utility. We also describe methods for constructing chemical reaction networks that can robustly simulate any combinational logic gate. Finally, we present two devices to log the state of a molecular program, where the first device logs a state upon receiving a request, and the second device ensures that the current state meets a defined validity property before allowing a log to be taken. All devices have been formally verified using model checking, simulations, or formal proof techniques. The methods used to construct and verify these devices can be adapted to the design of future molecular systems to assist in ensuring their correctness.

CHAPTER 1. INTRODUCTION

Molecular programming is a domain where matter, such as DNA strands or chemicals, is programmed to achieve a desired behavior. The beginnings of the field trace to Seeman in the 1980s [53], for his work on DNA crystals that self assembled from DNA strands. Interest in the field began growing when Winfree proved the Turing universality of the model [66]. Many advancements have appeared in the field in years since, from Boolean circuits [7, 57, 62], to DNA based nano-robots [6, 12], to devices that contain drugs for delivery to targeted cells [15, 67, 44].

Though DNA strands are a physical concrete medium, a large amount of research is performed on more abstract mediums, such as Chemical Reaction Networks (CRNs). CRNs are an abstract model of computation that emulates the behavior of many molecular programs. They have been widely studied since at least the 1940s [14]. In recent years the CRN model has begun to be viewed as a programming language for the development of molecular programs. The change is largely due to work by Soloveichik, Seelig, and Winfree, who showed that DNA strands can be used to implement any CRN [55]. Now, CRNs are widely used as an abstract programming language that can, through the use of automated tools [2], be compiled into a DNA strand displacement system for experimentation or deployment.

A system is safety-critical if a system failure can lead to loss of life, equipment, or significant amounts of money. Many intended applications of molecular devices are safety-critical in nature, such as using nano-robots to deliver drugs, therefore it is critical to ensure that devices not only achieve the desired behavior, but are also robust to any failures.

Software and safety engineering techniques have been used to achieve these goals for many years. Techniques such as goal oriented requirements engineering [64], a process by which informal system goals are turned into a formal specification, aid in the analysis of a system's intended use for possible faults or failures. Previously, goal oriented requirements engineering has been used to analyze a molecular system and find obstacles that would hinder correct operation [45, 46]. Formal methods, specifically model checking, have been shown to be useful in the study of programmed nanodevices [38].

In this thesis, we present results concerning the application of software engineering techniques towards the construction of molecular devices to ensure safety in molecular programs. We utilize requirements engineering techniques to develop a specification for the devices to ensure that they reflect the intended behavior. Using formal methods including simulation, model checking and formal proofs we verify the accuracy of our designs in terms of the specifications.

The capability to detect and recover from potential faults in a device is important in a molecular program. We introduce an extension of the Molecular Watchdog Timer (MWT) constructed from stochastic chemical reaction networks, a device that monitors the health of another system and issues an alarm if a fault is detected, in Chapter 2. Watchdog timers are widely used in software systems to monitor the health of critical components. We briefly review the initial version, originally presented in Samuel Ellis' Masters thesis [20], then discuss the development of subsequent iterations. We present a goal diagram, generated using goal oriented requirements engineering [64] techniques, for a MWT and a formal specification using Continuous Stochastic Logic (CSL) [1]. We then discuss the iterative tool assisted process used to verify the design of the device against the specification and the results of the verification.

Chapter 3 introduces a device called a **Concentration Monitor**, used to detect whether a concentration is above or below specified bounds. We present a set of requirements for the concentration monitor and use formal analysis to ensure that the design satisfies the

specification. The concentration monitor is designed to be robust with respect to a number of factors, discussed further in Section 3.1, including errors at initialization and during runtime. We provide an example application of the concentration monitor through the construction of a molecular watchdog timer for deterministic chemical reaction networks.

Logic circuits play a key role in the operation of computer systems, and it is imperative that their operation is both correct and robust. Utilizing the concentration monitor, we present a method for constructing arbitrary logic circuits out of CRNs in Chapter 4. We present the construction of a CRN based NAND gate and formally prove its correctness. We present a proof for the creation of arbitrary logic gates using the CRN based NAND gate and show that the operation of the composed circuit remains correct and robust.

Software systems often use logs, append only files that are written to over time, to study the behavior of a system, especially if the system encountered a fault. In Chapter 5, we present two devices to construct a log of a molecular system's state over time. The first device is capable of logging the state of its monitored system at any time by being given a predetermined input signal. The second device extends the first by only logging the state when it is determined to be valid, based on a predetermined validity property. Both devices provide a log which can be read by an external user or another molecular program. We show give a proof that both devices perform the desired behavior robustly and correctly. We then discuss an extension of the logging devices to create a history of stored states which can be used for checkpointing and rollback.

All the devices presented assist in the construction of robust molecular programs. They enable the detection of faults both at runtime and through later study. The devices can also support the recovery of a system from faults through initiating recovery actions and rolling back to a previous system state. In addition to the devices themselves, the processes used in their specification, design, and verification can all be applied to the construction of future molecular systems.

1.1 Related Work

Model checking as applied to molecular systems has gained interest in recent years. Kwiatkowska and Thachuk described their use of the probabilistic model checker PRISM to perform probabilistic verification of Chemical Reaction Networks (CRNs) for biological systems [38]. Their work showed the benefits of probabilistic model checking for molecular systems and informed our work for the Molecular Watchdog Timer (MWT) and the Runtime Fault Detection Device (RFD). Visual DSD, a design tool for DNA strand displacement [42], can generate a model of a DNA system for use in PRISM. PRISM has been used previously to model a variety of biological case studies, including DNA circuits and DNA nanorobotic walkers [12, 40]. PRISM also has been used widely to model probabilistic protocols, e.g., in distributed sensor networks, and stochastic multi-player games [37].

We utilized the Lotka-Volterra 3-phase oscillator to construct an example monitored system for use with the RFD. Ballarini, Mardare and Mura, and Ballarini and Guerriero performed analysis on the Lotka-Volterra 3-phase oscillator using PRISM and described both of the failures modes that our MWT design successfully detects [4, 3]. Our work differs in that we performed verification on an interface attached to the oscillator and its composition with the RFD.

Stochastic Petri net formalisms, as in the model checker SMART, have been used to model signal transduction in biological pathways [26]. SMART has been used to model other types of stochastic systems, e.g., in a NASA-funded study to evaluate an airport runway safety monitor protocol for false alarms [54]. A good overview of the formalisms used in population models, including the stochastic petri nets used in SMART, appears in [27].

Due to the scale of molecular systems, there is a disconnect between the size of a feasible model for automatic checking and the intended system. We are often forced to find ways to restrict the size of our models to perform model checking. Pavese, Barberman and Uchitel described how to develop partial explorations of a system model automatically [51]. Their technique has promise for use in molecular programs that we hope to explore. Since many

molecular programs deal with extremely large, if not infinite, state spaces, probabilistic model checking on partial system explorations might provide bounds on the reliability of a molecular system that is too large to model check.

The values used for parameters in stochastic models are usually derived from experimentation or are only partially known. Small differences between a models' parameter values and their counterparts in the real world can have a drastic effect on verification results. Meedeniya et al. generated reliability evaluations of a probabilistic model of an antilock brake system with uncertain parameters using Monte Carlo simulations [47]. Su, Chen, Feng, and Rosenblum extended previous work by Su and Rosenblum [59] on perturbations in model checking parameters in discrete-time Markov chains to allow model checking on time-bounded continuous time Markov chains (CTMCs) with imprecise values for transition rates [58]. The imprecision present in the reaction rates of molecular systems raises the importance of determining the effects of perturbations in parameter values.

The modeling of biological systems has seen many advances in the past few years. Yordanov et al. formalized and encoded DNA computing to allow use of Satisfiability Modulo Theories (SMT) [68]. Fisher, Harel and Henzinger performed computational modeling of biological systems as reactive systems [22]. Hetherington et al. and Sumner et al. utilized sub-models of a biological system to compose an advanced model [28, 60]. David et al. created translators to convert SimBiology models for biological systems into CTMCs for stochastic model checking or into Ordinary Differential Equations (ODEs) for simulation [13]. SimBiology is a package from MATLAB that handles biological systems [61].

Through the development of the MWT and RFD, our experiences made it clear that the requirements and architecture should co-evolve, this is consistent with the "Twin Peaks" idea described by Nuseibeh [49]. We made extensive use of the probabilistic model checker to check the alignment of goals with design alternatives. The obstacle analysis gave us a framework for selecting among alternative designs [64]. We encountered great difficulty correctly specifying the requirements for the MWT. We often discovered that our requirements needed to change

to reflect new understandings gained through the automated analysis of the formal models. Often, the constraints of the domain drove our revision of the requirements. Whalen et al., reported similar experiences developing requirements for large avionic systems, in which the requirements were just as likely to be right or wrong. [65]. Like us, they found that analyzing formal models was effective in discovering inconsistencies between environmental assumptions and the requirements they derived for their systems.

Oscillators play an important role in biology which has increased interest in their study. Hori and Murray, in a recent paper on synthetic biochemical oscillators, stated that, "The reliable engineering of oscillators is an important milestone towards robust synthesis of more complex dynamical circuits in synthetic biology" [30]. Fern et al. reported the use of timer circuits to precisely coordinate chemical events in vitro [21]. 3-phase oscillators seem to have been first reported in [39] and more recently [8, 9, 41]. The 2-phase Lotka-Volterra oscillator also has been studied in the context of DNA strand displacement in [56, 41]. As stated earlier, formal model checking was performed on the Lotka-Volterra 3-phase oscillator in [4, 3].

Different motifs for designing molecular circuits have been proposed. Hjelmfelt, Weinberger, and Ross proposed the use of chemical based logic gates to construct neural networks and Turing machines [29]. Ogihara and Ray showed that a DNA computer could be used to simulate unbounded fan-in boolean circuits [50]. Qian and Winfree used DNA to construct a four-bit square-root circuit [52] Ge, Zhong, Wen, You, and Zhang presented different approaches for using Karnaugh maps to create combinatorial logic gates [23]. They used simulations to compare the stability and robustness of the different constructions. Our gates differ by being robust with respect to perturbations in rate constants, initial values, measurement function, and input concentrations. Our gates also update their output in real time, with a programmed delay, to changes in the inputs.

Molecular programs communicate with each other through the presence and absence of molecular signals. Barish, Rothemund and Winfree in 2005 showed that DNA self-assemblies

could pass data by propagating binary information along a DNA tube by making copies of that pattern [5]. Hsiao, Hori, Rothemund, and Murray showed that the sensing and recording of sequences of chemical inputs across bacterial cell populations could be performed with temporal logic gates [31]. Our work on the logging devices differs in a few ways. First, our devices are robust with respect to a number of factors. Second, the valid only logging device is capable of determining if a state is valid or not before performing a log.

1.2 Background

Here we present some background information to aid in understanding the work in this thesis.

1.2.1 Stochastic Chemical Reaction Networks

Chemical reaction networks (CRNs) are an abstraction of molecular processes that are occurring in a well mixed solution. CRNs are a desirable abstract model that acts as a programming language for molecular programs. Stochastic CRNs (SCRNs) are a class of CRNs where each molecular species has a finite integer count for its population. SCRNs are ideal for modeling small scale systems, tens to a few hundreds of molecules, since the presence or absence of a single molecule affects the behavior of the system, such as the presence of a single viral genome within a living cell. The SCRN model dates back to at least the 1940s [14], but has been widely studied since [24, 63, 25].

An SCRN consists of a tuple $N = (\mathbf{S}, R)$, where $\mathbf{S}$ contains a set of species and R contains a set of reactions over the species in $\mathbf{S}$. Each species is given an abstract label (e.g. A) and has an integer molecular count denoting its population at a given point in time. We define the current state of an SCRN as an integer vector over the set of species $\mathbf{S}$. The set R contains a set of chemical reactions, each of which is a tuple $\rho = (r, p, k)$ where

r - an integer vector of reactants over the set $\mathbf{S}$.

p - an integer vector of products over the set $\mathbf{S}$.

k - a rate constant specifying the speed the reaction occurs.

A reaction ρ is usually written in the form

$$A_0 + \cdots + A_i \xrightarrow{k} B_0 + \cdots + B_j.$$

$A_0, \ldots, A_i$ are the reactants, which are both necessary for ρ to occur and are consumed by the reaction. $B_0, \ldots, B_j$ are the products, these are produced when ρ occurs. If a species appears in both the reactant vector and the product vector it is called a catalyst, which is necessary for the reaction to occur but is not consumed by the reaction.

For clarity, consider the following example. Given a CRN $N = (\mathbf{S}, R)$ with $\mathbf{S} = \{A, B, C\}$ and R containing only the reaction

$$\rho \equiv A + B \xrightarrow{k} 2C.$$

The reaction ρ has a reactant vector $(1, 1, 0)$, a product vector $(0, 0, 2)$ and a rate constant k. If an A molecule and B molecule are both present in the solution and interact with one another, they will be consumed to produce two molecules of C at a rate k.

SCRNs contain integer molecular counts, meaning their behavior can be described by a continuous-time Markov chain (CTMC)[14]. CTMCs are a useful model for our work as they are supported by the model checker PRISM [37]. A CTMC is a directed graph where nodes are states of the system and edges are transitions between the states. We assign each state in a CTMC to a possible state in the operation of the SCRN. We start with an initial state, a vector with the initial population counts. From the initial state, for each reaction, we create a transition to a new state where the reaction has been applied, i.e. the reactants subtracted and the products added to the current state. We continue this process recursively for each state until the set of possible states for the SCRN are included. We call this set of states the statespace of the CTMC. We assign a rate to each transition based on the reactants and the rate constant. For a given reaction $\rho = (r, p, k)$, we assign a rate as

$$\frac{k(A_0)(A_1)\ldots(A_i)}{volume^{(i-1)}},$$

where $A_0, \ldots, A_i$ represent their current population within the solution and are only the species with non-zero counts in the reactant vector of ρ. If a reaction is unary, i.e. only has one reactant, its rate is a special case and is defined as

$$k(A_0).$$

The rate determines how much continuous time passes while a transition occurs. The probability of a transition occurring is determined by its rate over the sum of the rates of all possible reactions in the current state.

Likewise, an SCRN can be modeled as a probabilistic Petri-net (PPN). PPNs are a useful model for our work as they are supported by the model checker SMART [10]. A PPN consists of a set of states which contain integer numbers of tokens, and a set of transitions which move tokens between states. We construct a single state for each species in $\mathbf{S}$. The current population of a species is specified by the number of tokens in its place. We also construct a transition for each reaction in R. We assign rates and probabilities to the transitions as above in CTMCs.

1.2.2 I/O Chemical Reaction Networks

Input/Output Chemical Reaction Networks (I/O CRNs) are a special class of CRNs, defined here under deterministic mass action kinetics. I/O CRNs were first defined in Titus Klinge's thesis [32], and have been used to study robust logic gates [17]. We adopt their syntax to discuss I/O CRNs. I/O CRNs are a useful model for dealing with signal and information passing between different CRNs. The basic structure of deterministic CRNs is the same as SCRNs. A CRN $N = (\mathbf{S}, R)$ contains a set of species $\mathbf{S}$ and a set of reactions R. Each reaction ρ in R is a tuple $\rho = (r, p, k)$, where the reactant vector r, product vector p, and rate constant k are defined as above in SCRNs. The primary difference is in the counts of the species. Where $\mathbf{S}$ is an integer vector in an SCRN, here $\mathbf{S}$ is a vector of real numbers. We call the current population value of a species its concentration.

An I/O CRN is a tuple $N = (U, R, S)$ where $U, S \subseteq \mathbf{S}$ are finite sets of species and no element that appears in U can appear in S, that is $U \cap S = \emptyset$, and R is a set of reactions over $U \cup S$. The set U is called the set of input species and the set S is the set of state species. Any element that appears in U may only be used catalytically, i.e. non-destructively, by any reactions in R.

In deterministic mass action kinetics an I/O CRNs state is a vector $x \in [0, \infty)^{|S|}$. We note the concentration of a species $Y \in S$ by $x(Y)$. An I/O CRN also has an input state, a vector $u \in [0, \infty)^{|U|}$, and a global state, a vector $(x, u) \in [0, \infty)^{|U \cup S|}$. A W-signal space is a set $C[W] = C([0, \infty), [0, \infty)^{|W|})$ with $C(X, Y)$ as the set of continuous functions from X to Y, for a given finite set $W \subseteq \mathbf{S}$. An I/O CRN has a context, defined as a tuple $c = (u, V, h)$. We call u the input function of the I/O CRN and define it as $u \in C[U]$. Informally, u is a vector denoting the concentrations of all species in the input set U. We call V the output species, defined as $V \subseteq S$, which specifies the state species used to produce an output from the I/O CRN. Finally, we call h the measurement function, defined as $h : [0, \infty)^{|U \cup S|} \to [0, \infty)^{|V|}$ which defines the behavior of the output species V based on a global state, a set of concentrations for the input and state species. We use C_N to reference the set of all contexts for a given I/O CRN N.

Each reaction ρ in an I/O CRN N has a rate dependent on the concentrations of all involved species. For a given global state $(x, u) \in [0, \infty)^{|U \cup S|}$, the rate of a reaction ρ is defined by the real value

$$rate_{x,u}(\rho) = k(\rho) \prod_{Y \in (U \cup S)} (x, u)(Y)^{r(\rho)(Y)}. \tag{1.1}$$

Here, $k(\rho)$ is the rate constant of reaction ρ and $(x, u)(Y)^{r(\rho)(Y)}$ denotes the concentration of a species Y raised to the power of its activity in the reaction. More simply, if a species Y is not involved in the reaction, its activity is 0, meaning its involvement in the rate is a 1. If the species Y is involved in the reaction, its activity is a 1, meaning its involvement in the rate is equivalent to its concentration. A species $Y \in S$ has a deterministic mass action function that determines how it will change over time, given the current global state. This

function is defined as

$$F_Y(x,u) = \sum_{\rho \in R} \Delta\rho(Y) \cdot \text{rate}_{x,u}(\rho). \tag{1.2}$$

where $\Delta\rho$ is the net effect of the reaction, subtracting the reactants and adding the products.

In deterministic mass action kinetics, the behavior of a CRN is modeled by a series of ordinary differential equations (ODEs). For a given species $Y \in S$ and a context of the I/O CRN N, the ODE for Y is

$$y'(t) = F_Y(x(t), u(t)), \tag{1.3}$$

for all $t \in [0, \infty)$. Informally, given a time t, the species Y will change based on the global state at time t. The ODE can be rewritten in the form

$$x'(t) = F(x(t), u(t)), \tag{1.4}$$

for all $t \in [0, \infty)$ by defining a function for each $Y \in S$, $F(x,u)(Y) = F_Y(x,u)$.

Given an initial state $x_0 \in [0, \infty)^{|S|}$, by the standard theory of ODEs, the system of ODEs has a unique solution $x(t)$. Using the measurement function, we can now define the output signal of an I/O CRN N as

$$N_{x_0,c}(t) = h(x(t)), \tag{1.5}$$

for all $t \in [0, \infty)$ for a given initial state and context. The output signal is the behavior of the defined output species V, that is, how V will change over time given an initial state and a context.

CHAPTER 2. FAULT DETECTION OF MOLECULAR SYSTEMS AT RUNTIME

When dealing with safety-critical systems, a failure can have disastrous consequences, leading to massive losses. Therefore, it is important that any fault is quickly detected so it can be properly handled before any losses occur. Introducing devices to both monitor for faults and make either another system or a user aware when they occur is a necessary step. One such device is a watchdog timer, a device which monitors a system for a fault and raises an alarm upon its occurrence [34, 43]. It does so through the use of a heartbeat. The monitored system provides a heartbeat at a regular interval, as long as the signal is being received, the monitored system has not encountered a failure. If a long enough period of time has occurred since the previous heartbeat, the watchdog timer assumes that the monitored system has failed and produces an alarm signal.

Watchdog timers have widely been used in order to detect faults and prevent failures within software systems. One such example is the Voyager spacecraft. The command computer would receive a heartbeat signal every two seconds from the attitude control computer upon completion of its self tests. In the event that a heartbeat was not received, the command computer would respond by activating the backup processor [48]. We present designs for devices that monitor the health of a molecular system and provide an alarm if a fault is detected. Each device is an iteration of a Molecular Watchdog Timer (MWT) constructed from Stochastic Chemical Reaction Networks (SCRNs).

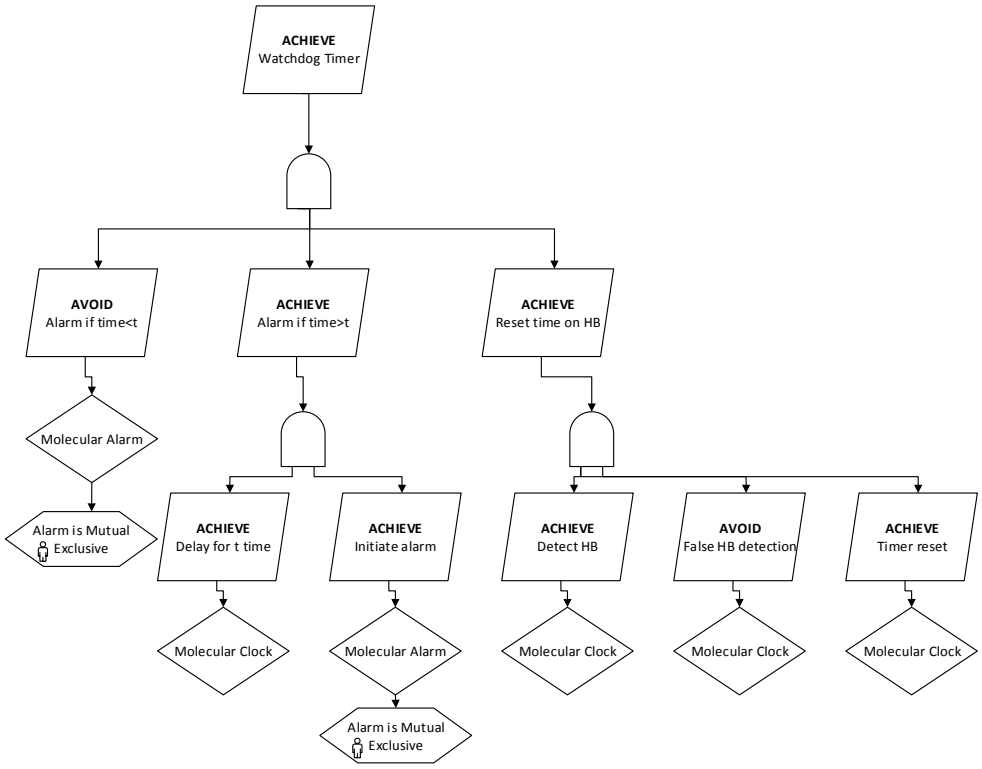

Figure 2.1 Original Goal Diagram for a Molecular Watchdog Timer.

2.1 Original Molecular Watchdog Timer

The work presented in this subsection (2.1) was previously presented in Samuel Ellis' master's thesis [20]. A brief review of it is included in this chapter because the remaining two sections of this chapter are extensions of it.

2.1.1 Requirements of the MWT

We constructed a goal diagram for the molecular watchdog timer. We defined the high level goal "*If the monitored system does not provide the appropriate signal within a specified time, then issue an alarm.*" We derived subgoals from the high level goal until we had only leaf goals.

LadderReactions	**Heartbeat Reset Reactions**
$L_r + U \xrightarrow{k} Y + U$	$Y + H \xrightarrow{k} L_1$
$\vdots$	$L_r + H \xrightarrow{k} L_1$
$L_3 + U \xrightarrow{k} L_4 + U$	$\vdots$
$L_2 + U \xrightarrow{k} L_3 + U$	$L_2 + H \xrightarrow{k} L_1$
$L_1 + U \xrightarrow{k} L_2 + U$	$L_1 + H \xrightarrow{k} L_1$

Figure 2.2 Original Absence Detector Reactions.

The leaf goals covered the tasks of correctly detecting if the heartbeats were present, resetting the timer if the heartbeat was present, and initiating an alarm if the heartbeat was absent for a sufficient length of time. To satisfy the leaf goals, we defined two agents, a molecular alarm and a molecular clock. The molecular alarm simply needed to be a chemical species. When the species is present, the alarm is considered "on", when it is absent, the alarm is "off." The molecular clock needed to be a device that could detect the heartbeat molecules and reset a timer based on their presence.

2.1.2 Design of the MWT

We tried a number of designs to satisfy the goals in the goal diagram. After encountering a number of obstacles, we settled on a design with two main components, an absence detector and a threshold filter. The absence detector consists of ladders, programmed cascades of reactions, that climb naturally over time and are reset by consuming heartbeat molecules. The CRN for the absence detector is shown in Figure 2.2. The absence detectors satisfy the goals of detecting the presence or absence of heartbeats and reacting accordingly.

The threshold filter reads the outputs of the absence detector to determine if a programmed threshold number of absence detector ladders have not found any heartbeat molecules, in which case it issues an alarm. The threshold filter CRN is shown in Figure 2.3.

Detection Reactions	**Threshold Reactions**
$T_n + Y \xrightarrow{k} D + Y$	$T_n + R \xrightarrow{k} T_1 + R$
$\vdots$	$\vdots$
$T_3 + Y \xrightarrow{k} T_4 + Y$	$T_3 + R \xrightarrow{k} T_1 + R$
$T_2 + Y \xrightarrow{k} T_3 + Y$	$T_2 + R \xrightarrow{k} T_1 + R$
$T_1 + Y \xrightarrow{k} T_2 + Y$	$T_1 + R \xrightarrow{k} T_1 + R$

Figure 2.3 Original Threshold Filter Reactions.

2.1.3 Verification of the MWT

We first discussed proofs for the MWT satisfying the goal diagram. Based on a set of domain properties, we reasoned about the MWTs behavior. We discussed how each leaf goal is satisfied by its agent. For example, the leaf goal Achieve[Delay for t time.] is satisfied by the absence detectors. The absence detectors have a programmable delay in their expected climb time. By programming the delay higher than t, the leaf goal is satisfied. We then showed that each set of subgoals satisfied their parent goals. In this way, our MWT design satisfied our goal diagram.

With no formal specification for each leaf goal, we could not perform model checking directly on the requirements. To show the correctness, we instead used the model checkers PRISM [37] and SMART [10, 11] to generate probabilities based on different constructions. The primary goal was to ensure that the probability of false positives and false negatives were both controllable and ideally very low. In terms of the MWT:

- **False Positive**: While heartbeats are present, the MWT issues an alarm.
- **False Negative**: While heartbeats are absent, the MWT does not issue an alarm.

We created models of the absence detector using both tools. We then defined a property to generate the probability of issuing an alarm before a specified time.

- PRISM Property:

$$P_{=?}\ [F_{=T}\ Y >= (0.8 * MAX_L)]$$

- SMART Property:

$$prob\_at(tk(Y) >= (0.8 * n), T)$$

We checked this property over different configurations of our models. We tested variations on number of heartbeats, size of heartbeats, the time heartbeats were inserted, and the length of the absence detectors.

We also tested variations of the threshold filter using a model in SMART. We wanted to show that the threshold filter had a low probability of initiating an alarm when the number of absence detectors that found no heartbeat molecules was below the threshold. We defined a SMART property to model this behavior.

$$probDatT : prob\_at(tk(D)! = 0, t) \tag{2.1}$$

To show the controllability of the threshold filter, we tested different lengths of threshold ladders in addition to the strength of the threshold to see their effect on this property.

2.2 Refining the MWT

The work presented in this subsection was previously presented at the 29^{th} IEEE/ACM International Conference on Automated Software Engineering (ASE 2014) and was performed in collaboration with Eric R. Henderson, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Divita Mathur, and Andrew S. Miner [16].

2.2.1 Single Use Watchdog Timer

We formalized the goal diagram from the original MWT to enable software based model checking of the design. We performed formal verification on the design and requirements of the molecular watchdog timer to ensure their accuracy using model checking and simulation.

2.2.2 Requirements Analysis

2.2.2.1 Goal Diagram

We specified the system goals for the MWT using an approach introduced in [45, 46] based on the KAOS [64] method developed by Axel van Lamsweerde. One of the primary advantages of goal-based approaches is in the refinement process. You begin with an informally written goal, a simple description of the behavior desired from the system. You then incrementally refine it into a formal specification. We constructed a goal diagram, a graph consisting of goals with AND/OR refinements, for the MWT. We began by defining a high level goal for the system, the desired behavior of the device. Then, using AND/OR refinements, we produced a set of subgoals whose satisfaction implies the satisfaction of their parent goals. We continue this process until each goal, called a leaf goal, can be achieved by a single agent of the system.

Each goal is specified as an ACHIEVE, AVOID, or MAINTAIN goal. An achieve goal is a behavior the system must meet to satisfy a requirement. An avoid goal is a behavior the system must not encounter to satisfy a requirement. Maintain goals are behaviors that must always be active during system operation.

We redefined the high level goal as "*the MWT shall issue an alarm if and only if no heartbeat signal is detected within a specified time bound*". This is reflected in the top level goal of the goal diagram, Figure 2.4, Achieve[Alarm iff no Heartbeat provided within t time].

We refined the high level goal into two main subgoals

- Initiating an alarm only when it is correct to do so.
- Correct alarm behavior.

The first task if further refined a few times and its leaf goals are assigned to the original components, the absence detector and the threshold filter. The second goal was refined into three leaf goals that are assigned to a new agent, the amplifier.

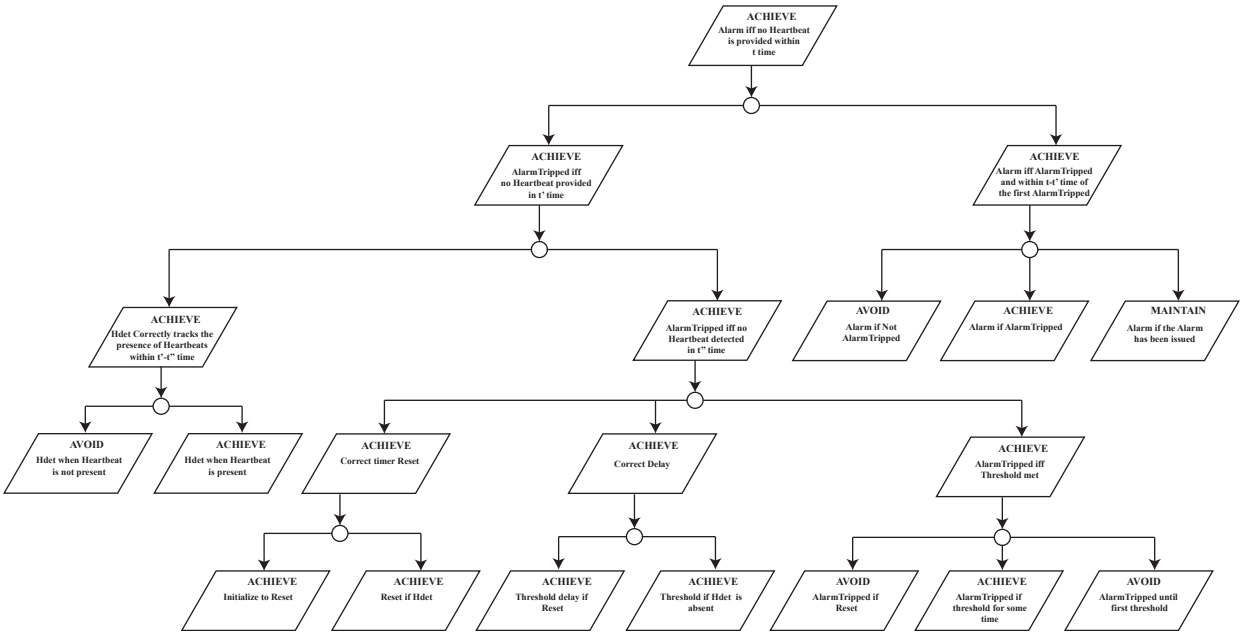

Figure 2.4 Goal diagram for the single use molecular watchdog timer.

2.2.2.2 Environmental Assumptions

To determine goal satisfaction, it is necessary to analyze the environment a system will be deployed in and determine any assumptions we must make about it. The MWT operates in a molecular environment, that is, a nanoscale environment with molecules floating around in a liquid solution. Some assumptions on the behavior of systems in molecular environments must be made in order to model their behavior. We utilize four environmental assumptions to ensure that the goal diagram is satisfied by our design.

1. Heartbeat presence is controlled entirely by the environment, meaning the MWT does not create any H molecules.

2. Heartbeats are "doses" of H molecules added into the solution. The size of the doses must fall within a specified range.

3. The only molecular species in the environment that interacts with the MWT is the heartbeat species H.

4. The MWT is operating in a well mixed solution, that is, molecules are present evenly throughout the solution, instead of being bunched together in small parts.

The size range of the heartbeat doses is determined based on other parameters within the goals, such as the population size of each species in the model.

2.2.2.3 Goal Formalization

Each goal is an informal description of a desired system behavior. However, in order to design a system and perform formal verification, it is necessary to have a formal specification for each goal. We use continuous stochastic logic (CSL) [1] to specify each goal. We chose to use CSL because it handles continuous time Markov chains (CTMCs) which define the behavior of our CRN model. It is also available in two model checking tools that we use, the Probabilistic Symbolic Model Checker (PRISM) [37] and the Stochastic Model-checking Analyzer for Reliability and Timing (SMART) [10, 11]. We include the formal specification in CSL of each goal in Figure 2.4. These goals are ordered in breadth first order following the order of the figure.

Our specification has four user defined parameters so that a construction can be made to meet nearly any needs. The parameters the user must provide are:

u - Alarm should not be issued within u-time of the last heartbeat

v - Alarm should be issued at least by v-time of the last heartbeat

ϵ - Probability of error allowed by the u-delay

δ - Probability of error allowed by the v-delay

Below is the top-level goal of our diagram. Given parameters, u, v, ϵ, δ, the following constraints are placed on the top-level goal:

$$(1 - \epsilon_1)(1 - \epsilon_2) = (1 - \epsilon)$$

$$(1 - \delta_1)(1 - \delta_2) = (1 - \delta)$$

ACHIEVE: Alarm iff no HB is provided within t-time

$$\begin{aligned}
&\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2}\Box_{\leq u}\neg A_{trip}\right)\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}\Diamond_{\leq v-w_a}\left(A_{trip} \vee H_{pres}\right)\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Alarm \implies \boldsymbol{\mathcal{P}}_{\geq 1}\Box Alarm\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\left(\neg Alarm\ \boldsymbol{\mathcal{W}}\ A_{trip}\right) && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[A_{trip} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_2}\Diamond_{\leq w_a} Alarm\right]
\end{aligned}$$

Below are the direct subgoals of the high-level goal. The subgoals are simply partitions of the six individual properties of the parent-goal, and their equivalence is clear.

ACHIEVE: AlarmTripped iff no HB provided in t' time

$$\begin{aligned}
&\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2}\Box_{\leq u}\neg A_{trip}\right)\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}\Diamond_{\leq v-w_a}\left(A_{trip} \vee H_{pres}\right)\right]
\end{aligned}$$

ACHIEVE: Alarm iff AlarmTripped and within $t - t'$ time of the first AlarmTripped

$$\begin{aligned}
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Alarm \implies \boldsymbol{\mathcal{P}}_{\geq 1}\Box Alarm\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\left(\neg Alarm\ \boldsymbol{\mathcal{W}}\ A_{trip}\right) && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[A_{trip} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_2}\Diamond_{\leq w_a} Alarm\right]
\end{aligned}$$

Below are the subgoals of "AlarmTripped iff no HB provided in t' time". This refinement abstracts the role of detecting the presence of a heartbeat from the rest of the goals and imposes the following constraint on the introduced variables:

- $1 - \epsilon_1 \leq (1 - \alpha)(1 - \beta)(1 - \epsilon_1')$
- $1 - \epsilon_2 \leq 1 - \epsilon_2'$
- $w_h \leq g$

- $1 - \delta_1 \leq (1 - \alpha)(1 - \beta)(1 - \delta_1')$

ACHIEVE: Heartbeat Detected correctly tracks the presence of Heartbeats within $t' - t''$ time

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}H_{det}\right] \quad \wedge$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right]$$

ACHIEVE: AlarmTripped iff no Heartbeat detected

$$\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} \quad \wedge$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1'}\Diamond_{\leq g-w_h}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2'}\Box_{\leq u}\neg A_{trip}\right)\right] \quad \wedge$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1'}\Diamond_{\leq v-w_a-w_h}\left(A_{trip} \vee H_{det}\right)\right]$$

AVOID: Alarm if not AlarmTripped

$$\boldsymbol{\mathcal{P}}_{\geq 1}\left(\neg Alarm\ \boldsymbol{\mathcal{W}}\ A_{trip}\right)$$

ACHIEVE: Alarm if AlarmTripped

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[A_{trip} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_2}\Diamond_{\leq w_a}Alarm\right]$$

MAINTAIN: Alarm if AlarmTripped

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Alarm \implies \boldsymbol{\mathcal{P}}_{\geq 1}\Box Alarm\right]$$

AVOID: Heartbeat Detected if Heartbeat not present

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right]$$

ACHIEVE: Heartbeat Detected if Heartbeat present

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}H_{det}\right]$$

Below are the subgoals of "AlarmTripped iff no Heartbeat detected". This refinement specifically enabled us to verify the components of the MWT individually by isolating their roles. The introduced variables are constrained in the following way:

$$(1-\gamma_1)(1-\gamma_2) = (1-\epsilon)$$
$$(1-\gamma_1)(1-\lambda_1)(1-\lambda_2)(1-\lambda_3)(1-\lambda_4) = (1-\epsilon'_1)(1-\epsilon'_2)$$
$$(1-\eta_1)(1-\eta_2)(1-\eta_3) = (1-\delta'_1)$$
$$w_{r_{off}} \leq g - w_h$$

ACHIEVE: Correct timer reset

$$Reset \qquad \wedge$$
$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_1}\Diamond_{\leq w_{r_{on}}} Reset\right]$$

ACHIEVE: Correct delay

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Reset \implies \boldsymbol{\mathcal{P}}_{\geq 1-\gamma_1}\Box_{\leq u} Th_L\right] \qquad \wedge$$
$$Th_L \implies \neg Th_H$$
$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_1}\Diamond_{v-w_a-2*w_h-w_{th}}\boldsymbol{\mathcal{P}}_{\geq 1-\eta_2}$$
$$(Th_H \,\boldsymbol{\mathcal{W}}\, \boldsymbol{\mathcal{P}}_{\geq 1-\eta_3}\Diamond_{\leq wh} H_{det})]$$

ACHIEVE: AlarmTripped iff Threshold met

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_L \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_2}\Diamond_{\leq w_{r_{off}}}\boldsymbol{\mathcal{P}}_{\geq 1-\lambda_3}\Box_{\leq u}\neg A_{trip}\right] \qquad \wedge$$
$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_H \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}}(A_{trip} \vee \neg Th_H)\right] \qquad \wedge$$
$$\boldsymbol{\mathcal{P}}_{\geq 1-\gamma_2}(\neg A_{trip}\,\boldsymbol{\mathcal{W}}\,\neg Th_L)$$

ACHIEVE: Initialize to Reset

$$Reset$$

ACHIEVE: Reset if Heartbeat detected

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_1}\Diamond_{\leq w_{r_{on}}} Reset\right]$$

ACHIEVE: Threshold delay if Reset

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Reset \implies \boldsymbol{\mathcal{P}}_{\geq 1-\gamma_1}\Box_{\leq u} Th_L\right]$$

ACHIEVE: Threshold if Heartbeat Detected is absent

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_1}\Diamond_{v-w_a-2w_h-w_{th}}\boldsymbol{\mathcal{P}}_{\geq 1-\eta_2}$$
$$(Th_H \,\boldsymbol{\mathcal{W}}\, \boldsymbol{\mathcal{P}}_{\geq 1-\eta_3}\Diamond_{\leq wh} H_{det})]$$

AVOID: AlarmTripped if reset

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_L \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_2}\Diamond_{\leq w_{r_{off}}}\boldsymbol{\mathcal{P}}_{\geq 1-\lambda_3}\Box_{\leq u}\neg A_{trip}\right]$$

ACHIEVE: AlarmTripped if threshold for some time

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_H \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}}\left(A_{trip} \vee \neg Th_H\right)\right]$$

AVOID: AlarmTripped until first threshold

$$\boldsymbol{\mathcal{P}}_{\geq 1-\gamma_2}\left(\neg A_{trip}\,\boldsymbol{\mathcal{W}}\,\neg Th_L\right)$$

2.2.2.4 Goal Model Implication Proofs

Here we present the formal proofs that the satisfaction of any goal is implied by the satisfaction of its subgoals.

Lemma 2.2.1. *The high-level goal is implied by its subgoals where*

$$\begin{aligned}
&\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2}\Box_{\leq u}\neg A_{trip}\right)\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}\Diamond_{\leq v-w_a}\left(A_{trip} \vee H_{pres}\right)\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Alarm \implies \boldsymbol{\mathcal{P}}_{\geq 1}\Box Alarm\right] && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\left(\neg Alarm\,\boldsymbol{\mathcal{W}}\,A_{trip}\right) && \wedge\\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[A_{trip} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_2}\Diamond_{\leq w_a} Alarm\right]
\end{aligned}$$

is the high-level goal specification, and below are the subgoals:

Subgoal 1:

$$\begin{aligned}
&\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} && \wedge \\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2}\Box_{\leq u}\neg A_{trip}\right)\right] && \wedge \\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}\Diamond_{\leq v-w_a}\left(A_{trip} \vee H_{pres}\right)\right]
\end{aligned}$$

Subgoal 2:

$$\begin{aligned}
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Alarm \implies \boldsymbol{\mathcal{P}}_{\geq 1}\Box Alarm\right] && \wedge \\
&\boldsymbol{\mathcal{P}}_{\geq 1}\left(\neg Alarm\ \boldsymbol{\mathcal{W}}\ A_{trip}\right) && \wedge \\
&\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[A_{trip} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_2}\Diamond_{\leq w_a} Alarm\right]
\end{aligned}$$

Proof. Note that the first subgoal contains the first three statements of the parent goal and the second subgoal contains the last three. These six statements trivially compose the high-level goal if both are satisfied. □

Lemma 2.2.2. *The children of "**ACHIEVE:** AlarmTripped iff no HB provided in t' time" imply their parent where the parent specification is:*

$$\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} \quad \wedge \tag{2.2}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2}\Box_{\leq u}\neg A_{trip}\right)\right] \quad \wedge \tag{2.3}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}\Diamond_{\leq v-w_a}\left(A_{trip} \vee H_{pres}\right)\right] \tag{2.4}$$

and the subchildren are:

Subgoal 1:

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha} H_{det}\right] \quad \wedge \tag{2.5}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right] \tag{2.6}$$

Subgoal 2:

$$\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} \qquad \wedge \qquad (2.7)$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1'}\Diamond_{\leq g-w_h}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2'}\Box_{\leq u}\neg A_{trip}\right)\right] \qquad \wedge \qquad (2.8)$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1'}\Diamond_{\leq v-w_a-w_h}\left(A_{trip} \vee H_{det}\right)\right] \qquad (2.9)$$

Proof. Assume that equations 2.2-2.4 true. We will now show that these five equations are sufficient to prove equations 2.2-2.4 each individually.

1. Equation (2.7) trivially implies (2.2).

2. In order to prove the implication in (2.3) holds, we assume that the boolean variable H_{pres} is true. By (2.5), with probability $(1-\beta)(1-\alpha)$, within w_h time, H_{det} will be true. When H_{det} is true, by (2.8), with probability $(1-\epsilon_1')(1-\epsilon_2')$, within $g-w_h$ time, $\neg A_{trip}$ becomes true. Thus, worst-case, with probability $(1-\alpha)(1-\beta)(1-\epsilon_1')(1-\epsilon_2')$, within g time, $\neg A_{trip}$ becomes true.

   Therefore, this implies the (2.3) if we enforce the constraints:

   - $1-\epsilon_1 \leq (1-\alpha)(1-\beta)(1-\epsilon_1')$
   - $1-\epsilon_2 \leq 1-\epsilon_2'$

3. Assume $\neg H_{pres}$ is true. By (2.6), with probability $(1-\beta)(1-\alpha)$, within w_h time, $\neg H_{det}$ will be true until H_{pres} is true. Once $\neg H_{det}$ is true, by (2.9), with probability $(1-\delta_1')$, within $v - w_a -_w h$ time, we will either A_{trip} or H_{det}. Here we have two cases:

   Case 1: If *Atrip* happens within the appropriate time from $\neg H_{det}$ being true, then with probability $(1-\alpha)(1-\beta)(1-\delta_1')$, we will A_{trip} within $v-w_a$ time. This satisfies (2.4).

   Case 2: If *Atrip* does not happen within the appropriate time from H_{det} being true, then similarly, with probability $(1-\alpha)(1-\beta)(1-\delta_1')$, within $v-w_a$ time, H_{det} will become true. If H_{det} became true, then by (2.6) it must have been because H_{pres}

became true. That means that H_{pres} became true in at most $v - w_a$ time and thus satisfies (2.4).

Therefore, the subchildren imply the parent if we enforce the constraints:

- $w_h \leq g$
- $1 - \epsilon_1 \leq (1 - \alpha)(1 - \beta)(1 - \epsilon_1')$
- $1 - \epsilon_2 \leq 1 - \epsilon_2'$
- $1 - \delta_1 \leq (1 - \alpha)(1 - \beta)(1 - \delta_1')$

□

Lemma 2.2.3. *The children of "**ACHIEVE:** Heartbeat Detected correctly tracks the presence of Heartbeats within $t' - t''$ time" imply their parent where the parent specification is:*

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}H_{det}\right] \quad \wedge$$
$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right]$$

and the specification for the subgoals are:

Subgoal 1:

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right]$$

Subgoal 2:

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}H_{det}\right]$$

Proof. It is clear that the children compose the parent and are equivalent. □

Lemma 2.2.4. *The children of "**ACHIEVE:** AlarmTripped iff no Heartbeat detected" imply their parent where the parent specification is:*

$$\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg A_{trip} \quad \wedge \tag{2.10}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1'}\Diamond_{\leq g-w_h}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2'}\Box_{\leq u}\neg A_{trip}\right)\right] \quad \wedge \tag{2.11}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1'}\Diamond_{\leq v-w_a-w_h}\left(A_{trip}\vee H_{det}\right)\right] \tag{2.12}$$

and the specification of the children are:

Subgoal 1:

$$Reset \quad \wedge \tag{2.13}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_1}\Diamond_{\leq w_{on}} Reset\right] \tag{2.14}$$

Subgoal 2:

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Reset \implies \boldsymbol{\mathcal{P}}_{\geq 1-\gamma_1}\Box_{\leq u} Th_L\right] \quad \wedge \tag{2.15}$$

$$\begin{aligned}\boldsymbol{\mathcal{P}}_{\geq 1}\Box[\neg H_{det} \implies & \boldsymbol{\mathcal{P}}_{\geq 1-\eta_1}\Diamond_{v-w_a-2w_h-w_{th}} \\ & \boldsymbol{\mathcal{P}}_{\geq 1-\eta_2}\left(Th_H\ \boldsymbol{\mathcal{W}}\ \boldsymbol{\mathcal{P}}_{\geq 1-\eta_3}\Diamond_{\leq w_h} H_{det}\right)]\end{aligned} \tag{2.16}$$

Subgoal 3:

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_L \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_2}\Diamond_{\leq w_{off}}\boldsymbol{\mathcal{P}}_{\geq 1-\lambda_3}\Box_{\leq u}\neg A_{trip}\right] \quad \wedge \tag{2.17}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_H \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}}\left(A_{trip}\ \vee\ \neg Th_H\right)\right] \quad \wedge \tag{2.18}$$

$$\boldsymbol{\mathcal{P}}_{\geq 1-\gamma_2}\left(\neg A_{trip}\ \boldsymbol{\mathcal{W}}\ \neg Th_L\right) \tag{2.19}$$

Proof. Assume the truth of equations 2.13-2.19. It suffices to show that equations 2.10-2.12 are true.

1. By (2.13), *Reset* is true. By (2.15), with probability $1-\gamma_1$, Th_L will be true for u time. By (2.19), $\neg A_{trip}$ will not be true until $\neg Th_L$ is true with probability $1-\gamma_2$. implying (2.10) if the following constraint is met:

   - $1-\epsilon \leq (1-\gamma_1)(1-\gamma_2)$

2. Assume that H_{det} is true. By (2.14), with probability $1-\lambda_1$, within w_{on} time, *Reset* will be true. By (2.15), with probability $1-\gamma_1$, Th_L will be true for u time. By (2.18), with probability $1-\lambda_2$, within w_{off} time, with probability $1-\lambda_3$, we will $\neg A_{trip}$ for u time. This implies (2.11) if we enforce the following constraints:

   - $(1-\epsilon_1')(1-\epsilon_2') \leq (1-\lambda_1)(1-\gamma_1)(1-\lambda_2)(1-\lambda_3)$
   - $g - w_h \geq w_{on} + w_{off}$

3. Assume $\neg H_{det}$ is true. By (2.16), with at least $(1-\eta_1)(1-\eta_2)(1-\eta_3)$ probability, within $v - w_a - 2w_h - w_{th}$ time, a path will enter a position that satisfies $Th_H \ \boldsymbol{\mathcal{W}} \ \Diamond_{w_h} H_{det}$. Thus we have two cases:

   Case 1: Once Th_H becomes true, before w_{th} time passes, $\Diamond_{w_h} H_{det}$ becomes true. Therefore, in at most $v - w_a - w_h$ time, we receive an H_{det} and thus satisfy (2.12).

   Case 2: Once Th_H becomes true, it stays true for at least w_{th} time. Then by (2.18), with probability $1-\gamma_2$, within w_{th} time of Th_H becoming true, we will A_{trip} or $\neg Th_H$. Since we know Th_H is true for at least w_{th} time, we know we must A_{trip} by w_{th} time. Therefore, we have an A_{trip} in no later than $w - w_a - w_h$ time and satisfy (2.12).

   The above cases only hold true if we enforce the following constraint:

   - $1-\delta_2 \leq (1-\eta_1)(1-\eta_2)(1-\eta_3)(1-\gamma_2)$

Because each of equations 2.10-2.12 hold true with our assumptions, it is clear our subgoals imply the parent. □

Lemma 2.2.5. *All parent goals of leaves are implied by their children.*

Proof. All leave goals are broken down by conjunction and trivially imply their parents. □

Theorem 2.2.1. All the leaf goals imply the high level goal of our goal diagram.

Proof. By all of the lemmas proven above, the leaves successfully imply the high level goal. □

2.2.3 Design

We utilized 3 agents to satisfy the goal diagram for the MWT. The absence detector and threshold filter were both included in the original MWT. We added a new component, called an amplifier, to handle the alarm signal from the threshold filter.

2.2.3.1 Absence Detector

The absence detector monitors the presence and absence of heartbeat signals, in other words, they detect when a fault has occurred. The absence detector consists of a number of ladders and acts as a timer between heartbeats. Each ladder in the absence detector notifies the threshold filter when they have not found an H molecule for a sufficiently long time. If an H molecule is detected by a ladder, it resets its timer to zero and begins counting again.

There are no changes to this device from the original MWT design and the CRN is shown in Figure 2.2. L_i molecules interact with U molecules, which have a constant supply, to move one step forward along the cascade. When an L_i molecule interacts with an H molecule, it consumes it to reset back to L_0. Intuitively, if the number of H molecules is low, the likelihood of an L_i interacting with an H is low, meaning it will proceed through the cascade until a Y is released. If the number of H molecules is high, the probability of an L_i interacting with enough H molecules to reach Y is very low. The absence detector is assigned responsibility for the leaf goals *Avoid [Hdet when Heartbeat is not present]*, *Achieve [Hdet when Heartbeat is present]*, *Achieve [Initialize to Reset]*, *Achieve [Reset if Hdet]*, *Achieve [Threshold delay if Reset]*, and *Achieve [Threshold if Hdet is absent]*.

2.2.3.2 Threshold Filter

The threshold filter monitors the absence detector to determine the correctness of any fault detection, this prevents false positives. The threshold filter consists of a ladder that acts as a barrier to alarming. The ladder molecules interact with Y molecules, released by the absence detector, to move forward through the cascade. The ladder interacts with R molecules, which have a constant supply, to reset to the bottom rung.

We modified the threshold filter from the original MWT. It consists of the following reactions.

$$T_i + Y \rightarrow T_{i+1} + Y$$

$$T_k + Y \rightarrow T_k + Y + D$$

$$T_i + R \rightarrow T_1 + R$$

Intuitively, if the number of Y molecules is sufficiently higher than the number of R molecules, there is a high probability of the threshold filter releasing a D molecule. However, if the number of Y molecules is low, the R molecules hold the cascades in the lower rungs.

The change from the original design is that D molecules are released by the threshold filter catalytically. This follows from the goals that require the threshold be reset, for example *Avoid [AlarmTripped if reset]*. This goal requires the AlarmTripped signal be turned off when the timers are reset. AlarmTripped specifies that a D molecule has been released. In the original design, if a reset occurred after a D molecule was released this goal would fail. In the new design, if a D molecule has been released, it will quickly be consumed by the amplifier, discussed in the next section, and AlarmTripped will be turned off. However, in line with the MWT requirements, once the alarm has been tripped, the alarm will go off. The threshold filter is assigned responsibility for the leaf goals *Avoid [AlarmTripped if Reset]*, *Achieve [AlarmTripped if threshold for some time]*, and *Avoid [AlarmTripped until first threshold]*.

2.2.3.3 Amplifier

The amplifier component is responsible for detecting that an alarm has been tripped and translating it into a visible alarm. The amplifier monitors for any D molecules that appear in the solution. Once it encounters at least one D molecule, it begins an irreversible chain reaction that floods the solution with up to a specified amount of alarm molecules Al. The amplifier contains the following reactions.

$$D + Ca \rightarrow Al + Ca$$

$$Al + Da \rightarrow 2Al$$

At initialization, we begin with a number of Ca molecules, these are the catalyst that search for D molecules to initiate an alarm, and a specified number, p, of Da molecules, which denotes how many Al molecules the amplifier must create. When a D molecule is introduced into the solution, it interacts with a Ca molecule to produce an Al molecule. The Al molecules are used catalytically to produce more Al molecules until their count is at least equal to p. The amplifier was assigned responsibility for the leaf goals *Avoid [Alarm if Not AlarmTripped]*, *Achieve [Alarm if AlarmTripped]*, and *Maintain [Alarm if the Alarm has been issued]*.

2.2.4 Tool Assisted Verification

In many of the intended uses of the MWT, an absence of a heartbeat indicates that the system is in an unsafe state. The safety-critical nature of such systems requires the MWT to be robust, reliable, and correct. It is necessary to perform verification over the design and system requirements for the MWT. Due to the complexity of the MWT and the inherently probabilistic nature of the domain, manual verification is complex, making software verification tools a desirable alternative.

Given the CRN design of each of the three components, we constructed a model of each component using the PRISM model checker [37], which has been used to perform verification

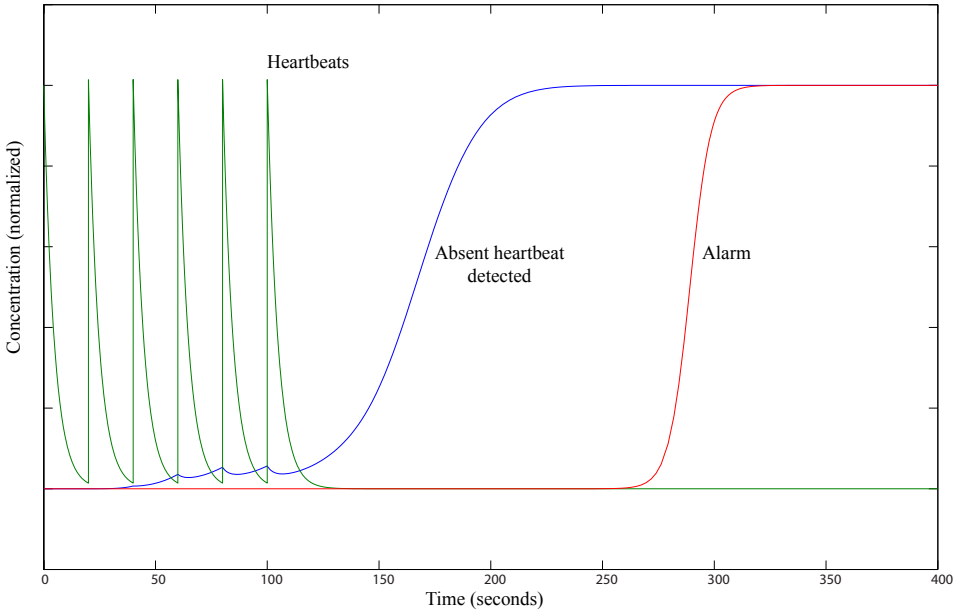

Figure 2.5 MWT ODE Simulation using MATLAB's SimBiology package.

on molecular systems previously [35, 38]. We utilized the formal CSL specification of each leaf goal as a property to check against each model. To fill in the internal parameters of the goal specifications, we used an incremental development process. We began with simulations of the model to find likely ranges for each parameter. Figure 2.5 shows a ODE simulation of the MWT design using the MATLAB package SimBiology. Then, using PRISM's capability to test a range of parameters, we performed incremental model checking on the components to narrow down the values for each parameter. The end result was, for each component, a model that specified all parameters for the goal diagram and that satisfied each of its assigned leaf goals.

2.3 Runtime Fault Detection Device

The work presented in this section was performed in collaboration with Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, and Andrew S. Miner and appears in [18]. Our goal was to extend the capabilities of the Molecular Watchdog Timer (MWT). Previously, its uses were limited to the detection of a single fault, which it would respond to by initiating an alarm. The first problem with the MWT is that it can only be used a single time. The amplifier component can only be used a single time, since the species involved in the reactions have limited counts and are consumed in their use. The second major problem is that it only produces an alarm, which is then present in the solution forever. The MWT would be a far more useful device if it was able to initiate direct response to a system failure.

Here we present the Runtime Fault Detection (RFD) device, an advancement on the MWT that maintains a high output signal whenever failure is detected and no output signal while failures are not detected. The RFD responds in real-time, with a controllable delay, to failures in the monitored system and provides a recovery signal, called henceforth an alarm, to either trigger a programmed response or as an externally visible alarm. Recovery actions can include any action that helps the system recover from its failure. For example, the alarm could lead to the correction of the fault within the monitored system or activate a backup system to replace the failed system. We discuss the requirements and specification of the RFD as a modification of the MWT. We show its correctness using formal methods. We present an example case of the RFD embedded with a molecular oscillator as a monitored system.

2.3.1 Requirements Analysis for the RFD

In this section, we discuss the requirements for the RFD. We describe the informal goal diagram for the RFD. We then describe the environmental assumptions used to satisfy the goals. Each goal is then given a formal specification in Continuous Stochastic Logic (CSL)

to have an unambiguous definition. We show that each leaf goal's formal specification leads to the satisfaction of their parent goals, allowing the entire goal diagram to be satisfied.

2.3.1.1 Goal Diagram for the RFD

There were two primary goals for the RFD, reusability and flexibility in application. Instead of always producing a single use alarm, and therefore limiting possible uses for the MWT, the RFD produces an alarm for use by another component. The RFD allows a client to create a component that reads the presence and absence of the alarm and takes a desired action. One such component could be a single use alarm, which detects the presence of the RFD alarm and initiates an irreversible alarm, however the RFD requirements do not include this third component. The responsibilities of the RFD lie in producing an alarm as long as no heartbeats have been detected for a sufficient time and producing no alarm while hearbeats are detected.

The high level goal of the RFD is *Achieve[Alarm iff no Heartbeat provided within t time]*, seen at the top of the goal diagram in Figure 2.6. The goals are refined into subgoals until leaf goals are reached. Notice that the goal diagram is a pruned version of the MWT goal diagram. We removed the subtree containing the single use alarm goals and propogated the change up the tree. This left the goal diagram for detecting faults and producing an alarm, but without the single use aspect. The removal of the subtree led to the removal of the amplifier component from the RFD, which lead to the renaming of the variable "AlarmTripped" to "Alarm".

2.3.1.2 Environmental Assumptions

We utilize environmental assumptions to make statements about the domain that are accepted as true. They are necessary to reason about the correctness of a set of requirements as applied to a domain. Any given domain may have unique environmental properties that a developer must be aware of when designing a device. The RFD works in a molecular

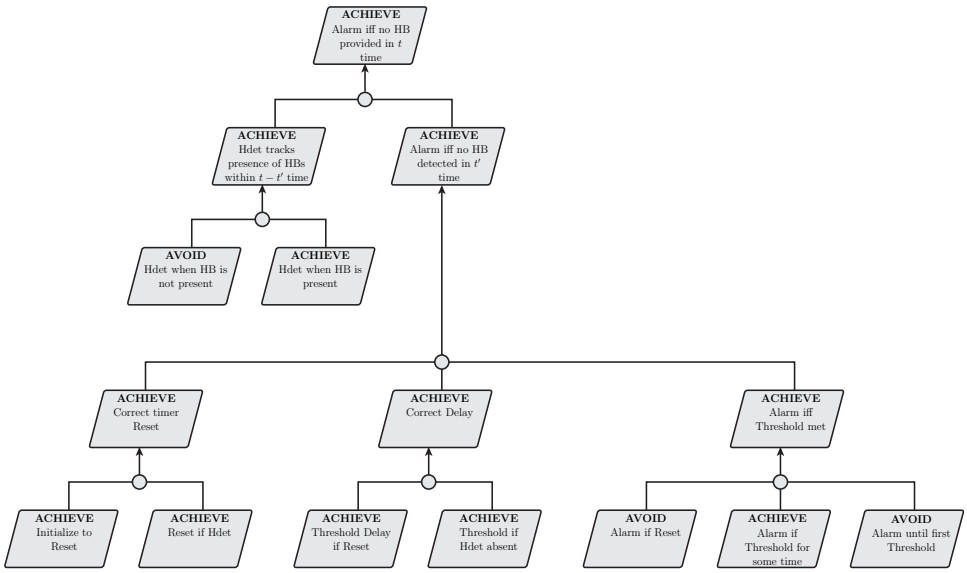

Figure 2.6 Goal Model for the Runtime Fault Detection Device.

domain, which consists of a set of molecules floating in a liquid solution. We make a number of assumptions about the domain to ensure our design performs correctly. In addition to the environmental assumptions listed with previous iterations of the MWT, see Section 2.2.2.2, we include the new assumption that H molecules are intrinsically ephemeral, so they will naturally decay over time. When a heartbeat is introduced into the solution, it will increase the number of H molecules by a programmed "dose" and then their count will naturally fall towards zero. If no new heartbeats are added, the H molecules will disappear from the solution entirely.

2.3.1.3 Formal Requirements for the RFD

In order to perform formal verification on the design of the RFD, each goal needs a formal specification. Each goal is assigned a formal specification in Continuous Stochastic Logic (CSL) as it is available in the model checking tools that we use, PRISM and SMART. The complete CSL specifications are shown in Table 2.1, which lists each goal, its formal

specification, and the agent assigned responsibility to it in a breadth first order over the goal diagram.

During the refinement of the high level goal into subgoals, and subsequent refinements into leaf goals, a number of parameters internal to the RFD specifications are generated. As in the MWT, the client specifies two time bounds and two error bounds.

u - The minimum time since the last heartbeat that the RFD may issue an alarm.

v - The maximum time since the last heartbeat by which the RFD must issue an alarm.

ϵ - Probability of error allowed by the u-delay

δ - Probability of error allowed by the v-delay

These parameters are used as a base to generate bounds on the internal parameters of the RFD. The internal parameters with a brief description are listed below.

- ϵ_1 and ϵ_2 are refinements from ϵ that determine the allowed error in avoiding *Alarms* while heartbeats are present.
- w_a is a time bound on turning on the alarm.
- α and β are allowed error in detecting the presence of heartbeats.
- w_h is the maximum time to detect the presence or absence of a heartbeat.
- ϵ_1' and ϵ_2' are allowed error in avoiding *Alarms* while a heartbeat is detected.
- g is the time allowed between detecting a heartbeat and keeping the *Alarm* off.
- δ_1' is the allowed error in initiating an *Alarm* when no heartbeat is detected.
- λ_1 is the allowed error in *Resetting* when a heartbeat is detected.
- w_{on} is the maximum time to *Reset* when a heartbeat is detected.
- γ_1 is the allowed error in setting the threshold to low when *Reset* is true.

- η_1, η_2, and η_3 are allowed errors in setting the threshold to high until a heartbeat is detected from a time when no heartbeat is detected.
- w_{th} is a time bound on how long it takes to set the threshold to high.
- λ_2 and λ_3 are allowed errors in avoiding $Alarms$ while the threshold is low.
- w_{off} is the maximum time allowed between a low threshold and keeping the $Alarm$ off.
- η_4 is the allowed error in turning the $Alarm$ on after the threshold is high.
- γ_2 is the allowed error that the $Alarm$ is off at least until the first time that the threshold is not low.

We utilized formal proofs to ensure that the formal specification of any goals children implied the satisfaction of their parent goals. As the goal diagram is the same as the MWT, with a subtree removed and only a single variable renamed, the proofs within Section 2.2.2.4 remain true over the new goal diagram, and no new proofs are needed.

2.3.2 Design

In this section we discuss the design of the RFD and how it differs from the MWT. The primary difference is that the RFD can be reset for reuse after it has produced an alarm. The amplifier restricted the MWT to only produce a single alarm. By removing the amplifier, we enable the RFD to trigger different types of responses. The RFD can provide a single use alarm if desired, but it is also capable of triggering a recovery signal to either correct the detected fault or activate backup systems. We briefly discuss the absence detector and threshold filter components and their responsibilities in goal satisfaction.

2.3.2.1 Absence Detector

The absence detector component remains largely unchanged from the previous iteration of the device, the MWT. It consists of ladders, sequential cascades of reactions that are

Table 2.1 The formal specification for each goal and its assigned agent.

Goal	**CSL Specification**	**Agent**
ACHIEVE: Alarm iff no Heartbeat provided within t time	$\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg Alarm \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2}\Box_{\leq u}\neg Alarm\right)\right] \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}\Diamond_{\leq v-w_a}\left(Alarm \vee H_{pres}\right)\right]$	RFD
ACHIEVE: Heartbeat Detected correctly tracks the presence of Heartbeats within $t - t'$ time	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}H_{det}\right] \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right]$	RFD
ACHIEVE: Alarm iff no Heartbeat detected within t' time.	$\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon}\Box_{\leq u}\neg Alarm \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{det} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_1'}\Diamond_{\leq g}\left(\boldsymbol{\mathcal{P}}_{\geq 1-\epsilon_2'}\Box_{\leq u}\neg Alarm\right)\right] \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{det} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\delta_1'}\Diamond_{\leq v-w_a}\left(Alarm \vee H_{pres}\right)\right]$	RFD
AVOID: Heartbeat Detected when Heartbeat not present	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[\neg H_{pres} \implies\right.$ $\left.\boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}\left(\neg H_{det}\ \boldsymbol{\mathcal{W}}\ H_{pres}\right)\right]$	AD
ACHIEVE: Heartbeat Detected when Heartbeat present	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[H_{pres} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\beta}\Diamond_{\leq w_h}\boldsymbol{\mathcal{P}}_{\geq 1-\alpha}H_{det}\right]$	AD

Table 2.1 (Continued)

Goal	**CSL Specification**	**Agent**
ACHIEVE: Correct Timer Reset	$Reset \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_1}\Diamond_{\leq w_{on}} Reset]$	RFD
ACHIEVE: Correct Delay	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[Reset \implies \boldsymbol{\mathcal{P}}_{\geq 1-\gamma_1}\Box_{\leq u} Th_L] \wedge$ $Th_L \implies \neg Th_H$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_1}\Diamond_{v-w_a-2*w_h-w_{th}} \boldsymbol{\mathcal{P}}_{\geq 1-\eta_2}$ $(Th_H \,\boldsymbol{\mathcal{W}}\, \boldsymbol{\mathcal{P}}_{\geq 1-\eta_3}\Diamond_{\leq wh} H_{det})]$	RFD
ACHIEVE: Alarm iff Threshold met	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[Th_L \implies$ $\boldsymbol{\mathcal{P}}_{\geq 1-\lambda_2}\Diamond_{\leq w_{off}} \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_3}\Box_{\leq u}\neg Alarm] \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[Th_H \implies$ $\boldsymbol{\mathcal{P}}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}} (Alarm \vee \neg Th_H)] \wedge$ $\boldsymbol{\mathcal{P}}_{\geq 1-\gamma_2} (\neg Alarm \,\boldsymbol{\mathcal{W}}\, \neg Th_L)$	RFD
ACHIEVE: Initialize to Reset	$Reset$	AD
ACHIEVE: Reset if Hdet	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\lambda_1}\Diamond_{\leq w_{on}} Reset]$	AD
ACHIEVE: Threshold delay if Reset	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[Reset \implies \boldsymbol{\mathcal{P}}_{\geq 1-\gamma_1}\Box_{\leq u} Th_L]$	AD

Table 2.1 (Continued)

Goal	**CSL Specification**	**Agent**
ACHIEVE: Threshold if Hdet is absent	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box[\neg H_{det} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_1}\Diamond_{v-w_a-2w_h-w_{th}}$ $\boldsymbol{\mathcal{P}}_{\geq 1-\eta_2}(Th_H \,\boldsymbol{\mathcal{W}}\, \boldsymbol{\mathcal{P}}_{\geq 1-\eta_3}\Diamond_{\leq wh} H_{det})]$	AD
AVOID: Alarm if Reset	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[Th_L \implies$ $\boldsymbol{\mathcal{P}}_{\geq 1-\lambda_2}\Diamond_{\leq w_{off}}\boldsymbol{\mathcal{P}}_{\geq 1-\lambda_3}\Box_{\leq u}\neg Alarm]$	TF
ACHIEVE: Alarm if Threshold for some time	$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\,[Th_H \implies$ $\boldsymbol{\mathcal{P}}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}}(Alarm \vee \neg Th_H)]$	TF
AVOID: Alarm until first Threshold	$\boldsymbol{\mathcal{P}}_{\geq 1-\gamma_2}(\neg Alarm \,\boldsymbol{\mathcal{W}}\, \neg Th_L)$	TF

reset upon the introduction of the heartbeat species H. However, in response to our new environmental assumptions, H is used catalytically, meaning that the reactions do not "use up" the H molecules when they occur, to reset the ladders. The behavior of the device remains the same. While no H molecules are present in the solution, the ladders will proceed forward through their cascades. If no H molecules are introduced before they reach the top rung, they each release a molecule notifying the threshold filter that they have not detected a heartbeat in a sufficient period of time. When H molecules are introduced into the solution, the ladders are reset to their initial state and remain in their lower rungs until the H molecules disappear. The difference now is that the absence detector bears no responsibility or control over the heartbeat species; its presence and absence are entirely controlled by the environment. This has the added advantage of making the ladders completely independent of one another, meaning any analysis performed on a single ladder can be applied to the absence detector as a whole.

The absence detector is assigned responsibility for the leaf goals *Avoid [Hdet when Heartbeat is not present]*, *Achieve [Hdet when Heartbeat is present]*, *Achieve [Initialize to Reset]*, *Achieve [Reset if Hdet]*, *Achieve [Threshold Delay if Reset]*, and *Achieve [Threshold if Hdet is absent]* as shown in Figure 2.6. The updated CRN for the absence detector ladders contains the following reactions

$$L_i + U \to L_{i+1} + U \quad (0 \leq i < k),$$
$$L_i + H \to L_0 + H \quad (0 < i \leq k).$$

We rename the top rung of each ladder Y to denote its use as a catalyst in the threshold filter.

2.3.2.2 Threshold Filter

The threshold filter also remains largely unchanged from the MWT. It consists of a set of ladders with a catalyst R, which restrains the ladders from proceeding through their cascades in the absence of Y molecules. The change made to the threshold filter is that we

no longer release D molecules into the solution while at the top rung of the ladder. Instead, we rename the top rung of the ladder D, meaning we produce an alarm as long as one or more of the ladders is sitting in the top rung. In the absence of Y molecules, no ladder will reach the top rung, meaning no D molecules will be produced into the solution. In the presence of many Y molecules, the ladders will still climb to the top rung, which produces a number of D molecules into the solution. Unlike the MWT, if the monitored system recovers from its failure, the absence detectors will fall out of the top rung, removing the Y molecules from the solution. The absence of Y molecules will force the threshold filter ladders to fall out of their top rungs, removing D from the solution.

The threshold filter is assigned responsibility for the leaf goals *Avoid [Alarm if Reset]*, *Achieve [Alarm if Threshold for some time]*, and *Avoid [Alarm until first Threshold]*. The threshold filter consists of the reactions

$$T_i + Y \to T_{i+1} + Y \quad (0 \leq i < k),$$
$$T_i + R \to T_0 + R \quad (0 < i \leq k),$$

where the top rung is renamed D.

2.3.3 Verification of the RFD

Formal verification methods provide assurance that a proposed system design satisfies the intended system behavior, specified by the requirements. We must ensure with high probability that if a heartbeat is present, the RFD will produce no alarm and while heartbeats remain absent for sufficiently long, the RFD will quickly produce an alarm. We discuss below the techniques used to verify the accuracy and robustness of the RFD and the results of the verification. We show using software tools that formal model checking and simulation aid in the verification of the RFD. We demonstrate that the RFD satisfies the formal specification of the goal diagram using two software model checkers PRISM [37] and SMART [10, 11].

We show the functionality of the RFD using a specific example of a system that needs to be monitored during runtime and by verifying that its heartbeat behavior is correct. Finally,

we discuss the addition of a recovery module, specific to the example monitored system, which triggers a recovery response to the RFD's alarm and show using simulation that the behavior of the composed system satisfies the intended behavior.

2.3.3.1 Verification of the RFD design

The RFD is an extension of the MWT. It utilized two components of the MWT, with very minor modifications. First, the heartbeat species is used catalytically, so it is not consumed by the absence detector. Second, the threshold filter can be reset from a state containing an alarm, a non-zero number of D molecules, to a state with no alarm, zero D molecules.

We modified the existing models, the PRISM and SMART models, for the absence detector and the threshold filter to reflect these few changes. We updated the reactions for the absence detector ladders in both models and introduced a new reaction to reflect the environmental assumption that H molecules decay naturally over time. By programming the rate of decay of the H molecules to be equivalent to the rate of consumption by the MWT absence detector, the behavior of the absence detector within both devices remains identical. We updated the reactions for the threshold filter in both models to reflect the design changes. The only changes made were to rename the top rung of the threshold filter ladders D and remove D as an additional production from the reaction, which simplified the model. We re-performed verification, as specified in Sections 2.1.3 and 2.2.4, on both models within PRISM and SMART to show that the design still satisfied its responsibilities from the goal diagram.

Using simulation, via the MATLAB package SimBiology, we verified that the RFD is reusable. From a state containing an alarm, if the monitored system begins producing heartbeats again, the RFD is reset. At system initialization, the absence detector begins with most of its population in the lower rungs of the ladder and it returns to this state when a heartbeat is detected, resetting the RFD for reuse.

2.3.3.2 Verification of RFD Interaction with a Monitored System

We assign a set of assumptions on the behavior of a monitored system. We present a system to be monitored by the RFD and show that it can satisfy the assumptions we assigned to it. We show using model checking and simulation that the interaction is correct over two different molecular population sizes.

Model checking has a number of limitations, primarily in memory and computation cost. A molecular system containing a small number of molecules, but a large number of reactions may lead to a large size model that consumes either too much memory or too much computation time to be reasonably verified. As an example, an absence detector with 5 ladders produced a continuous time Markov-chain (CTMC) with over 150,000 states, while 10 absence detectors produced a CTMC with over 9 million states. Due to the restrictions on model size, verification of the RFD is reasonable on molecular counts of up to 5. Verification on larger population counts is performed using simulations. Using SimBiology to perform simulations and PRISM and SMART for model checking, we verified that the RFD correctly translates the presence or absence of heartbeats into a correct alarm. We first utilized a hypothetical heartbeat, an abstract signal received from a monitored system for both types of verification.

We then introduced an example monitored system to show that the heartbeat assumptions made are realistic. We utilized simulation and model checking to ensure that the monitored system correctly produces heartbeats while healthy and fails to produce heartbeats while unhealthy. Model checking the monitored system suffers from the same restrictions that apply to the RFD, model checking large population sizes is infeasible. Instead, we used simulations to show that its behavior was correct on population sizes up to 100,000.

We utilized a chemical oscillator, many of which can occur naturally in nature to demonstrate the capabilities of the RFD when attached to a monitored system. Oscillators have been widely studied in molecular programming, and have previously been used as benchmarks, such as [21, 2]. A chemical oscillator can also be easily extended to produce a

heartbeat, since it continually returns to the same states. In the SCRN domain, an oscillator can be controlled to test both the normal case, heartbeats output within a specified period, and the failure case, heartbeats cease to be produced.

We used the Lotka-Volterra 3-Phase Oscillator, discussed in [8], as an example monitored system. The oscillator uses three molecular species A, B, and C, where each species corresponds to a phase of oscillation. At initialization, one of the species is given a high molecular count while the other two species are give a low, but non-zero, molecular count. The oscillator will cycle between phases in the order A to B to C and return back to A. As an example, consider the following case. If A is dominating and B and C have similar molecular counts, then reactions (2.20) and (2.22) below are equally likely to occur. However, when reaction (2.20) or (2.22) fires, the rates of all the reactions change, increasing the rate of reaction (2.20) and decreasing the rate of reaction (2.22). This continues until B is dominating, completing the transition to phase B. A similar sequence of events occurs for each phase transition.

In order to test the monitored system, we needed to construct a heartbeat interface. The RFD requires that a heartbeat interface be attached to a monitored system in order to provide heartbeats. If no heartbeat interface exists, the monitored system cannot produce heartbeats. We extended the CRN model of the oscillator with a heartbeat interface. The heartbeat interface produces H molecules as long as the oscillator is healthy. The CRN for the oscillator with the attached heartbeat interface is:

$$A + B \xrightarrow{k} 2B + H \tag{2.20}$$

$$B + C \xrightarrow{k} 2C \tag{2.21}$$

$$C + A \xrightarrow{k} 2A \tag{2.22}$$

$$H \xrightarrow{k_2} \emptyset \tag{2.23}$$

If a device has more than one failure mode, an attached heartbeat interface must cover all possible failures. In the stochastic domain, the 3-phase oscillator has two failure modes.

First, if any of the phase species A,B, or C has a molecular count of zero, no oscillations can occur. From such a state, only one of the three reactions can occur, leading to a single phase species having all of the molecules in the solution. Second, if the oscillator remains at or near equilibrium, all three species have similar molecular counts, the oscillations will be negligible and any device using it may fail. By the heartbeat interface, while the oscillator is healthy, a large number of H molecules will be produced as reaction 2.20 occurs, that is as the phase transitions from A to B. The production of H molecules must be distinct between healthy and unhealthy states. If any of the molecular species has a count of zero, the rate of all the oscillator reactions will quickly approach 0. Since the production of H is dependent on reaction 2.20, no more heartbeats will be produced. If all three species are near an equilibrium state, the production of H molecules will fall to a roughly constant amount, below the necessary threshold to be considered a heartbeat, causing the absence detector to activate.

In a "real-world" scenario, the monitored system is responsible for producing a correct heartbeat, meaning the RFD assumes that the heartbeat correctly reflects the health of a monitored system. Therefore, to verify the correct behavior of the RFD with the oscillator, we needed to confirm that the heartbeat output by the oscillator reflects its health. We define states of the monitored system to be healthy or unhealthy. If the oscillator is in a healthy state, it will send a heartbeat within a reasonable amount of time or the state will quickly become unhealthy. If the oscillator is in an unhealthy state, no heartbeat will be sent within a reasonable amount of time or the state will quickly become healthy. Formally, we define a healthy state as any state with species counts A,B,$C > 0$ AND $(A-B)^2 + (B-C)^2 + (C-A)^2 > \tau$, where τ is defined as a distance from equilibrium.

Based on the desired behavior of the heartbeat interface, we created three properties to verify over the monitored system:

Achieve[Produce heartbeats while healthy]

$$\boldsymbol{\mathcal{P}}_{\geq 1}[\Box(\text{healthy} \implies \boldsymbol{\mathcal{P}}_{\geq 1-\delta_1}[\Diamond_{\leq t_1}((\text{hbHigh} \vee \neg\text{healthy})])]$$

Avoid [Produce heartbeats while unhealthy]

$$\boldsymbol{\mathcal{P}}_{\geq 1}[\Box(\neg\text{healthy} => \boldsymbol{\mathcal{P}}_{\geq 1-\delta_2}[\Diamond_{\leq t2}(\boldsymbol{\mathcal{P}}_{\geq 1-\delta_3}[\text{hbLow } \boldsymbol{\mathcal{W}} (\boldsymbol{\mathcal{P}}_{\geq 1-\delta_4}[\Box_{\leq t_3}\text{healthy}])])])]$$

Heartbeat decays

$$\boldsymbol{\mathcal{P}}_{\geq 1}[\Box(\text{hbHigh} => \boldsymbol{\mathcal{P}}_{1-\delta_5}[\Diamond_{<=t_4}\neg\text{hbHigh}])]$$

We performed model checking on the oscillator with the three properties on population sizes of up to 200, that is the sum of the molecular counts of A, B, and C was equal to 200. PRISM and SMART both verified true that the oscillator plus heartbeat interface satisfied the above properties. We also performed simulation over a wide range of initial counts of A, B, and C along with variations in the initial ratio (e.g., 80% in A, 10% in B and C) and an initial count of 0 for H and checked these properties against the results. The simulations demonstrated that the heartbeats correctly reflect the health of the oscillator. Figure 2.7 shows a simulation of the first failure mode, where one of the species counts reached 0, causing the production of heartbeats to cease.

We used SimBiology to extensively simulate the composed system of the oscillator and RFD. A stochastic simulation of the composed system can be seen in Figure 2.8. The oscillator begins in a healthy state and produces heartbeats correctly. When one of the species counts reaches 0, the oscillations and heartbeat production cease. The absence of heartbeats is quickly detected and a recovery signal, discussed further in Section 2.3.3.3, is produced to correct the fault. The oscillator returns to a healthy state and resumes the production of heartbeats.

The oscillator is provided as an example of a monitored system and of how a client can provide a heartbeat interface, specified as a rate and size of heartbeat production, and we can generate an RFD to monitor it. The RFD is designed to work independent of the heartbeat production method, meaning any system that can produce heartbeat while healthy and cease heartbeat production while unhealthy can be monitored by the RFD. The client

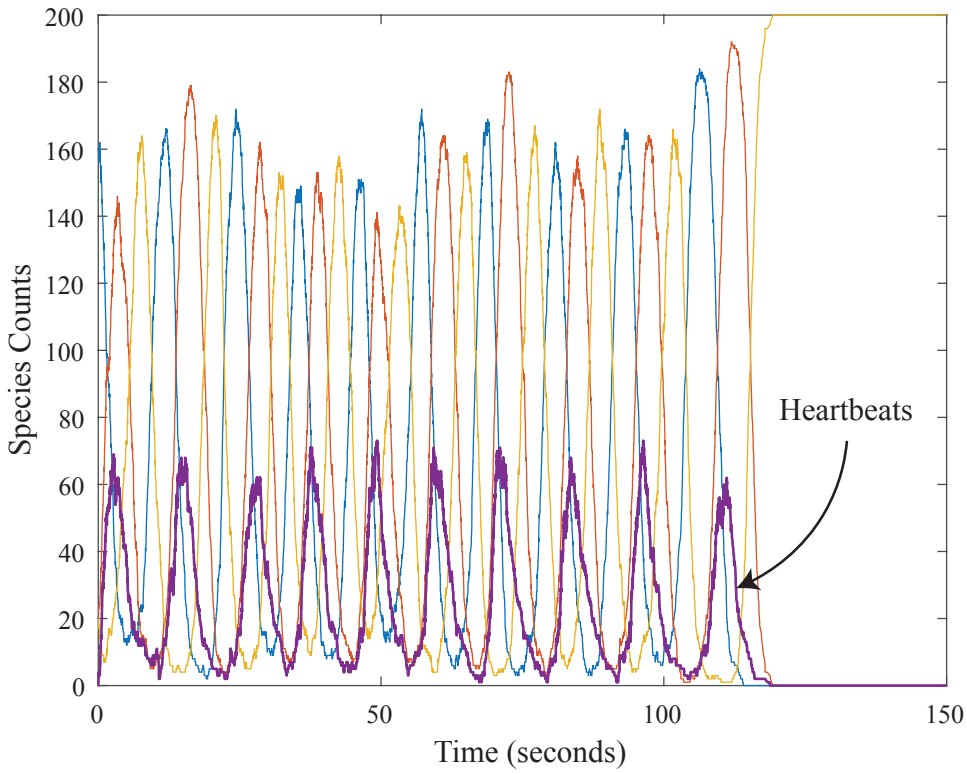

Figure 2.7 A simulation of the oscillator with the heartbeat interface.

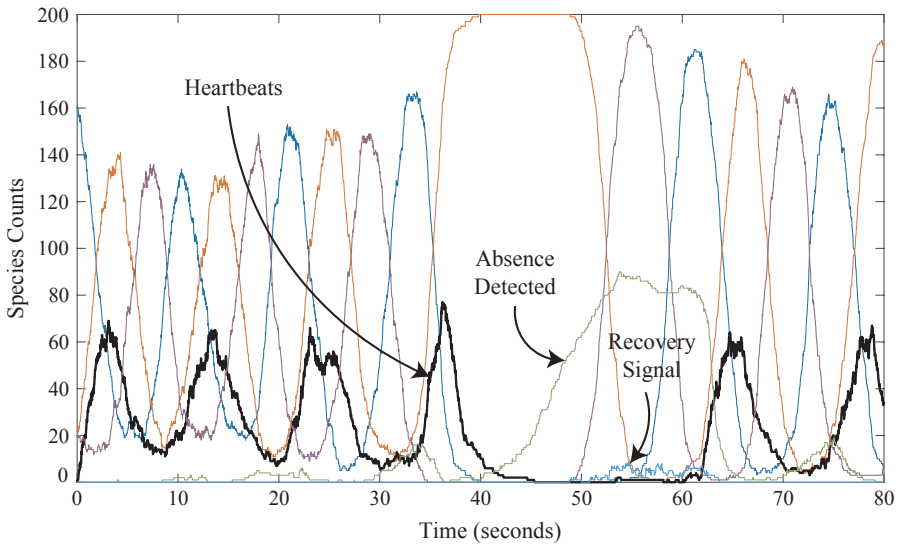

Figure 2.8 A simulation of the oscillator with the RFD.

specifies the 4 parameters, described in Section 2.3.1.3. Given a set of parameters, we will provide a design model of the RFD that satisfies the goal diagram.

2.3.3.3 Recovery Module for a Molecular Oscillator

A system that can recover from failure is far more robust than one that cannot recover autonomously. The RFD can do more than simply report the failure of the monitored system, it can trigger a recovery response. We designed a recovery module, to be attached to the output of the RFD, to reboot the oscillator from a failed state.

The CRN for the recovery module is:

$$D + A \xrightarrow{k} D + B \tag{2.24}$$

$$D + B \xrightarrow{k} D + C \tag{2.25}$$

$$D + C \xrightarrow{k_2} D + A, \tag{2.26}$$

where it is important that the reaction rate of the third reaction is different from the other two. D is used catalytically in all three reactions, meaning they are only triggered in the

presence of D molecules. If the oscillator fails due to a molecular species disappearing, the RFD produces an alarm used by the recovery module to jump start the oscillations. If the oscillator fails at equilibrium, the recovery module will turn on and since the rates of the three reactions are different, the oscillator will move away from an equilibrium state. In either case, once the oscillator is running again, the heartbeat production resumes, turning off the alarm and recovery module.

CHAPTER 3. CONCENTRATION MONITOR

The work presented in this chapter was performed in collaboration with James I. Lathrop and Robyn R. Lutz and was previously presented at the 4^{th} ACM International Conference on Nanoscale Computing and Communication Washington DC, USA, September 27-29, 2017 [19]. It is an extension of previous work by Titus H. Klinge [32].

In deterministic mass-action chemical reaction networks, it can be useful to view the concentration of a species as a signal. Signals are only allowed to be used as a catalyst in any reaction that uses them. Given a signal, we often want to react to the strength of a signal. However, in a CRN, it can be difficult to distinguish minor variations in signal strength. We define a **Concentration Monitor** as a device that monitors a signal and provides an output dependent on the concentration of the input signal. A Concentration Monitor has two thresholds, α and β, which are "high" and "low" respectively. The Concentration monitor will provide one output if the input signal has a concentration above α and the other output if the input is below β.

3.1 Robustness Properties

Many CRNs are designed to complete a specific task. In many cases, they are even designed to only work for a specific set of parameters. Any changes in the parameters can disrupt the functionality of the CRN. We seek to define a CRN to monitor the concentration of a signal robustly correct, i.e., the correct output even with some perturbation on the parameters and the inputs. Robustness of CRNs has been formally defined in [33]. Here we give a brief overview of the robustness parameters used.

We use the following parameters for robustness. A vector $\boldsymbol{\delta} = (\delta_1, \delta_2, \delta_3, \delta_4)$ defines four different types of parameters on the construction of the CRN. δ_1 defines the perturbation allowed on the input signal. This means, if an input signal X has two valid states, α and β, then X is a valid input when

$$X > \alpha - \delta_1 \text{ or } X < \beta + \delta_1,$$

meaning X is high or low respectively.

δ_2 is the accuracy of the output function. That is, when the CRN computes its output, up to δ_2 error is allowed.

δ_3 is the perturbation allowed in the initial state. If a CRN is constructed with an initial value of Y_0 in species Y, the CRN is robust in terms of δ_3 if it outputs correctly for any initial condition $\hat{Y_0}$, where

$$Y_0 - \delta_3 < \hat{Y_0} < Y_0 + \delta_3.$$

δ_4 is the perturbation allowed in the rate constants. A CRN is considered robust in terms of δ_4 if for every rate constant k, it outputs correctly for any rate constant $\hat{k}$ where

$$k - \delta_4 < \hat{k} < k + \delta_4.$$

We use ϵ to describe the perturbation allowed in the output. If an output signal X has two valid states, 0 and 1, then X is a valid output when

$$X > 1 - \epsilon \; or \; X < \epsilon,$$

meaning X is 0 or 1 respectively.

3.2 Construction of a Concentration Monitor

Given an input signal X, the device monitors for certain concentration thresholds. If the concentration of X is above the target threshold α, then within τ time the concentrations of

the output species, $\hat{X}$ and $\hat{Y}$ will have concentrations of the form

$$\hat{X} > 1 - \epsilon \tag{3.1}$$

$$\hat{Y} < \epsilon \tag{3.2}$$

Similarly, if the concentration of X is below the target threshold β then within τ time the concentrations of the output species, $\hat{X}$ and $\hat{Y}$ will have concentrations of the form

$$\hat{X} < \epsilon \tag{3.3}$$

$$\hat{Y} > 1 - \epsilon \tag{3.4}$$

The concentration monitor device extends previous work that defined a signal restoration device [32]. The signal restorer monitored for inputs of above $1 - \delta_1$ and below δ_1. Our work extends this construction for arbitrary inputs $\alpha - \delta_1$ and $\beta + \delta_1$.

We formally define the I/O Requirement for the concentration monitor as $\Phi(\tau) = (\alpha, \phi)$. We begin by defining the context assumption α as

$$\alpha(u, V, h) \equiv \Big[V = \{\hat{X}, \hat{Y}\} \text{ and } h = h_0\Big] \tag{3.5}$$

where h_0 is a measurement function defined as $h_0(x)(\hat{X}) = x(\hat{X})$ for all $x \in [0, \infty)^{|S \cup U|}$ and $\hat{X} \in V$.

As far as the requirement is concerned, the input has two states, which we will define as a and b. We define a to be any state where the input signal X has a concentration greater than α and b to be any state where the input signal has a concentration below β. We first define a predicate on the input value. For a concentration $n \in \mathbb{R}^+$

$$\phi_n(\boldsymbol{I}) \equiv (\forall t \in \boldsymbol{I})\big[u(X)(t) = n = p - u(\overline{X})(t)\big],$$

where p is the constant $X + \overline{X}$ for all $\boldsymbol{I} = [t_2, t_2]$ such that $t_2 - t_1 \geq \tau$. We define a predicate on the output value. For a bit $n \in \{0, 1\}$

$$\psi_n(\boldsymbol{I}) \equiv (\forall t \in \boldsymbol{I})\big[v(\hat{X})(t) = n = 1 - v(\hat{Y})(t)\big]. \tag{3.6}$$

We can now define the I/O Requirement ϕ of $\Phi(\tau)$ as

$$\phi(u,v) \equiv \big[\,[\phi_a(\boldsymbol{I}) \implies \psi_1(t_1+\tau,t_2)] \text{ and } [\phi_b(\boldsymbol{I}) \implies \psi_0(t_1+\tau,t_2)]\,\big] \tag{3.7}$$

for all $\boldsymbol{I} = [t_2, t_2]$ such that $t_2 - t_1 \geq \tau$. We now specify a construction for a CRN to satisfy the I/O Requirement.

3.2.1 Construction 1

Given the real numbers $\tau > 0$, $\epsilon \in \left(0, \frac{1}{2}\right)$, $\alpha > 0$, $0 \leq \beta < \alpha$, and $\boldsymbol{\delta} = (\delta_1, \delta_2, \delta_3, \delta_4)$ where $0 < \delta_1 < \frac{\alpha-\beta}{2+\alpha+\beta}$, $\delta_2 \in [0, \epsilon)$, $\delta_3 \in (0, \frac{1}{2})$ and $\delta_4 > 0$, let $b = \frac{(\alpha-\delta_1)(1+\beta)}{(1+\alpha)(\beta+\delta_1)}$ and $n = \lceil 2log_b(\frac{8}{\epsilon-\delta_2})\rceil$. Define the I/O CRN [32] $N(\tau, \epsilon, \boldsymbol{\delta}) = (U, R, S)$ where

$$U = \{X\}, \qquad S = \{L_i | 0 \leq i \leq n\} \cup \{\hat{X}, \hat{Y}\} \tag{3.8}$$

and where R contains the reactions

$$X + L_i \xrightarrow{k_1} X + L_{i+1} \qquad (\forall 0 \leq i < n) \tag{3.9}$$

$$L_i \xrightarrow{k_1} L_0 \qquad (\forall 0 < i \leq n) \tag{3.10}$$

$$\hat{Y} + L_n \xrightarrow{k_2} \hat{X} + L_n \tag{3.11}$$

$$\hat{X} \xrightarrow{k_2} \hat{Y}, \tag{3.12}$$

and where the rate constants k_1 and k_2 are defined by

$$k_1 = 2\delta_4 + \frac{2n\log(2n)}{\tau(\alpha-\delta_1)} + \frac{2}{\tau}\log\left(10\left(\frac{8}{\epsilon-\delta_2}\right)^2\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n\right) + \frac{\delta_4(2+\delta_1)}{\delta_1} \tag{3.13}$$

$$k_2 = \frac{2}{\tau}\log\left(\frac{3}{\epsilon-\delta_2}\right) + 4\delta_4 \tag{3.14}$$

We define the initial concentrations of the species as

$$\hat{X} = 0 \tag{3.15}$$

$$\hat{Y} = q \tag{3.16}$$

$$L_0 = p \tag{3.17}$$

$$L_i = 0 \qquad (\forall 0 < i \leq n) \tag{3.18}$$

where $p = \frac{10}{\epsilon - \delta_2}\left(\frac{1+\alpha}{\alpha - \delta_1}\right)^n + \delta_3$ and $q = 1 + \delta_3$.

3.2.2 Proof of Correctness

The device consists of two components, a cascade of species $L_0, \ldots, L_n$, and an output component of species $\hat{X}$ and $\hat{Y}$. The cascade moves in two directions, it climbs forward one step at a time, and it falls backwards all the way to the first species. The output component moves the concentration of $\hat{X}$ and $\hat{Y}$ based on the concentration of L_n.

The ODEs for the species $L_0, \ldots, L_n$ can be defined from their reactions.

$$\frac{dl_0}{dt} = \sum_{i=1}^{n} k_1 l_i - (k_1 x)(l_0), \tag{3.19}$$

$$\frac{dl_i}{dt} = (k_1 x) l_{i-1} - (k_1 x + k_1) l_i, \qquad \text{for } 0 < i < n, \tag{3.20}$$

$$\frac{l_n}{dt} = (k_1 x) l_{n-1} - k_1 l_n. \tag{3.21}$$

The ODEs for $\hat{X}$ and $\hat{Y}$ are

$$\frac{d\hat{x}}{dt} = (k_2 l_n)\hat{y} - k_2 \hat{x} \tag{3.22}$$

$$\frac{d\hat{y}}{dt} = k_2 \hat{x} - (k_2 l_n)\hat{y} \tag{3.23}$$

Analysis on these ODEs was performed in [32], and it provides a number of useful lemmas to prove the correctness of our CRN.

Our goal is to show that the device has correct output for a given input. It suffices to show the correctness of the device on an input greater than α and an input lower than β over a time interval $\boldsymbol{I} = [t_1, t_2]$

Lemma 3.2.1. *Given a CRN defined as in Construction 1, with a time interval $\boldsymbol{I} = [t_1, t_2]$. If the concentration of the input signal X is above $\alpha - \delta_1$ for all $t \in \boldsymbol{I}$, then by time $t_1 + \tau$,*

$$\hat{x}(t) > 1 - \epsilon. \tag{3.24}$$

We first analyze the value of $l_n(t)$, the concentration of L_n at time t. Using this, we will show the value of the output component species. First consider the behavior for an

input above α. This means that for all $t \in \boldsymbol{I}$, the input X is above α, so $x(t) > \alpha - \delta_1$. By this assumption, the input X has a constant concentration during the interval $\boldsymbol{I}$. Therefore, we wrap the concentration of X and the rate constant k_1 into the forward rate variable f. Since the backward rate is always constant we abstract it to the backward rate variable b.

We minimize the forward rate of the cascade by assuming all the concentration of $L_0, \ldots, L_n$ is in L_0 at time t_1, that is $l_0(t) = p$. Since the rate constants k_1 can be perturbed by at most δ_4, we minimize the forward rate f and maximize the backwards rate b, giving $f = (k_1 - \delta_4)(\alpha - \delta_1)$ and $b = (k_1 + \delta_4)$. By Lemma 6.6 from [32], for all time $t \in \boldsymbol{I}$,

$$l_n(t) > p\left(\frac{f}{f+b}\right)^n \sum_{i=n}^{\infty} \frac{t^i (f+b)^i}{i!} e^{-(f+b)(t-t_1)}, \tag{3.25}$$

Since l_n is monotonically increasing, for all $t \in [t_1 + \frac{\tau}{2}, t_2]$, $l_n(t) \geq l_n(\frac{\tau}{2})$ and therefore

$$l_n(t) > p\left(\frac{f}{f+b}\right)^n \sum_{i=n}^{\infty} \frac{t^i (f+b)^i}{i!} e^{-(f+b)\frac{\tau}{2}}, \tag{3.26}$$

Applying Lemma 6.7 from [32], for all $t \in [t_1 + \frac{\tau}{2}, t_2]$,

$$l_n(t) > p\left(\frac{f}{f+b}\right)^n \left(1 - ne^{-\frac{1}{n}(f+b)\frac{\tau}{2}}\right). \tag{3.27}$$

Since $k_1 > \delta_4 + \frac{2n\log(2n)}{\tau(\alpha-\delta_1)}$, Corollary 6.8 from [32] tells us

$$l_n(t) > p\left(\frac{f}{f+b}\right)^n \left(\frac{1}{2}\right) = \frac{p}{2}\left(\frac{(k_1-\delta_4)(\alpha-\delta_1)}{(k_1-\delta_4)(\alpha-\delta_1)+(k_1+\delta_4)}\right)^n \tag{3.28}$$

$$= \frac{p}{2}\left(\frac{(\alpha-\delta_1)}{(\alpha-\delta_1)+u}\right)^n \tag{3.29}$$

where $u = \frac{k_1+\delta_4}{k_1-\delta_4}$. Since $k_1 > \frac{\delta_4(2+\delta_1)}{\delta_1}$, we know $u < 1 + \delta_1$ and therefore

$$l_n(t) > \frac{p}{2}\left(\frac{(\alpha-\delta_1)}{(1+\alpha)}\right)^n \tag{3.30}$$

Since the initial concentration can be perturbed by at most δ_3, $p > \frac{10}{\epsilon-\delta_2}\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n$ and therefore

$$l_n(t) > \frac{5}{\epsilon - \delta_2} \tag{3.31}$$

Now consider the output component. The cascade component performed its operations in the first $\frac{\tau}{2}$ time of the interval $\boldsymbol{I}$. Let f be the rate of converting $\hat{Y}$ into $\hat{X}$ and b be the rate of converting $\hat{X}$ back into $\hat{Y}$. By Equation 3.31, we know that by $t_1 + \frac{\tau}{2}$,

$$l_n(t) > \frac{5}{\epsilon - \delta_2}. \tag{3.32}$$

Therefore, $f = (k_2 - \delta_4)\frac{5}{\epsilon - \delta_2}$ and $b = k_2 + \delta_4$. Lemmas 6.12 and 6.13 from [32] show that for f,b,δ_4, and k_2 as defined above, $\hat{x}(t) > q - \epsilon + \delta_2$. Since the output decision can be perturbed by at most δ_2,

$$\hat{x}(t) > q - \epsilon. \tag{3.33}$$

Since q can be perturbed by at most δ_3,

$$\hat{x}(t) > 1 - \epsilon. \tag{3.34}$$

Lemma 3.2.2. *Given a CRN defined as in Construction 1, with a time interval $\boldsymbol{I} = [t_1, t_2]$. If the concentration of the input signal X is below $\beta + \delta_1$ for all $t \in \boldsymbol{I}$, then by time $t_1 + \tau$,*

$$\hat{x}(t) < \epsilon. \tag{3.35}$$

Now consider an input below β. This means that for all $t \in \boldsymbol{I}$, the input X is below β, so $x(t) < \beta + \delta_1$. Since the rate constants k_1 can be perturbed by at most δ_4, we maximize the forward rate f and minimize the backwards rate b, giving $f = (k_1 + \delta_4)(\beta + \delta_1)$ and $b = (k_1 - \delta_4)$. By Lemma 6.10 from [32], for all $t \in \boldsymbol{I}$,

$$l_n(t) < pe^{-b(t-t_1)} + p\left(\frac{f}{f+b}\right)^n \left(1 - e^{-b(t-t_1)}\right) \tag{3.36}$$

Since this function is monotonically decreasing, for all $t \in [t_1 + \frac{\tau}{2}, t_2]$,

$$l_n(t) < p\left(\frac{f}{f+b}\right)^n + pe^{-b\frac{\tau}{2}} \tag{3.37}$$

$$= p\left(\frac{(k_1 + \delta_4)(\beta + \delta_1)}{(k_1 + \delta_4)(\beta + \delta_1) + (k_1 - \delta_4)}\right)^n + pe^{-b\frac{\tau}{2}} \tag{3.38}$$

$$= p\left(\frac{(\beta + \delta_1)}{(\beta + \delta_1) + u}\right)^n + pe^{-b\frac{\tau}{2}}, \tag{3.39}$$

where $u = \frac{k_1-\delta_4}{k_1+\delta_4}$. Since $k_1 > \frac{\delta_4(2-\delta_1)}{\delta_1}$, we know that $u > 1-\delta_1$ and therefore for all $t \in [t_1 + \frac{\tau}{2}, t_2]$,

$$l_n(t) < p\left(\frac{(\beta+\delta_1)}{(1+\beta)}\right)^n + pe^{-b\frac{\tau}{2}}. \tag{3.40}$$

Since $p < \frac{10}{\epsilon-\delta_2}\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n + 2\delta_3$,

$$l_n(t) < \frac{10}{\epsilon-\delta_2}\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n\left(\frac{(\beta+\delta_1)}{(1+\beta)}\right)^n + 2\delta_3\left(\frac{(\beta+\delta_1)}{(1+\beta)}\right)^n + pe^{-b\frac{\tau}{2}} \tag{3.41}$$

$$< \frac{10}{\epsilon-\delta_2}\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n\left(\frac{(\beta+\delta_1)}{(1+\beta)}\right)^n + \left(\frac{(\beta+\delta_1)}{(1+\beta)}\right)^n + pe^{-b\frac{\tau}{2}} \tag{3.42}$$

$$< \frac{10+\epsilon-\delta_2}{\epsilon-\delta_2}\left(\frac{(1+\alpha)(\beta+\delta_1)}{(\alpha-\delta_1)(1+\beta)}\right)^n + pe^{-b\frac{\tau}{2}} \tag{3.43}$$

$$< \frac{32}{3(\epsilon-\delta_2)}\left(\frac{(\alpha-\delta_1)(1+\beta)}{(1+\alpha)(\beta+\delta_1)}\right)^{-n} + pe^{-b\frac{\tau}{2}}. \tag{3.44}$$

Since $n \geq \log_{\left(\frac{(\alpha-\delta_1)(1+\beta)}{(1+\alpha)(\beta+\delta_1)}\right)}\left(\frac{64}{(\epsilon-\delta_2)^2}\right)$,

$$l_n(t) < \frac{32}{3(\epsilon-\delta_2)}\left(\frac{(\epsilon-\delta_2)^2}{64}\right) + pe^{-b\frac{\tau}{2}} \tag{3.45}$$

$$= \frac{(\epsilon-\delta_2)}{6} + pe^{-b\frac{\tau}{2}}. \tag{3.46}$$

As we showed before $p < \frac{32}{3(\epsilon-\delta_2)}\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n + 2\delta_3$,

$$l_n(t) < \frac{(\epsilon-\delta_2)}{6} + \frac{32}{3(\epsilon-\delta_2)}\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n e^{-b\frac{\tau}{2}} + 2\delta_3 e^{-b\frac{\tau}{2}}. \tag{3.47}$$

Since $b = k_1 - \delta_4 > \frac{2}{\tau}\log\left(\left(\frac{640}{(\epsilon-\delta_2)^2}\right)\left(\frac{1+\alpha}{\alpha-\delta_1}\right)^n\right)$,

$$l_n(t) < \frac{(\epsilon-\delta_2)}{6} + \frac{(\epsilon-\delta_2)}{60} + 2\delta_3 e^{-b\frac{\tau}{2}} \tag{3.48}$$

$$< \frac{(\epsilon-\delta_2)}{6} + \frac{(\epsilon-\delta_2)}{30}, \tag{3.49}$$

whence for all $t \in [t_1 + \frac{\tau}{2}, t_2]$

$$l_n(t) < \frac{(\epsilon-\delta_2)}{5} \tag{3.50}$$

Now consider the output component. The cascade component performed its operations in the first $\frac{\tau}{2}$ time of the interval $\boldsymbol{I}$. Let f be the rate of converting $\hat{Y}$ into $\hat{X}$ and b be the

rate of converting $\hat{X}$ back into $\hat{Y}$. By Equation 3.50, we know that by $t_1 + \frac{\tau}{2}$,

$$l_n(t) < \frac{(\epsilon - \delta_2)}{5} \tag{3.51}$$

Therefore, $f = (k_2 + \delta_4)\frac{(\epsilon-\delta_2)}{5}$ and $b = k_2 - \delta_4$. Lemmas 6.12 and 6.13 from [32] show that for f,b,δ_4, and k_2 as defined above, $\hat{y}(t) > q - \epsilon + \delta_2$. Since $\hat{x}(t) + \hat{y}(t) = q$ for all $t \in [0, \infty)$,

$$\hat{x}(t) < \epsilon - \delta_2 \tag{3.52}$$

$$\hat{y}(t) > q - \epsilon + \delta_2 \tag{3.53}$$

for all $t \in [t_1 + \tau, t_2]$. Since q can be perturbed by at most δ_3, and the output can be perturbed by at most δ_2

$$\hat{x}(t) < \epsilon \tag{3.54}$$

$$\hat{y}(t) > 1 - \epsilon \tag{3.55}$$

Thus a CRN defined as in Construction 1 satisfies the I/O Requirement for a concentration monitor.

CHAPTER 4. ROBUST CIRCUITS

The work presented in this chapter was performed in collaboration with Titus H. Klinge and James I. Lathrop.

Logic gates play a key role in computing with uses ranging from simple boolean operations to microprocessors. The correctness of logic gates is important to ensure no errors arise during computation. The utility of logic gates is also desirable in safety-critical programmed molecular systems. For example, logic gates can be used to handle data and to verify properties on data.

In this chapter we present a construction for a 2-input NAND gate made from a chemical reaction network. We prove that the design simulates a NAND gate correctly and robustly with respect to perturbations on rate constants, initial concentrations, output calculation and input values. We then prove that any combinational logic gate can be simulated using the 2-input NAND gate. All the CRNs presented in this section use deterministic mass action kinetics.

4.1 2-Input NAND Gate

We construct an NAND gate that takes 2 inputs, X_1 and X_2. X_1 and X_2 are "dual-railed", meaning they have compliment species, $\overline{X}_1$ and $\overline{X}_2$ respectively, such that $X_1 + \overline{X}_1 = C$ is always true for some initial constant C. The NAND Gate has two output species, $\hat{X}$ and $\hat{Y}$, where $\hat{Y}$ high is considered to be an output of 1 and $\hat{X}$ high is considered to be an output of 0.. We consider all inputs to be binary, that is they are considered to be 1 when their concentration is above some $1 - \hat{\delta_1}$ and they are considered to be 0 if their

concentration is below $\hat{\delta_1}$. If both inputs X_1 and X_2 are considered to be 1 over a time interval $\boldsymbol{I} = [t_1, t_2]$, then by $t_1 + \tau$ time the concentrations of the output species, $\hat{X}$ and $\hat{Y}$ will have concentrations of the form

$$\hat{X} > 1 - \epsilon \tag{4.1}$$

$$\hat{Y} < \epsilon \tag{4.2}$$

Similarly, if the concentration of either X_1 or X_2 is below $\hat{\delta_1}$ then within τ time the concentrations of the output species, $\hat{X}$ and $\hat{Y}$ will have concentrations of the form

$$\hat{X} < \epsilon \tag{4.3}$$

$$\hat{Y} > 1 - \epsilon \tag{4.4}$$

More formally, given the propagation delay τ, as a positive real number, we can define the requirement for a NAND gate as $\Phi(\tau) = (\alpha, \phi)$. We define the context assumption α as

$$\alpha(u, V, h) \equiv \Big[V = \{\hat{Y}, \hat{X}\} \text{ and } h = h_0\Big] \tag{4.5}$$

where h_0 is a measurement function defined as $h_0(x)(\hat{Y}) = x(\hat{Y})$ for all $x \in [0, \infty)^{|S \cup U|}$ and $\hat{Y} \in V$.

The two inputs can have the binary values of $n, m \in \{0, 1\}$, so we define the predicate on an input as

$$\begin{aligned} \phi_{n,m}(\boldsymbol{I}) \equiv (\forall t \in \boldsymbol{I})\big[u(X_1)(t) = n = 1 - u(\overline{X}_1)(t) and \\ u(X_2)(t) = m = 1 - u(\overline{X}_2)(t)\big] \end{aligned} \tag{4.6}$$

for all $\boldsymbol{I}$ of length greater than or equal to τ. We also define a predicate on the output value. For a bit $n \in \{0, 1\}$

$$\psi_n(\boldsymbol{I}) \equiv (\forall t \in \boldsymbol{I})\big[v(\hat{Y})(t) = n = 1 - v(\hat{X})(t)\big]. \tag{4.7}$$

We can now define the I/O Requirement ϕ of $\Phi(\tau)$ as

$$\begin{aligned} \phi(u, v) \equiv \big[\,[\phi_{11}(\boldsymbol{I}) \implies \psi_0(t_1 + \tau, t_2)] \text{ and} \\ [\phi_{00}(\boldsymbol{I}) \vee \phi_{01}(\boldsymbol{I}) \vee \phi_{10}(\boldsymbol{I}) \implies \psi_1(t_1 + \tau, t_2)]\,\big] \end{aligned} \tag{4.8}$$

for all $\boldsymbol{I} = [t_2, t_2]$ such that $t_2 - t_1 \geq \tau$.

4.1.1 Construction 2

We construct the NAND gate as above in Construction 1 with a minor modification. We replace reaction 3.9 with the following reactions

$$X_1 + L_i \xrightarrow{k_1} X_1 + L_{i+1} \qquad (\forall 0 \leq i < n) \tag{4.9}$$

$$X_2 + L_i \xrightarrow{k_1} X_2 + L_{i+1} \qquad (\forall 0 \leq i < n) \tag{4.10}$$

The ODEs for this cascade change to

$$\frac{dl_0}{d_t} = \sum_{i=1}^{n} k_1 l_i - k_1(x_1 + x_2)(l_0), \tag{4.11}$$

$$\frac{dl_i}{d_t} = k_1(x_1 + x_2)l_{i-1} - (k_1(x_1 + x_2) + k_1)l_i, \qquad \text{for } 0 < i < n, \tag{4.12}$$

$$\frac{l_n}{dt} = (k_1(x_1 + x_2))l_{n-1} - k_1 l_n. \tag{4.13}$$

That is, both input species equally push the cascade forward.

We also add some constraints to the values of α, β, and δ_1. We define α as

$$\alpha = 2, \tag{4.14}$$

since we want the input to be above α when both input species X_1 and X_2 are considered to be 1. We define β as

$$0 \leq \beta < \alpha, \tag{4.15}$$

however, the closer β approaches α the more accurate the NAND gate is. Ideally, we define β as at least

$$\beta = \alpha - 1 = 1. \tag{4.16}$$

Each input species is allowed to be perturbed by at most $\hat{\delta_1}$. Therefore we add the constraint

$$2\hat{\delta_1} < \delta_1 \tag{4.17}$$

which ensures that, no matter how far either input species is perturbed within its bounds, the perturbation will never exceed δ_1.

4.1.2 Proof of Correctness

Lemma 4.1.1. *Given a CRN N defined as in Construction 2, with a time interval $\boldsymbol{I} = [t_1, t_2]$ such that $t_2 - t_1 \geq \tau$, then $N \models_{\epsilon}^{\boldsymbol{\delta}} \Phi(\tau)$.*

Since X_1 and X_2 are pushing the same cascade forward, it is useful to define the function $x(t) = x_1(t) + x_2(t)$. Observe that if we substitute this input $x = x_1 + x_2$ into Construction 2, the ODEs are the same as in Construction 1.

The NAND gate requirement cleanly breaks into two subcomponents. It suffices to show that each case is satisfied. Since the NAND gate is an extension of the concentration monitor, by Lemmas 3.2.1 and 3.2.2, it suffices to show instead that

$$[\phi_{11}(\boldsymbol{I}) \implies x(t) > \alpha - \delta_1] \text{ and} \tag{4.18}$$

$$[\phi_{00}(\boldsymbol{I}) \vee \phi_{01}(\boldsymbol{I}) \vee \phi_{10}(\boldsymbol{I}) \implies x(t) < \beta + \delta_1] \tag{4.19}$$

Consider the case where all inputs $X_0, \ldots X_{k-1}$ are considered to be 1. By their perturbation bounds, this means that

$$x_1(t) > 1 - \hat{\delta_1}, \tag{4.20}$$

$$x_2(t) > 1 - \hat{\delta_1}. \tag{4.21}$$

By the definition of X,

$$x(t) = x_1(t) + x_2(t). \tag{4.22}$$

Since both inputs are considered to be 1,

$$x(t) > 2(1 - \hat{\delta_1}) \tag{4.23}$$

$$= 2 - 2\hat{\delta_1} \tag{4.24}$$

$$= \alpha - 2\hat{\delta_1} \tag{4.25}$$

$$x(t) > \alpha - \delta_1. \tag{4.26}$$

For all other sets of inputs, the worst case is when only 1 input species X_l has a value below $\hat{\delta_1}$ and the other species has its highest value,

$$x_j(t) < 1 + \hat{\delta_1} \tag{4.27}$$

By the definition of X,

$$x(t) = x_1(t) + x_2(t). \tag{4.28}$$

Assume that input X_l has a value below $\hat{\delta_1}$. Since the other input is considered to be 1,

$$x(t) < (1 + \hat{\delta_1}) + x_l(t) \tag{4.29}$$

$$< 1 + \hat{\delta_1} + \hat{\delta_1} \tag{4.30}$$

$$= 1 + 2\hat{\delta_1} \tag{4.31}$$

Since $\alpha = 2$,

$$x(t) < \alpha - 1 + 2\hat{\delta_1}. \tag{4.32}$$

Since $\beta = \alpha - 1$,

$$x(t) < \beta + 2\hat{\delta_1}. \tag{4.33}$$

Whence by the definition of $\hat{\delta_1}$,

$$x(t) < \beta + \delta_1. \tag{4.34}$$

4.2 Combinational Circuits

The following proof is an application of a proof defined in [17]. We now present a proof for robustly simulating any combinational circuit. We must first define the I/O requirement for a robust combinational circuit. Consider a combinational circuit $C_{n,m}$ with n-inputs and m-outputs. We do not consider feedback loops, so the circuit is a directed acyclic graph where each node is a 2-input NAND gate. The circuit has n binary inputs and m binary outputs, so there is an edge in for each input and an edge out for each output. The circuit has a depth equivalent to the longest path from an input edge to an output edge within the circuit.

We define the input species for a circuit $C_{n,m}$ as

$$U = \{X_i, \overline{X}_i | 0 \leq i < n\}. \tag{4.35}$$

We define the requirement $\Phi(C_{n,m}, \tau) = (\phi, \alpha)$ with α defined as

$$\alpha(u, V, h) \equiv \left[V = \{\hat{Y}_i, \hat{X}_i \mid 0 \leq i < m\} \text{ and } h = h_0\right]. \tag{4.36}$$

The circuit $C_{n,m}$ has an expected set of outputs for any binary input string w of length n. For such an input string $w \in \{0,1\}^n$ and an input $u \in C\,[U]$, we use $u(t) = w$ to indicate that $u(t)(X_i) = w\,[i]$ for each $0 \leq i < n$. Since X_i and $\overline{X}_i$ are dual railed, this also means that $u(t)(\overline{X}_i) = \overline{w}\,[i]$. Which allows us to define the predicates

$$\phi_w(I) \equiv (\forall t \in I)\big[u(t) = w\big]$$
$$\psi_w(I) \equiv (\forall t \in I)\big[v(t) = w\big].$$

Using these predicates, we can define the I/O requirement ϕ for a combinational circuit $C_{n,m}$ as

$$\phi(u, v) \equiv (\forall w \in \{0,1\}^n)\big[\phi_w(I) \implies \psi_{C_{n,m}(w)}(t_1 + \tau, t_2)\big] \tag{4.37}$$

for all $\boldsymbol{I} = [t_1, t_2]$ of length greater than or equal to τ.

4.2.1 Construction 3

Given an arbitrary combinational circuit $C_{n,m}$ with G gates and a depth of d. Given the real numbers $\tau > 0$, and $\boldsymbol{\delta} = (\delta_1, \delta_2, \delta_3, \delta_4)$ where $\delta_1 \in \{0, \frac{1}{3}\}$, $\delta_2 \in [0, \epsilon)$, $\delta_3 \in (0, \frac{1}{2})$ and $\delta_4 > 0$, with $\epsilon \in \big(0, \delta_1\big)$. We define the CRN N by joining G copies of the two-input NAND gates with the above constants and τ as $\frac{\tau}{d}$ in the obvious way to simulate $C_{n,m}$.

Lemma 4.2.1. *Given a combinational circuit $C_{n,m}$ and a CRN N defined as in Construction 3, $N \models_{\epsilon}^{\boldsymbol{\delta}} \Phi(C_{n,m}, \tau)$*

4.2.2 Proof of correctness

This lemma follows from the robustness of each two-input NAND gate. Each NAND gate robustly changes its output within $\frac{\tau}{d}$ time. Since the longest path in the circuit is d, within τ time, the circuit will have the correct output. The output of each circuit will be within ϵ of the correct output. Since ϵ is smaller than δ_1, the input of each circuit will be within δ_1 of 1 or 0 meaning that each circuit will robustly compute the correct output. Therefore the CRN N will robustly simulate the circuit $C_{n,m}$ within the proper time period τ.

CHAPTER 5. LOGGING OF CRNS

The work presented in this chapter was performed in collaboration with James I. Lathrop and Robyn R. Lutz. This work was previously presented at the 4^{th} ACM International Conference on Nanoscale Computing and Communication Washington DC, USA, September 27-29, 2017 [19]. All CRNs presented in this section utilize deterministic mass action kinetics.

5.1 Logging the State of a CRN

In software systems, logs have a multitude of uses. They can be read externally, by either another system or an external user. They can be used to synchronize multiple devices across a network, i.e., to make sure all devices have the same value for a variable. A log can be used to ensure data is correct or valid before it is passed to another device. They can also be used as a checkpoint for future rollbacks in the case of an error. Each of these behaviors can be desirable in a molecular program. We present a method to create a log of a molecular program.

5.1.1 Log Device

We first define the device used to record the state of a CRN, called the **Log Device**. This device tracks a set of input signals, specified at system design, and a log signal. When the log signal is low, the device will not track the values of the input signals. However, when the log signal is high, for each tracked signal, the log device will set the concentration of a copy signal equal to that of the tracked signal. This allows the log device to only take a log when desired.

A single log device can track any number of species, but can only maintain a single log at a time.

We construct a device to record the state of a CRN using two components, a **Log Device** and a **Log Interpreter**, see Figure 5.1. We specify a Log Device to copy the values of all the species to be logged, made up of **Follower CRNs**. Consider a CRN device with s number of species to be logged. Let X_i be the ith species to be logged in the device, C_i be the species that will contain the value of X_i after it is logged, and log be a catalyst. For each such species X_i, we construct a Follower CRN as an I/O CRN with

$$U = \{X_i, log\}$$

$$S = \{C_i\},$$

and where R contains the reactions

$$log + X_i \xrightarrow{k} log + X_i + C_i, \tag{5.1}$$

$$log + C_i \xrightarrow{k} log \tag{5.2}$$

where X_i is only used catalytically, since it is an input signal. Since log is a catalyst, reactions (5.1) and (5.2) have a propensity dependent on the concentration of log. When log is entirely absent, i.e., the concentration of log is 0, then neither reaction can occur, meaning that C_i will remain at a constant value. When log is present in any amount, reactions (5.1) and (5.2) can occur, meaning that C_i will approach the value of X_i.

Intuitively, reactions (5.1) and (5.2) have similar rates. They both share the catalyst log and the rate constant k. The only difference is that reaction (5.1) has X_i in its set of reactants and reaction (5.2) has C_i in its set of reactants. When the concentration of X_i is larger than the concentration of C_i, reaction (5.1) will happen more often. On the other hand, if the concentration of C_i is larger than the concentration of X_i, then reaction (5.2) will happen more often. Therefore, whatever the values of X_i and C_i, the value of C_i will be more likely to approach the value of X_i than it will be to leave the value of X_i. This behavior remains the same for any concentration of log above 0, but the speed differs.

To take a log, we first add log to the system to a target amount, speeding up the follower reactions and allowing them to copy their target species. We then need to remove as much log from the system as possible, making the rates of the follower reactions approach 0.

Since we are in a deterministic setting, it is impossible to make the concentration of $log = 0$ after it has been introduced to the system. Therefore, we seek to lower the value of log to some $\epsilon > 0$. The smaller we make ϵ, the closer to 0 the concentration of log will be, and thus the slower reactions (5.1) and (5.2) will be. We accomplish this using the previously discussed Concentration Monitor.

To control the concentration of log between $1 - \epsilon$ and ϵ, let $logSignal$ be the input that requests a log be taken. We construct a Log Interpreter, a Concentration Monitor on the input $logSignal$. We choose α and β based on the desired log signal input range. Intuitively, the user provides a threshold above which the input signal is On (α) and a threshold below which the input signal is Off (β), with the range between α and β unknown. We choose a $\tau > 0$ time delay and an ϵ as small as desired. We assign the species log and $\overline{log}$ to the output species $\hat{X}$ and $\hat{Y}$ respectively, where log and $\overline{log}$ are "dual-railed", i.e., a high concentration of $\overline{log}$ indicates a low concentration of log, and vice versa.

When the concentration of $logSignal$ is above α, log will have a concentration of at least $1 - \epsilon$. When it is below β, log will have a concentration of at most ϵ. A log request inputs enough $logSignal$ to raise its concentration above α, raising the concentration of log to at least $1 - \epsilon$. When we remove enough of $logSignal$ to lower its concentration below β, we reduce the concentration of log to at most ϵ. Figure 5.2 shows the Logging Device applied to record the concentration of a signal.

5.2 Logging of Valid States of a CRN

Some applications require that only states meeting certain specifications be logged. For example, consider a CRN system where all the signals used are read as digital signals, i.e.,

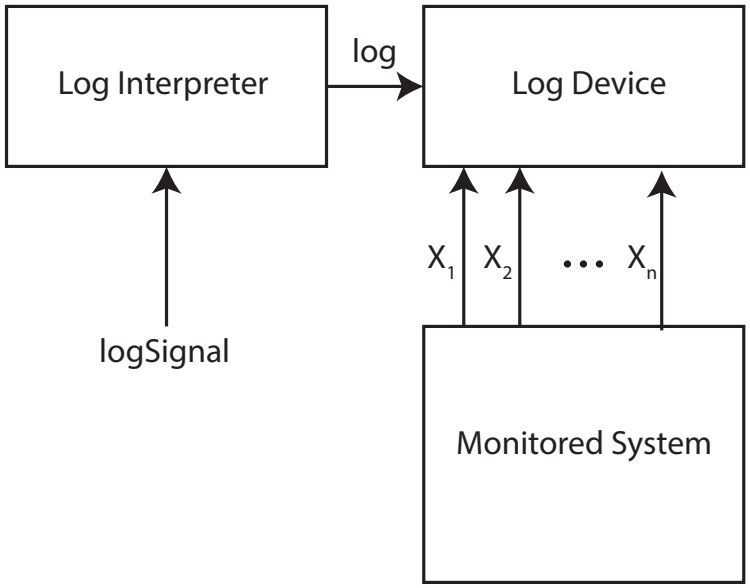

Figure 5.1 Logging a State.

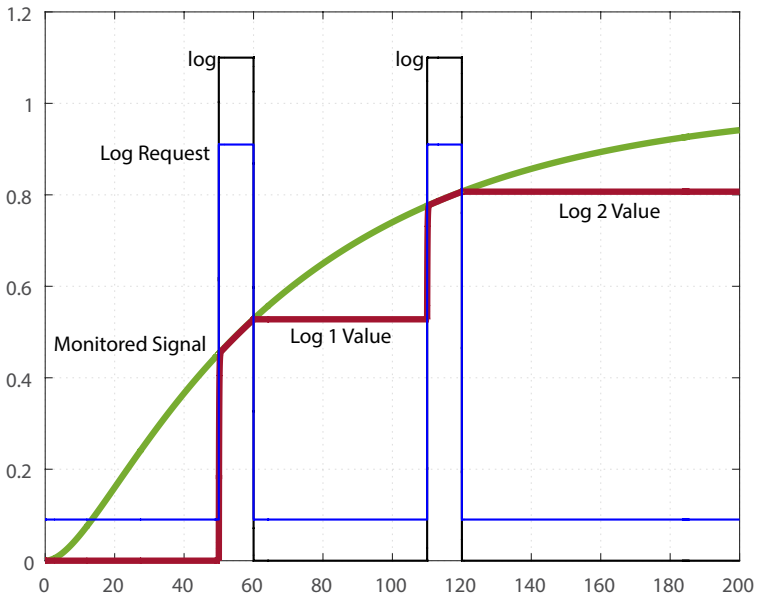

Figure 5.2 Simulation of the Logging Device on a monitored system adapted from [36].

within some ϵ of 0 or 1. In such a case, saving a system state containing values in between 0 and 1 is undesired.

The state logging algorithm is insufficient to only log valid states, as it will log when it is told to regardless of the validity of the system state. Logging only valid states adds two new requirements.

1. Detect whether a signal X satisfies its validity properties, and thus has a valid concentration.

2. Detect whether a state validity property is satisfied by the current system state.

We extend the state logging algorithm to only allow the logging of valid system states.

5.2.1 State Validity

We first define what it means for a signal to be valid. Each signal is a concentration of a species, meaning that its validity is determined by properties of its concentration. We define the validity of a signal in terms of upper and/or lower bounds. We specify validity bounds for a signal using real numbers a and b with $a > 0$ and $0 < b < a$. When defining a bound we use a and b to determine the range of correct detection. When defining a lower bound, we say the signal is *valid* if its concentration is above a, *invalid* if its concentration is below b and *unknown* if its concentration is in between a and b. For an upper bound, we say the signal is *valid* if its concentration is below b, *invalid* if its concentration is above a and *unknown* if its concentration is in between a and b. Figure 5.3 shows a graphical representation of these bounds. A validity property is a boolean expression of the validity bounds of a signal using combinational relations. A state validity property is a boolean expression of the validity properties of the system signals.

5.2.2 Validity Detector

To satisfy the first requirement, we begin by constructing a device that detects whether a signal satisfies a validity bound. For a given signal X_i and an a and b defining a validity

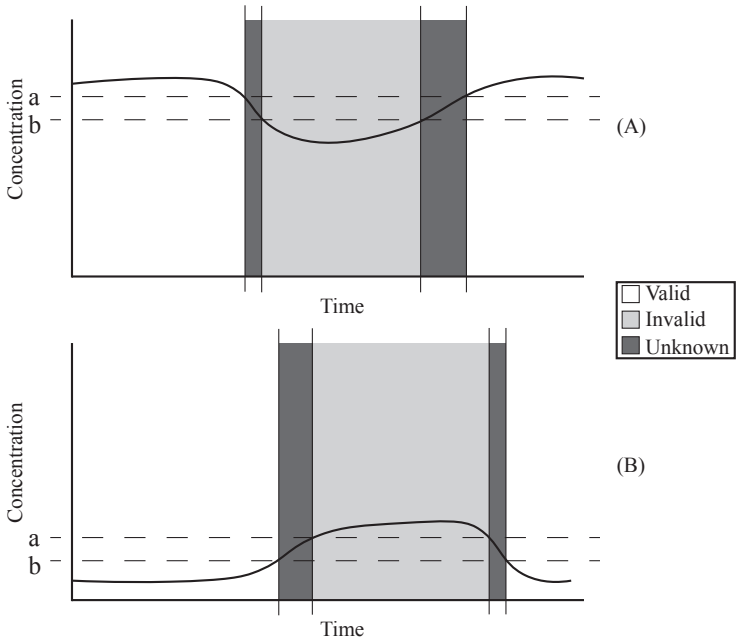

Figure 5.3 These figures depict the validity of an input signal over time. (A) shows a lower bound and (B) shows an upper bound.

property, we construct a Concentration Monitor. We define a **Validity Detector** as a Concentration Monitor (see Chapter 3) with the following properties. Let $\hat{X}_i$ and $\hat{Y}_i$ represent the validity of the signal, meaning that if $\hat{X}_i$ has a high concentration, the signal is valid and if $\hat{Y}_i$ has a high concentration, the signal is invalid. Assign the validity bounds a and b to be α and β respectively. Assign X_i to be the input signal X. Assign τ to be a value reflecting the acceptable delay between input and output. Assign ϵ to be the accuracy tolerance of the output. Assign $\hat{X}_i$ and $\hat{Y}_i$ to be the outputs $\hat{X}$ and $\hat{Y}$ respectively for computing a lower bound. Assign $\hat{Y}_i$ and $\hat{X}_i$ to be the outputs $\hat{X}$ and $\hat{Y}$ respectively for computing an upper bound.

By the definition of a Concentration Monitor, the Validity Detector satisfies its requirements. Suppose there is a lower bound a on the input signal X_i. If the concentration of the input species X_i is above a,

$$X_i > \alpha$$

and thus, within τ time

$$\hat{X}_i > 1 - \epsilon.$$

If the concentration of X_i is below b,

$$X_i < \beta$$

and thus, within τ time

$$\hat{Y}_i > 1 - \epsilon.$$

Suppose there is an upper bound b on the input signal X_i. If the concentration of the input species X_i is below b,

$$X_i < \beta$$

and thus, within τ time

$$\hat{X}_i > 1 - \epsilon.$$

If the concentration of X_i is above a,

$$X_i > \alpha$$

and thus, within τ time

$$\hat{Y}_i > 1 - \epsilon.$$

If the concentration of X_i is between a and b, its state is unknown and therefore no specific output is guaranteed.

5.2.3 Validity Gate

As shown in Section 4.2, we can robustly simulate any combinational circuit. We construct a validity gate as a combinational circuit to simulate the state validity property of the monitored system. The boolean variables taken as input are the outputs of the validity detectors of each signal. Since the output of each validity detector is robust, the output of the combinational circuit will be robust and correct by Lemma 4.2.1.

5.2.4 Log Controller

The log controller is an extension of the log interpreter from the simple logging device. Instead of allowing a log to occur whenever the log signal is high, we also require the output of the validity gate to be high, i.e., the state is valid. We can construct an AND gate robustly using two NAND gates. We input the log signal and validity gate output into a NAND gate. We then feed the output of the NAND gate into both outputs of another NAND gate. This creates a NOT gate attached to the output of a NAND gate, making an AND gate. Therefore, the log controller only has a high output when both the log signal and validity gate output are high.

5.2.5 Valid-only Log device

We now show how to construct a logging device that only logs valid states, shown in Figure 5.4, using the components described in Sections 5.2.2, 5.2.3, and 5.2.4. Specifically, this helps prevent data from being transmitted that does not meet data quality requirements.

As before, we utilize Follower CRNs to log each signal.

For each monitored signal and a validity bound, we construct a validity detector. Each validity detector provides a high output if its input signal satisfies its validity bound and a low output if it is known to not satisfy its validity bound. If the input signal to a validity detector falls within its error region, no guarantees are made on the output of the validity detector.

We construct a validity gate over the outputs of all the validity detectors to determine if the current state satisfies a state validity property. The validity gate provides an output of $Y > 1 - \epsilon$ if the state validity property is satisfied and an output of $Y < \epsilon$ if the state validity property is not satisfied. If any of the monitored signals are in an unknown known state, there are no guarantees on the output of the Validity Gate.

We then construct a log controller as the AND of $logSignal$ and Y, with the error tolerances on $logSignal$ and Y to determine the parameters, while still having its outputs as

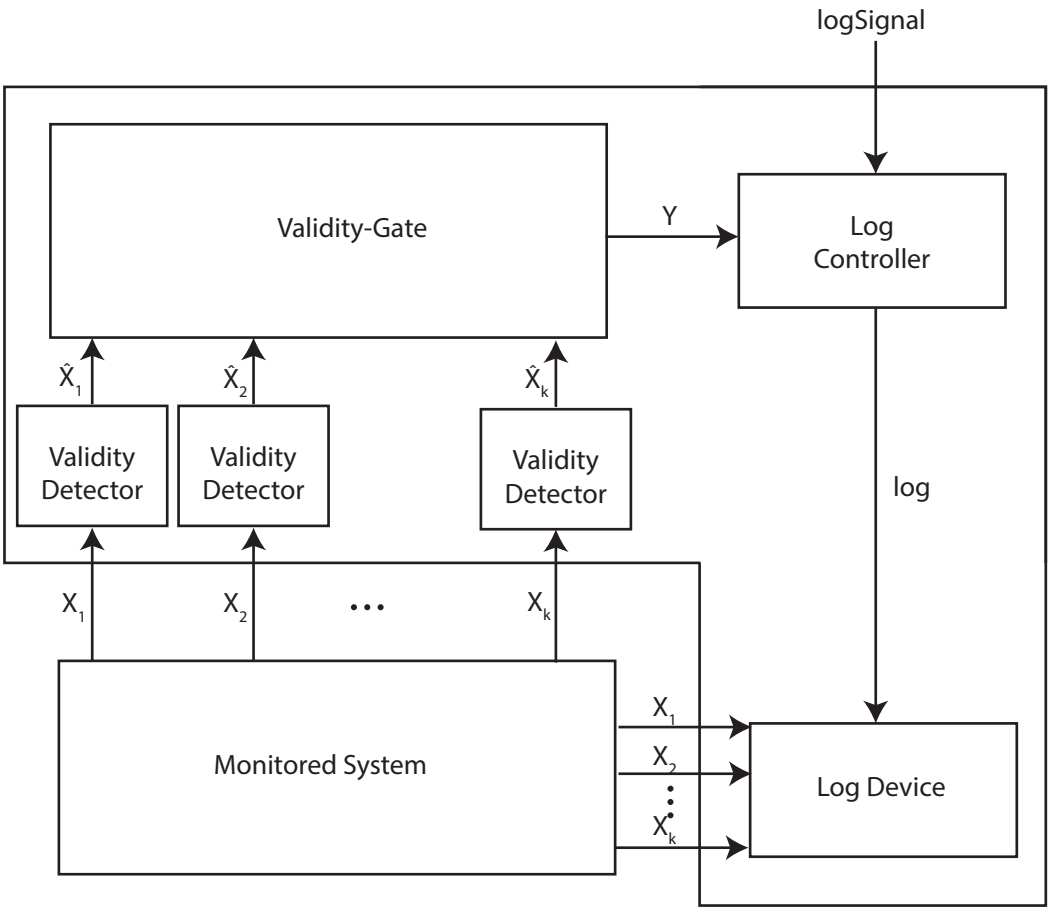

Figure 5.4 Logging Device Conditional on Validity.

log and $\overline{log}$. We use a log controller to control the presence of the catalyst for the Follower CRNs.

Figure 5.5 shows two simulations of the logging device conditional on validity. Both plots use the same set of devices in their construction with the validity property ($X_1 \geq 0.9 \wedge X_2 \geq 0.9$). As shown by the figure on the left, when both X_1 and X_2 have concentrations of at least 0.9, the state is considered valid, meaning the validity gate has an high output and the state is logged when the log is requested. The figure on the right is the same model, with the only difference being that X_1 does not satisfy its validity bound. Therefore, the validity gate has a low output and no log is performed when the log is requested.

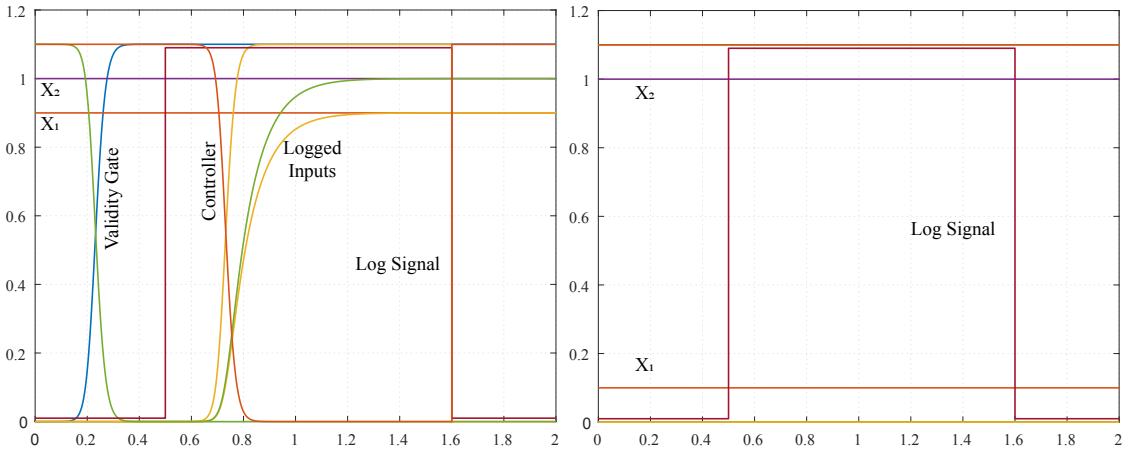

Figure 5.5 Two simulations of the Logging Device Conditional on Validity.

5.3 Further Applications

We now present some further applications of the logging devices discussed in this chapter.

5.3.1 Logging Multiple Devices

Both devices discussed for logging can be applied to multiple devices in a system. Consider two systems A and B that contain m and n signals to be logged, respectively. To take a log of their states, we construct a Follower CRN for each signal in $m \cup n$ and a single Log Interpreter. Upon receiving a log request, all desired signals in A and in B will be logged.

Similarly, for a valid only log device, we construct a Follower CRN for each signal in $m \cup n$. In addition, we construct validity detectors based on the validity properties of each system. We have two choices, either to use a single validity gate and log interpreter to cover both systems, or to use a validity gate and log interpreter for each system. In the first case, a device will only be logged if both devices are valid, while in the second case, each device can be logged independently.

5.3.2 Logging a History of a CRN

We can construct a history of logs using the components discussed previously. For each signal being monitored, we construct two Follower CRNs, A and B, that use different catalysts, i.e. L_1 and L_2. We construct two Log Interpreters, outputting L_1 and L_2 respectively, that take in different inputs, $logSignal$ and $hist$ respectively. We assign A to copy the desired signal and B to copy the output of A. By ensuring that we never send both $logSignal$ and $hist$ at the same time, we will maintain one history log. We can then chain such devices to create a finite length log.

A consequence of our operating in the deterministic mass action domain is that all Follower CRNs are always active, regardless of how slow they are. Therefore, the stored log signal will degrade over time. However, since we can control the concentrations of the catalysts L_1 and L_2, and make them as small as desired, we can control the rate of deterioration.

5.3.3 Checkpoint and Rollback

With a log of a molecular program, we have one or more saved states that the system realized at a previous point in time. When an error occurs, it may be desirable to revert to a previous state that was known to be working. The creation of checkpoints is already handled by either of the log devices. Attaching a number of history devices allows the storage of multiple checkpoints.

In order to rollback, the molecular program must be able to load the desired state on command. All the components required for this behavior have already been described here. We utilize log devices and log interpreters. As stated previously, the log devices copy the concentration of one molecular species into the concentration of another. We used them to copy the monitored signals' concentrations into the copy signals. In order to load a checkpoint, we need to create another log device, called a **Rollback device**, where the signals that have been logged, each C_i, are the signal to be copied and each monitored signal,

X_i are the copied signals. When the rollback device is activated, it will copy the saved values back into the monitored signals, resetting the current state to the saved state. The rollback device requires its own unique activation signal, so that it will only be activated when desired. A log controller can be used to manipulate the activation signal to be below ϵ or above $1 - \epsilon$ as desired.

When a history of logs is stored, a unique rollback device and unique log controller must be created for each log stored. That is, if there are k logs stored, k unique rollback devices with k associated log controllers are required for full rollback capability. As long as each rollback device and log controller has a unique log signal, only the desired checkpoint will be rolled back at a given time.

CHAPTER 6. CONCLUSIONS

In this thesis, we presented a number of devices that aid in the development of robust and correct molecular programs. The devices presented here, along with the processes and methodologies used to develop them, help to ensure that molecular programs remain fault free or that they remain safe even in the case of a fault. Safety is critical in many molecular programs due to their potential in medical applications. Requirements engineering techniques and formal methods, such as simulation and model checking, are essential in the development of safe molecular programs.

The Runtime Fault Detection device (RFD) and its predecessor, the Molecular Watchdog Timer (MWT) can be used to ensure that users and other molecular programs are made aware of any failures that occur. The RFD provides the possibility for a molecular program to autonomously recover from a failure by triggering a recovery response.

The concentration monitor can be used to validate bounds on a chemical species' concentration during runtime. It can be used to ensure that data conforms to specifications either before storage or before the data is passed as input to another molecular program. Reducing the possibility of data corruption leads to a reduction in system faults. We further demonstrated the utility of the concentration monitor by presenting an MWT, constructed with a concentration monitor, for deterministic mass action kinetic chemical reaction networks (CRNs) ensuring that there exists an MWT for the two most common kinetics of CRNs.

We presented a robust CRN NAND gate, using the concentration monitor, that can be nested to create an arbitrary combinational circuit. We proved that the circuits update their

outputs to reflect changes to the inputs correctly and robustly. Using these gates, along with the concentration monitor, we developed two devices to log the state of chemical reaction networks. A log allows devices to recover from failures by rolling back to a state before the failure occurred. A log can be viewed by an external user or another molecular system to monitor for faults and aids in determining the cause of faults.

We hope that the devices and processes discussed here prove useful in the development of molecular programs.

BIBLIOGRAPHY

[1] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[2] Stefan Badelt, Seung Woo Shin, Robert F. Johnson, Qing Dong, Chris Thachuk, and Erik Winfree. A general-purpose CRN-to-DSD compiler with formal verification, optimization, and simulation capabilities. In *Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming*, page to appear, 2017.

[3] Paolo Ballarini and Maria Luisa Guerriero. Query-based verification of qualitative trends and oscillations in biochemical systems. *Theoretical Computer Science*, 411(20):2019 – 2036, 2010.

[4] Paolo Ballarini, Radu Mardare, and Ivan Mura. Analysing biochemical oscillation through probabilistic model checking. *Electronic Notes in Theoretical Computer Science*, 229(1):3 – 19, 2009.

[5] Robert D. Barish, Paul W. K. Rothemund, and Erik Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano Letters*, 5(12):2586–2592, 2005.

[6] Michael A. Boemo, Alexandra E. Lucas, Andrew J. Turberfield, and Luca Cardelli. The formal language and design principles of autonomous DNA walker circuits. *ACS Synthetic Biology*, 5(8):878–884, 2016.

[7] Kuntala Boruah and Jiten Ch. Dutta. Development of a DNA computing model for Boolean circuit. In *Proceedings of the 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics*, pages 301–304, Feb 2016.

[8] Luca Cardelli. Artificial biochemistry. Technical report, Microsoft Research, 2006.

[9] Luca Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23(2):247–271, 2013.

[10] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. In *Proceedings of the 13th International Conference on Computer Performance Evaluation. Modelling Techniques and Tools*, volume 2794 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2003.

[11] Gianfranco Ciardo, Andrew S. Miner, and Min Wan. Advanced features in SMART: the Stochastic Model checking Analyzer for Reliability and Timing. *SIGMETRICS Performance Evaluation Review*, 36(4):58–63, 2009.

[12] Frits Dannenberg, Marta Kwiatkowska, Chris Thachuk, and Andrew J. Turberfield. DNA walker circuits: Computational potential, design, and verification. In *Proceedings of the 19th International Conference on DNA Computing and Molecular Programming*, volume 8141 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2013.

[13] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for biological systems. *International Journal on Software Tools for Technology Transfer*, 17(3):351–367, 2015.

[14] Max Delbrück. Statistical fluctuations in autocatalytic reactions. *The Journal of Chemical Physics*, 8(1):120–124, 1940.

[15] Shawn M. Douglas, Ido Bachelet, and George M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070):831–834, 2012.

[16] Samuel J. Ellis, Eric R. Henderson, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Divita Mathur, and Andrew S. Miner. Automated requirements analysis for a molecular watchdog timer. In *Proceedings of the 29th International Conference on Automated Software Engineering*, pages 767–778. ACM, 2014.

[17] Samuel J. Ellis, Titus H. Klinge, and James I. Lathrop. Robust combinatorial circuits in chemical reaction networks. In *Proceedings of the Sixth International Conference on the Theory and Practice of Natural Computing (To appear)*, 2017.

[18] Samuel J. Ellis, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, and Andrew S. Miner. Runtime fault detection in programmed molecular systems. Technical Report 1710.09494, arXiv.org e-Print archive, 2017.

[19] Samuel J. Ellis, James I. Lathrop, and Robyn R. Lutz. State logging in chemical reaction networks. In *Proceedings of NANOCOM '17*, 2017.

[20] Samuel Jay Ellis. Designing a molecular watchdog timer for safety critical systems. Master's thesis, Iowa State University, 2014.

[21] Joshua Fern, Dominic Scalise, Angelo Cangialosi, Dylan Howie, Leo Potters, and Rebecca Schulman. DNA strand-displacement timer circuits. *ACS Synthetic Biology*, 6(2):190–193, 2017. PMID: 27744682.

[22] Jasmin Fisher, David Harel, and Thomas A. Henzinger. Biology as reactivity. *Communications of the ACM*, 54(10):72–82, 2011.

[23] Lulu Ge, Zhiwei Zhong, Donglin Wen, Xiaohu You, and Chuan Zhang. A formal combinational logic synthesis with chemical reaction networks. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 3(1):33–47, March 2017.

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

[25] Daniel T Gillespie. The deterministic limit of stochastic chemical kinetics. *The Journal of Physical Chemistry B*, 113(6):1640–1644, 2009.

[26] Monika Heiner, David Gilbert, and Robin Donaldson. Petri nets for systems and synthetic biology. In Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016, pages 215–264. Springer, 2008.

[27] Thomas A. Henzinger, Barbara Jobstmann, and Verena Wolf. Formalisms for specifying Markovian population models. In Olivier Bournez and Igor Potapov, editors, *Reachability Problems*, volume 5797, pages 3–23. Springer, 2009.

[28] J. Hetherington, T Sumner, R. M. Seymour, L. Li, M. Varela Rey, S. Yamaji, P. Saffrey, O. Margoninski, I. D. L. Bogle, A. Finkelstein, and A. Warner. A composite computational model of liver glucose homeostasis. I. building the composite model. *Journal of The Royal Society Interface*, 9(69):689–700, 2012.

[29] Allen Hjelmfelt, Edward D. Weinberger, and John Ross. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences*, 88(24):10983–10987, 1991.

[30] Yutaka Hori and Richard M. Murray. Engineering principles of synthetic biochemical oscillators with negative cyclic feedback. In *Proceedings of the 54th IEEE Conference on Decision and Control, CDC 2015*, pages 584–589, December 2015.

[31] Victoria Hsiao, Yutaka Hori, Paul W. K. Rothemund, and Richard M. Murray. A population-based temporal logic gate for timing and recording chemical events. *Molecular Systems Biology*, 12(5):869, 2016.

[32] Titus H Klinge. *Robust and Modular Computation with Chemical Reaction Networks*. PhD thesis, Iowa State University, 2016.

[33] Titus H. Klinge, James I. Lathrop, and Jack H. Lutz. Robust biomolecular finite automata. Technical Report 1505.03931, arXiv.org e-Print archive, 2015.

[34] John Knight. *Fundamentals of Dependable Computing for Software Engineers*. CRC Press, 2012.

[35] M. Kwiatkowska. Challenges in automated verification and synthesis for molecular programming. In *Essays for the Luca Cardelli Fest*, pages 155–170. Microsoft Research, 2014.

[36] Marta Kwiatkowska, Gethin Norman, and David Parker. *Symbolic Systems Biology*, chapter Probabilistic Model Checking for Systems Biology, pages 31–59. Jones and Bartlett, 2010.

[37] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[38] Marta Kwiatkowska and Chris Thachuk. Probabilistic model checking for biology. *Software Systems Safety*, 36:165–189, 2014.

[39] Michael Lachmann and Guy Sella. *The computationally complete ant colony: Global coordination in a system with no hierarchy*, pages 784–800. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[40] Matthew R. Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.

[41] Matthew R. Lakin, Simon Youssef, Luca Cardelli, and Andrew Phillips. Abstractions for DNA circuit design. *Journal of The Royal Society Interface*, 9(68):470–486, 2012.

[42] Matthew R Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.

[43] Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company, 1995.

[44] Xiaowei Liu, Yan Liu, and Hao Yan. Functionalized DNA nanostructures for nanomedicine. *Israel Journal of Chemistry*, 53(8):555–566, 2013.

[45] Robyn Lutz, Jack Lutz, James Lathrop, Titus Klinge, Eric Henderson, Divita Mathur, and Dalia Abo Sheasha. Engineering and verifying requirements for programmable self-assembling nanomachines. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1361–1364. IEEE, 2012.

[46] Robyn R. Lutz, Jack H. Lutz, James I. Lathrop, Titus H. Klinge, Divita Mathur, Donald M. Stull, Taylor G. Bergquist, and Eric R. Henderson. Requirements analysis for a product family of DNA nanodevices. In *Proceedings of the 20th International Conference on Requirements Engineering*, pages 211–220. IEEE, 2012.

[47] Indika Meedeniya, Irene Moser, Aldeida Aleti, and Lars Grunske. Evaluating probabilistic models with uncertain model parameters. *Software & Systems Modeling*, 13(4):1395–1415, 2014.

[48] NASA. Computers in spaceflight: The NASA experience, 1988.

[49] Bashar Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, 2001.

[50] Mitsunori Ogihara and Animesh Ray. Simulating Boolean circuits on a DNA computer. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, RECOMB '97, pages 226–231, New York, NY, USA, 1997. ACM.

[51] Esteban Pavese, Víctor Braberman, and Sebastián Uchitel. Less is more: Estimating probabilistic rewards over partial system explorations. *ACM Transactions on Software Engineering and Methodology*, 25(2):16:1–16:47, 2016.

[52] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[53] Nadrian C. Seeman. Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 99(2):237 – 247, 1982.

[54] Radu I. Siminiceanu and Gianfranco Ciardo. Formal verification of the NASA runway safety monitor. *International Journal on Software Tools for Technology Transfer*, 9(1):63–76, 2007.

[55] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. In *Proceedings of the 14th International Meeting on DNA Computing*, volume 5347 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 2009.

[56] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

[57] Xin Song, Abeer Eshra, Chris Dwyer, and John Reif. Renewable DNA seesaw logic circuits enabled by photoregulation of toehold-mediated strand displacement. *RSC Advances*, 7:28130–28144, 2017.

[58] Guoxin Su, Taolue Chen, Yuan Feng, and David S. Rosenblum. ProEva: runtime proactive performance evaluation based on continuous-time markov chains. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 484–495, 2017.

[59] Guoxin Su and David S Rosenblum. Asymptotic bounds for quantitative verification of perturbed probabilistic systems. In *Formal Methods and Software Engineering*, volume 8144 of *Lecture Notes in Computer Science*, pages 297–312. Springer, 2013.

[60] T. Sumner, J. Hetherington, R. M. Seymour, L. Li, M. Varela Rey, S. Yamaji, P. Saffrey, O. Margoninski, I. D. L. Bogle, A. Finkelstein, and A. Warner. A composite computational model of liver glucose homeostasis. II. exploring system behaviour. *Journal of The Royal Society Interface*, 9(69):701–706, 2012.

[61] The MathWorks, Inc. MATLAB SimBiology package Release 2017b, 2017. Natick, Massachusetts, United States.

[62] Anupama J. Thubagere, Chris Thachuk, Joseph Berleant, Robert F. Johnson, Diana A. Ardelean, Kevin M. Cherry, and Lulu Qian. Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nature Communications*, 8, 2017.

[63] N G Van Kampen. *Stochastic Processes in Physics and Chemistry*. North-Holland Personal Library. Elsevier Science, 1992.

[64] Axel van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[65] Michael W. Whalen, Andrew Gacek, Darren D. Cofer, Anitha Murugesan, Mats Per Erik Heimdahl, and Sanjai Rayadurgam. Your "what" is my "how": Iteration and hierarchy in system design. *IEEE Software*, 30(2):54–60, 2013.

[66] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

[67] John C. Wooley and Herbert S. Lin. *Catalyzing Inquiry at the Interface of Computing and Biology*. National Academies Press, 2005.

[68] Boyan Yordanov, Christoph M Wintersteiger, Youssef Hamadi, and Hillel Kugler. SMT-based analysis of biological computation. *NASA Formal Methods*, 7871:78–92, 2013.