2014

Designing a molecular watchdog timer for safety critical systems

Samuel Jay Ellis
Iowa State University

Recommended Citation

Ellis, Samuel Jay, "Designing a molecular watchdog timer for safety critical systems" (2014). *Graduate Theses and Dissertations*. 13848.
https://lib.dr.iastate.edu/etd/13848

Designing a molecular watchdog timer for safety critical systems

by

Samuel Jay Ellis

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

James Lathrop, Co-major Professor

Robyn Lutz, Co-major Professor

Stephen Gilbert

Iowa State University

Ames, Iowa

2014

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation grants 0916275 and 1247051.

I would like to thank my Co-Major Professors, Dr. James Lathrop and Dr. Robyn Lutz. They provided support and guidance while I learned about safety-critical systems and molecular programming. Their knowledge was a great help as I performed my research and my writing and presentation skills have both improved thanks to their advice.

Many thanks go to the members of the Molecular Programming Group in the Computer Science Department at Iowa State University. Their knowledge of the field provided interesting discussions and good feedback on my research. A special thanks goes to Titus Klinge. He introduced me to the field of molecular programming and guided me while I learned about model checking and CRNs. Without him my research probably would have been along a different path.

My thanks go to Eric Henderson's Lab at Iowa State University and to Divita Mathur for teaching me about biology. I appreciate their patience and understanding as I learned how to perform experiments in a laboratory setting.

I would also like to thank my family and friends for their support. They provided encouragement and understanding while I spent my time at Iowa State University. Most notably my mother (Cathy Ellis) and my brother (Matthew Ellis) who listened when I needed someone to talk to.

ABSTRACT

The programming of matter (molecular programming) is often realized through DNA strand displacement (DSD). Computations and algorithms using DNA strand displacement are often modeled using chemical reaction networks (CRN) that abstract away DNA specific reactions and terminology. These CRNs can yield complex behavior similar to computer programs. If these molecular programs are further used in vivo to effect cell change or fight disease, the molecular program becomes a safety critical system.

In this paper we propose and analyze a molecular watchdog timer, based on a software component often used to monitor the health of a critical system. Using goal-oriented requirements engineering and a stochastic CRN model (SCRN) we design a watchdog timer using two components, a delay clock and a threshold detector. The models are directed, informed, and verified by use of a probabilistic model checker.

Analyzing requirements and the design uncovered several defects that were addressed in subsequent iterations. During each phase, the system was modeled using formal verification tools and simulations to verify correctness of the model. It was found that this iterative methodology with verification was potent at illuminating requirement and design flaws. In addition, the final verified model helped determine specific parameters and molecules for initial biological experiments.

CHAPTER 1. INTRODUCTION

Recent years have seen a growing interest in molecular programming, programming matter at a nanoscale level. Molecular programming draws on computer science and biology to create molecular systems. Instead of using circuits and wires, a molecular system uses structures such as DNA, RNA, or other chemical molecules to perform computational tasks. Molecular systems have been used to create boolean circuits [23], perform digital signal processing [12] and build molecular robots [6, 26] at the nanoscale level.

A molecular system can be modeled through the use of chemical reaction networks (CRNs). Stochastic chemical reaction networks (SCRNs) model systems that use finite numbers of molecules through stochastic chemical kinetics [8]. Mass action chemical kinetics can be used to model systems in terms of molecular concentrations, i.e., what percentage of a solution is made up by each species. One way to implement CRNs is to use DNA Strand Displacement (DSD). Molecular systems are inherently probabilistic, having an amount of uncertainty in all interactions. This adds complexity to their construction and use because nothing is guaranteed.

SCRNs are a useful programming paradigm. The capabilities of SCRNs have been thoroughly analyzed, and they have been compared to a number of other computational models [3]. SCRNs are Turing Universal, able to compute any function a Turing Machine can compute, given an arbitrarily small probability of error [28].

As research expands the molecular programming field and finds additional applications for it, potential safety-critical applications of molecular programming emerge. Previous work has shown some medical applications of nanoscale assemblies such as cell targeting, drug delivery, bioimaging, and vaccine deployment [5].

Computer scientists can bring software engineering and safety awareness techniques into the molecular programming field, providing a foundation for engineering safety-critical molecular systems. Systems normally associated with computer science have been adapted into molecular systems; for example, robots become molecular robots [6, 26] when they are converted into molecular systems and have potential uses in safety-critical applications such as using a molecular robot to deliver a drug only to cells that meet specified conditions. Drug delivery is safety-critical if improper delivery can have harmful effects.

Watchdog timers, described in 2.1., are a common safety mechanism used in safety-critical systems to monitor an external system and issue an alarm if it fails [20, 15]. This paper proposes the design of a molecular watchdog timer to be used in safety-critical molecular systems. We perform the requirements engineering of the watchdog timer using goal-oriented requirements modeling [17, 18] and create a model based on the requirements to analyze the feasibility of the molecular watchdog timer. The process of creating a molecular watchdog timer provided insight into challenges involved in creating a molecular watchdog timer, supported reasoning about what is needed to achieve one that operates as intended, and led to a more complete specification of requirements.

We describe the incremental refinement and elaboration of the watchdog timer and explore alternatives using a model checker. We present the final design, a unary counter-based watchdog timer, and support this decision using automated model checking. We use the Probabilistic Symbolic Model Checker (PRISM)[16] and the Stochastic Model Checking Analyzer for Reliability and Timing (SMART)[2] to perform verification of the models for the molecular watchdog timer.

CHAPTER 2. BACKGROUND

2.1 Watchdog Timer

A watchdog timer is a device used in safety-critical systems to inform either a user or another system when a specific system fails. A watchdog timer can be as simple as a counter with one input and one output. The counter increments until a specified value and sends a signal on its output to represent an alarm. Upon receiving input, the counter is reset to zero and begins counting again. Nancy Leveson describes a watchdog timer as “a timer that the system must keep restarting. If the system fails to restart the timer within a given period, the watchdog initiates some type of protection action” [20]. John Knight describes a watchdog timer as “a countdown timer that is set by an application and which raises an interrupt when it expires” [15]. It is a device designed to monitor a single external system and take a specified action, usually issuing an alarm, when the system fails.

Figure 2.1 and Figure 2.2 show the two possible outcomes of a watchdog timer where the counter threshold value is set to 4. A watchdog timer has a delay component, an alarm component, and detects the presence of an external heartbeat. In the absence of a heartbeat for some time period, the alarm is triggered. The heartbeat acts to delay the alarm for another time period by resetting the counter. If another heartbeat is not input into the system during the new time period, the alarm will go off. The heartbeats are added at specified intervals, to keep the alarm from being issued. If the alarm is issued, it means that the monitored system is unable to release a heartbeat and, therefore, is in a failed state.

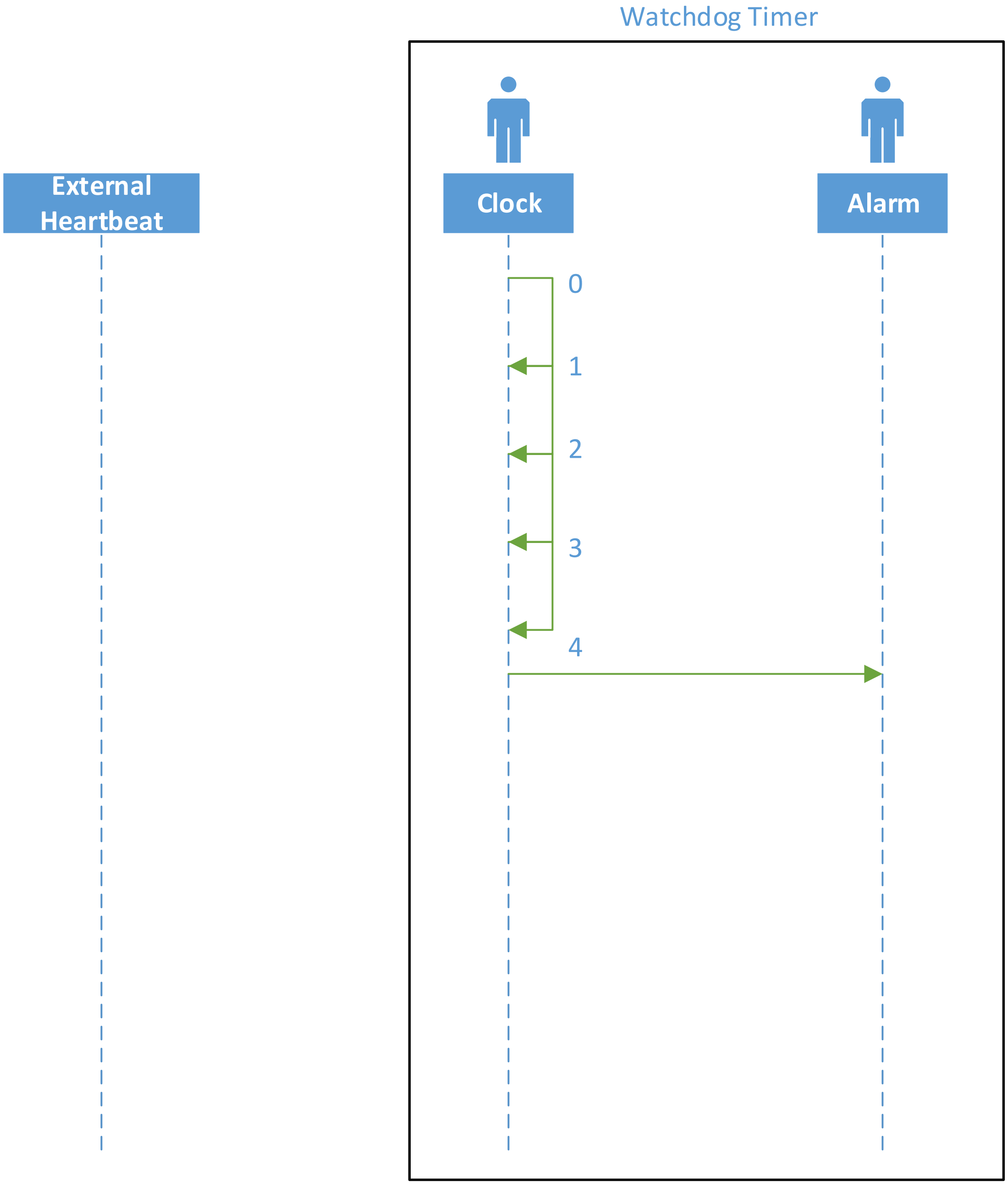

Figure 2.1 A watchdog timer that does not receive a heartbeat. It counts until reaching its specified value and sends out an alarm.

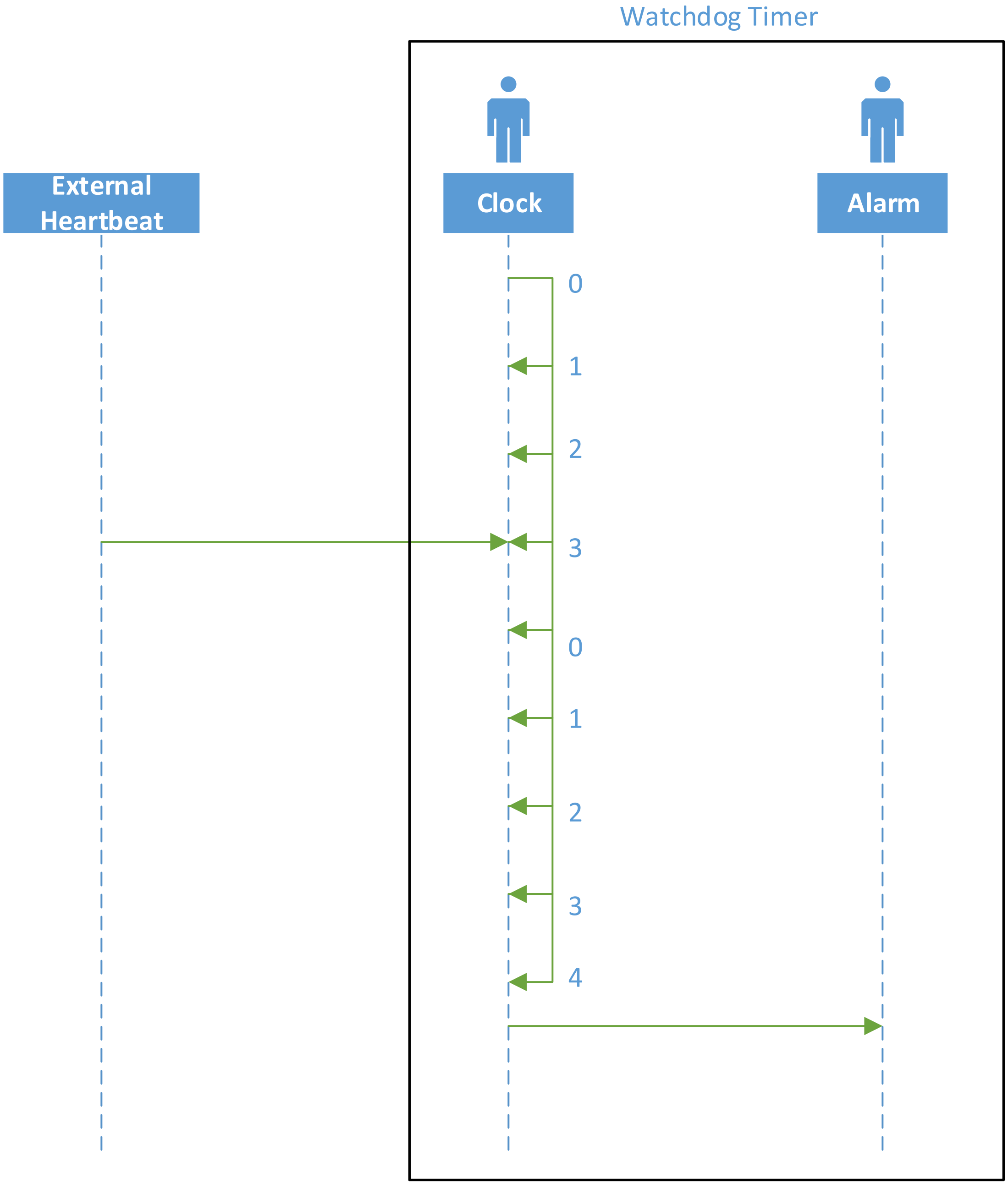

Figure 2.2 A watchdog timer that receives a heartbeat at 75% of its specified value. It resets its counter to zero and begins counting again. It does not receive another heartbeat so it releases an alarm upon completion.

2.2 Stochastic Chemical Reaction Networks (SCRNs)

Initial research on SCRNs was performed by Max Delbrück [4]. Further developments in SCRNs performed by McQuarrie[24], Erdi & Tóth[7], Gillespie[8, 9], and van Kampen[30].

We adopt the terminology and syntax used by Soloveichik in [27] to describe SCRNs. SCRNs consist of a set of species S and a set of reactions R. The current state of S is defined as an integer vector. For example, an SCRN consisting of three species, (A, B, C), may have an associated vector $(10, 5, 4)$, meaning that the chemical solution currently consists of 10 molecules of A, 5 molecules of B, and 4 molecules of C. The set R consists of chemical reactions which have three components: a set of reactants, a set of products, and a rate constant k [27]. The reactant set defines a set of inputs and restrictions to a reaction. A reaction $r \in R$ can only occur if the state of S contains all the elements in the reactant set of r. When a reaction r occurs, the reactant set is subtracted from the state of S. The product set is the output of the reaction. When a reaction r occurs, the product set is added to the current state of S. A rate constant k is used to describe how quickly or likely a reaction is to occur in the event its' reactants meet.

Given that S has $(10, 5, 4)$ elements of the species (A, B, C) respectively, consider an example reaction r:

$$r : A + B \overset{\text{k}}{\rightarrow} C \tag{2.1}$$

r has a reactant set $(1, 1, 0)$ and a product set $(0, 0, 1)$ with a rate constant k. Given the current state of S, reaction r can proceed and will change the current state from $(10, 5, 4)$ to $(9, 4, 5)$.

In our work, we consider two orders of reactions: unimolecular reactions and bimolecular reactions. A reaction is unimolecular if it contains exactly one reactant,

$$r : A \overset{\text{k}}{\rightarrow} B \tag{2.2}$$

and is bimolecular if it contains exactly 2 reactants,

$$r : A + B \overset{\text{k}}{\rightarrow} C \tag{2.3}$$

Some research has included higher order reactions in the model[30], but the usefulness of higher order reactions is still debated [27].

Since SCRNs contain integer numbers of molecules, a Continuous-Time Markov Chain (CTMC) can be used to describe the state of an SCRN and how the reactions change the state [30, 24, 8, 9, 7]. A CTMC has a set of states which are directly mapped to the possible states in a SCRN. That is, for every possible state s in an SCRN S, there is a state in a CTMC C that represents s. We call the set of states in C the statespace. Each reaction r in S is used to move between states in C. From an initial state s_0 we apply each reaction that can occur to generate a new set of states. From each new state, we repeat the process until we cannot apply anymore reactions, building the entire statespace of C.

Given a current state s and a volume V of the solution, each reaction r has a propensity, or the likelihood of reaction r occurring in the current state [27]. If a reaction r is unimolecular with a reactant S_i, the ith element in the vector S, then its propensity is defined as

$$S_i k \tag{2.4}$$

where S_i is the number of r's reactant and k is the rate constant of r. If r is bimolecular with reactants S_i and $S_{i'}$, where $i \neq i'$, the reactants must collide within the solution in order to react. To correctly model the collision, the volume of the solution must be included in the propensity of r:

$$\frac{(S_i S_{i'} k)}{V} \tag{2.5}$$

Where S_i and $S_{i'}$ are the number of each of r's reactants, k is the rate constant, and V is the volume of the solution. If $i = i'$, r requires two of the same reactant, the propensity of r is: [27]

$$\frac{(S_i(S_i - 1)k)}{2V} \tag{2.6}$$

Given a CTMC C for an SCRN S, it is possible to calculate the probability that a reaction r is the next reaction to occur in the solution. Let $a_r(S)$ be the propensity of a reaction r given the state S. The probability of a reaction r occurring next is: [27]

$$\frac{a_r(S)}{\sum_i a_i(S)} \tag{2.7}$$

which is the propensity of reaction r divided by the sum of the propensity of every reaction in R.

2.3 Goal-Oriented Requirements Engineering

We used the Goal-oriented Requirements Engineering process defined in [17, 18] to define our requirements. Van Lamsweerde defines a goal as “a prescriptive statement of intent the system should satisfy through cooperation of its agents” [17]. A goal models an aspect of the target system in a simple and concise manner. A simple view of the requirements can be useful for explaining the system to stakeholders. A goal defines a task the system needs to complete. We define a top-level goal that represents the task. We then refine each goal into subgoals and repeat the process until we have leaf level goals. [17]

Goal-Oriented Requirements Engineering distinguishes between behavioral goals and soft goals. Behavioral goals describe intended system behavior and can be Achieve goals or Maintain/Avoid goals. An Achieve goal is a target that must be completed. A Maintain goal is a condition that needs to be maintained during operation. An Avoid goal is a condition that needs to be avoided. Soft goals describe preferences between alternative system behaviors and are used to compare alternative options.

A goal model is an AND/OR graph that shows how each high level goal is satisfied by the subgoals (goal refinement) and how subgoals work together to satisfy high level goals (goal abstraction). An AND link shows that a parent goal can only be satisfied if all the subgoals are satisfied. An OR link shows that a parent goal can be satisfied if at least one of the subgoals is satisfied.

Goal models provide a number of advantages to the Requirements Engineering (RE) process [17, 18]:

- Refining goals naturally produces the requirements. Goals give a criterion for requirements, i.e., we create a set of requirements to satisfy a goal or set of goals.

- The structure of a goal model provides a set of satisfaction arguments, showing which subgoals are needed to complete a parent goal. Chaining arguments together can show which leaf level goals need to be satisfied to satisfy high-level goals.

- Goal models support traceability. Low-level goals provide requirements, while high-level goals show the need for those requirements.

Defining leaf-level goals also identifies the requirements for a system. We can then assign agents to complete each leaf-level goal. Van Lamsweerde defines an agent as "an active system component having to play some role in goal satisfaction..." [17]. We use satisfaction arguments to determine which agents must work correctly to satisfy the top level goal. Each agent either describes a subsystem that must be created to satisfy its assigned goals or is an environmental agent that satisfies an assigned goal.

Previous work has shown a successful application of goal-oriented requirements engineering to the molecular programming domain. Lutz et al. used goal modeling to analyze the requirements of a product family of DNA pliers [22]. DNA pliers are used to detect molecules of a specific species in a solution and are created using DNA origami, which is folding DNA strands to create an object. They use goal models to capture satisfaction requirements for the pliers. The paper also extends the goal model to include threshold functions in addition to AND/OR nodes [21]. Due to the probabilistic nature of molecular systems and the fact that many molecular systems contain a very large number of devices, it is infeasible to expect or require all of the devices to complete their tasks. Threshold nodes are satisfied when a specified number of devices satisfy the goal. For example, if 70% of the pliers report detection of the target molecule of concern, the system reports that the molecule is in the solution. Goal modeling revealed failure states in the design of the pliers.

2.4 Toehold-Mediated DNA Strand Displacement

Toehold-mediated DNA Strand Displacement can be used to program the behavior of DNA systems. Strands consist of single strands of nucleotides, the building blocks of DNA, and can bind to each other to form dual-stranded complexes. Each strand consists of domains, sequences of nucleotides, where short domains are called toeholds [32, 31]. Toehold domains are used to give a DNA strand a starting point to displace another strand, where displacement means a single strand of DNA is replaced by a new strand of DNA. A longer toehold has a faster binding rate than a shorter toehold which allows you to change the rate constants of your DSD reactions by increasing and decreasing the length of the toehold domains.

Toehold-mediated DNA strand displacement reactions can approximate any generic CRN [29] so they are useful in implementing a CRN. CRNs are a high level programming language for molecular systems, such as Java or C++ are high level programming languages for computers. CRNs can be compiled into generic DNA strands, much like a Java program can be compiled into assembly. The generic DNA strands can then be converted into real DNA strands with specified nucleotide sequences in the same manner that an assembly program can be compiled into machine code. CRNs combined with toehold-mediated DNA strand displacement provide a high level programming language that can be converted into the base code of a molecular program.

2.5 Related Work on Molecular Clocks, Counters, and Alarms

Molecular clocks have been created previously to manage computations. We look at some molecular clock designs and explain why they are unsuitable for a watchdog timer.

Soloveichik et al. created (designed and implemented in DNA) a clock to control a molecular Register Machine which is a model of computation using a fixed set of registers to store data and a set of instructions that can be applied to the registers [28]. They created a CRN that emulates a random walk. Molecules $C_1 - C_n$ interact with catalysts A and A^*, molecules that are not consumed during their reactions, to move forward and backward through the n steps of the walk. The computation reactions interact only with C_1, meaning computations can only occur when C_1 is present in the solution. We use the term clock phase to describe the periods of time when C_1 is present to more easily describe the computation process. The duration of the clock phase and how often they occur is controlled by the random walk. The clock phases duration and occurrence can be manipulated by changing the length of the random walk, changing the number of A's or A^*'s present in the solution, or by changing the rates at which the increasing and decreasing reactions occur.

Whereas the clock phases can happen at different intervals and last for different lengths of time with varying probabilities, we want the clock for the watchdog timer to delay for a specified time, and to fully reset upon a heartbeat input.

Cook, Soloveichik, Winfree, and Bruck created (designed and implemented in DNA) a clock module to control the operation of a Register Machine [27]. They created a set of sequential reactions which released a molecule C upon completion, then reset to the initial state. The Register Machine reactions required C as a catalyst to react, which created a delay between reactions as a new C was generated. The clock was designed so that the time needed to generate a C slowly increased over time, reducing error by allowing more time for each computation. However, for our purpose, a watchdog timer needs a consistent amount of time until an alarm is released. The increasing delay makes the clock module unworkable for our envisioned watchdog timer.

Hjelmfelt, Weinberger, and Ross use an oscillating catalyst to synchronize reactions [10].They use reactions to alternate the concentration of a catalyst ε between high and low amounts. When ε's concentration is high, an "on" phase, it interacts with other molecules to drive computations forward. When ε's concentration is low, an "off" phase, it has a low probability of interacting with other molecules. This oscillator design does not suit the needs of the watchdog timer. There is not enough control over the presence of the catalyst, the periods of time the catalyst is available for computation, to guarantee with high probability that reactions will only occur during "on" phases.

Kharam, Jiang, Riedel, and Parhi created a binary counter with chemical reactions and a three-phase oscillator to act as a clock [14]. Later work by Jiang, Riedel, and Parhi utilized delay elements to store information in between computation cycles for digital signal processing [12]. They used a three-phase oscillation to synchronize their computations in both papers. Their oscillator design creates a delay between computation steps to ensure that each computation completes. They perform digital signal processing operations such as scalar multiplication, adder, and fan-out during the computation phase.

We describe the following two related works in more detail because both were used in creating the molecular watchdog timer.

First, Jiang, Riedel, and Parhi created a four-phase oscillating clock [11], which they later generalized into an n-phase oscillator [13]. The primary components of the clock are the absence indicators r, g, b, and y; and the phase indicators R, G, B and Y (which stand for the four phases

Red, Green, Blue, and Yellow). The absence indicators are generated at a slow and constant rate, but are consumed by their associated phase indicators at a fast rate. Therefore, if a phase indicator is present, it is unlikely that its associated absence indicator will be present. The presence of a phase indicator is used to represent the clock phase; for example, a large number of R's represents the red phase. Phase indicators are consumed to produce the phase indicator for the succeeding phase upon reacting with the preceding phase's absence indicator; for example, $R+y \xrightarrow{\text{k}} G$ (the Red phase indicator molecule and the preceding Yellow phase absence indicator molecule react to create a succeeding Green phase molecule). Absence indicators restrict phase transitions. Since a phase indicator needs its predecessors absence indicator to transition, and the absence indicators are only present in the absence of the associated phase indicator, a phase transition has a low probability of beginning until the previous phase transition has completed. This restriction prevents opposite phases from occurring at the same time; i.e., from a red and blue phase happening simultaneously. Opposite phases are therefore disjoint, allowing each computation phase to complete before the next begins. The adjacent phases help keep the opposite phases disjoint by adding a delay in between the opposite phases.

Second, a molecular ladder, also known as a frog in the well in the language of [1] is used to develop the watchdog timer. The ladder consists of k rungs and a molecule to push the ladder upward. Rungs function in the same way as rungs on an actual ladder with the exception that when you move up the ladder, you move up a single rung, but when you move down the ladder, you fall all the way to the bottom rung. The ladder starts with a single molecule of rung 1, call it L_1, and a single molecule to push the ladder upward, U. U "climbs" the ladder by repeatedly reacting with a rung molecule and converting it into the next "higher" rung molecule. Thus,

$$L_i + U \xrightarrow{\text{k}} L_{i+1} + U \tag{2.8}$$

where L_i is the ith rung of the ladder.

The ladder also contains a reset species. A rung molecule that reacts with the reset species is reset to a rung 1 molecule, analogous to falling to the bottom of the ladder. The ladder has a low probability of reaching the top rung while the reset species is present. The use of the ladder will be explained further in section 3.2 of this paper.

An alarm (as in a watchdog timer) is not useful unless its presence can be detected. Molecular systems consist of a number of devices larger than 10^{10}. With such a large number of devices, it is impossible to expect all of the devices to work properly. For the watchdog timer, an alarm is considered active when a threshold, a percentage of specified number, of the watchdog timers have completed. Qian et al. introduce a threshold reaction in their seesaw circuit design in order to delay a computation reaction until a threshold number of molecules have been detected [25]. A DNA complex binds with the target strand and converts it into waste at a fast rate. Increasing the initial concentration of the threshold complex increases the amount of the target molecule that must be added into the system before the threshold is met. Increasing the length of the threshold toehold domains increases the rate of the threshold reaction. The accuracy of the threshold delay increases as the threshold rate increases over the computation rate. The threshold circuit in [25] does not suit our needs. Toehold length manipulation grants a coarse grain control over the accuracy of the threshold, meaning that large changes in the accuracy occur when you change the toehold length. Working with safety critical systems, we require a more fine grain control of our threshold system, meaning we want small changes in accuracy with small changes to the device.

CHAPTER 3. OUR APPROACH

3.1 Goal Modeling

We define the following basic goal that a watchdog timer should achieve.

If the monitored system does not provide the appropriate signal within a specified time, then issue an alarm.

The purpose of a watchdog timer is to provide an alarm if the monitored system does not provide timely input. If the system does not provide input within the time period, a failure may have occurred and corrective actions may need to be taken by a user or another system.

Using this high-level intent as a starting point, we develop a goal diagram for the watchdog timer, following the goal-oriented requirements engineering process described in [17, 18]. Figure 3.1 shows the resulting initial goal diagram for a watchdog timer. The top level goal, Achieve[Alarm if the monitored system does not provide a signal within the specified time t], abbreviated as Achieve[Watchdog Timer], must be satisfied for our system to be a watchdog timer. We refined the top level goal into subgoals and then further refined those subgoals as needed.

To achieve the top level goal of our diagram, the system must satisfy each AND refinement of the goal diagram. For instance, to satisfy the top-level goal, Achieve [Watchdog Timer], the system must satisfy the three subgoals under the AND refinement: Avoid[Alarm if time<t], Achieve[If time>t alarm], and Achieve[Reset time on Heartbeat]. Since all of the second-level refinements are ANDs, the system must satisfy all subgoals to satisfy the top level goal.

Once a subgoal cannot be refined further, it is assigned an agent. The agents work to satisfy their assigned subgoals. For example, we assign a Molecular Clock to Achieve [Delay for t time], meaning the Molecular Clock agent must implement a delay period of time t.

Two agents are needed to satisfy all of the goals in the diagram: Molecular Clock and

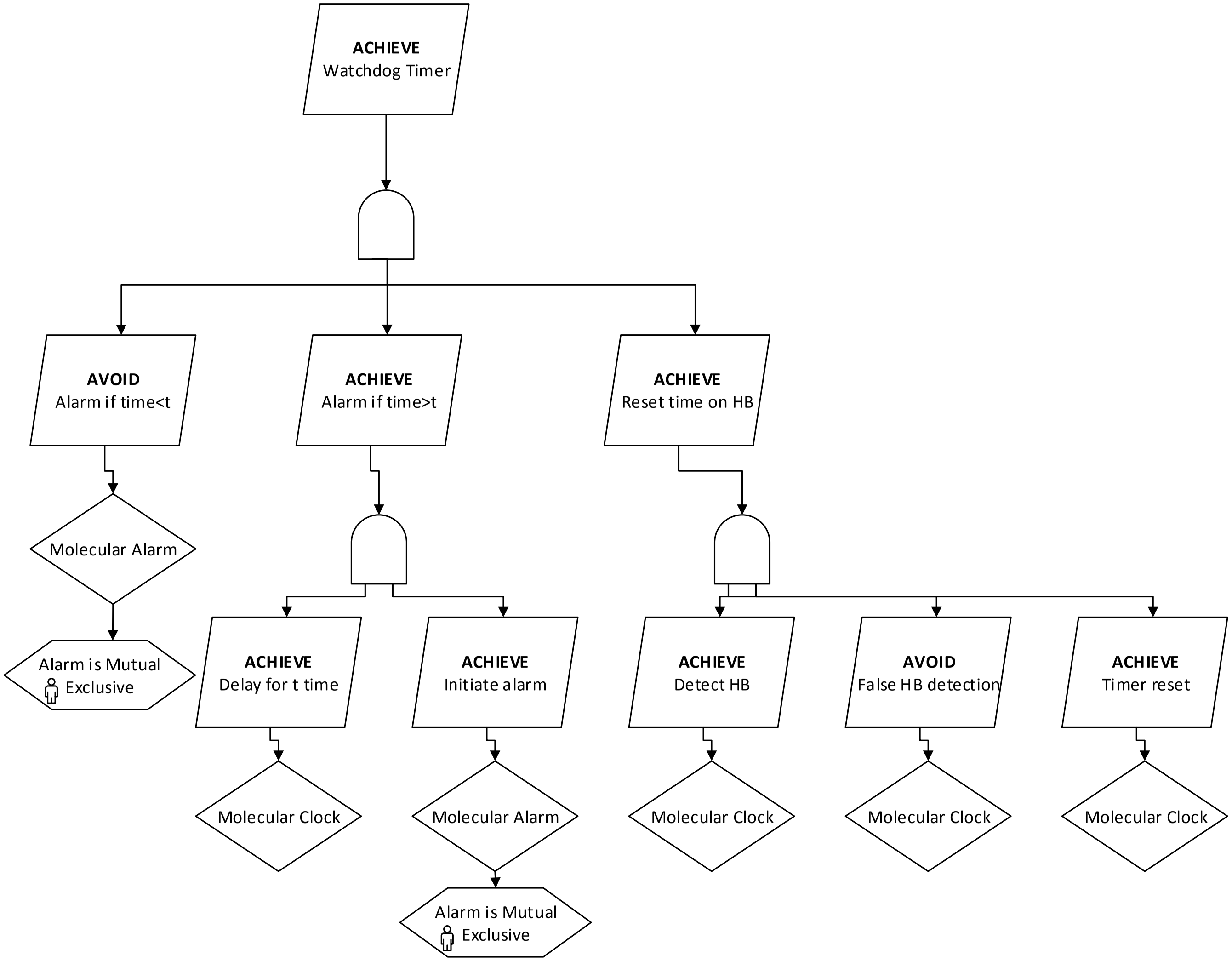

Figure 3.1 Goal Diagram for a Molecular Watchdog Timer

Molecular Alarm. We define the Molecular Alarm agent as a single species that is released into the system upon completion of the delay phase. We define the Molecular Clock agent as a device with a delay phase that is reset upon receiving a heartbeat input. We elaborate on the Molecular Clock in the following sections.

3.2 Obstacles to Achieving a Molecular Watchdog Timer

Obstacle analysis is "aimed at identifying, assessing, and resolving the possibilities of breaking assertions in the goal model" [17]. The next step in creating a molecular watchdog timer is to identify obstacles that could keep the leaf subgoals from being satisfied. Obstacles were identified using PRISM and SMART models, and with simulation using SimBiology, a Matlab package for modeling CRNs. We describe below four major obstacles identified during the requirements analysis process along with their resolutions.

3.2.1 Obstacle 1. Incorrect Clock Value

While designing the molecular watchdog timer, we discovered the obstacle: incorrect clock value. This is an obstacle to the leaf goal Achieve[Alarm if time>t]. Simply put, when trying to maintain a clock, there are many errors that may appear. If an error should occur that alters the value of the clock, the clock is no longer reliable and the device fails as a watchdog timer. We describe alternative clock approaches proposed in the literature and evaluate their usefulness in avoiding the incorrect clock value obstacle.

3.2.1.1 Candidate Resolution 1. 4-Phase Clock with Binary Counter

In software engineering, we use past knowledge to help solve current problems. Following this approach, we considered the use of a binary counter to create a timer. Counting from zero to a target value creates a delay. To create the delay phase for the watchdog timer, we define a molecular binary counter. Our counter starts at zero and counts to a target value in the same manner as a software based binary counter. Upon reaching the target value, it initiates the alarm. When a heartbeat from the external system is received, the counter is reset to zero. The reactions for a 2-bit molecular binary counter are shown in Figure 3.2.

$$\begin{array}{rr} \underline{\textbf{Red Phase Reactions}} & \underline{\textbf{Blue Phase Reactions}} \\ R + B'^{0}_{0} \xrightarrow{\text{k}} B^{+}_{0} R & B + B^{0}_{0} \xrightarrow{\text{k}} B'^{1}_{0} + B \\ R + B'^{1}_{0} \xrightarrow{\text{k}} B^{1}_{0} + R & B + B^{1}_{0} \xrightarrow{\text{k}} BN + B \\ R + B'^{0}_{1} \xrightarrow{\text{k}} B^{0}_{1} + R & BN + B^{0}_{1} \xrightarrow{\text{k}} B'^{1}_{1} + B'^{0}_{0} \\ R + B'^{1}_{1} \xrightarrow{\text{k}} B^{1}_{1} + R & BN + B^{1}_{1} \xrightarrow{\text{k}} B'^{0}_{1} + B'^{0}_{0} \end{array} \tag{3.1}$$

Figure 3.2 Molecular Binary Counter Reactions

A binary counter requires a clock to enforce consistent updates to the value. We use the four-phase synchronous clock designed by Jiang et al. [11], described in Section 2.5, as our clock. We use the red-phase molecules R as catalysts to perform memory transfer between each counter update. We use the blue-phase molecules B as catalysts to perform each counter update.

The clock design, the binary counter with the four-phase synchronous clock, works as follows. B^j_i denotes that bit i's value is j. A primed version B'^j_i denotes that B^j_i was output from a counter update and needs to be converted during a red phase before it can be used in a counter update again. The use of the primed and unprimed molecules is intended to enforce that only a single update of the counter completes during each blue phase.

3.2.1.2 Candidate Resolution 2. 4-Phase Ladder-based Clock with Binary Counter

We use the same binary counter from part a) with a different clock to create an alternative clock design for the watchdog timer. We use four ladders to create a four phase clock that controls the binary counter. Each phase consists of a single ladder that uses a corresponding phase indicator as its reset species; i.e. red-phase uses R to reset the ladder. Phase indicators transition into the next phase in a similar way to the four-phase synchronous clock except they are first converted to an intermediate stage and are only converted into the phase indicators for the next phase when the ladder completes. Waiting until the ladder completes allows for more control over the probability of two phases occurring at the same time; that is, having

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

Figure 3.3 Unary Ladder-based Clock Reactions

phase indicators for two phases present in the solution at the same time and creating crosstalk, two signals communicating to the device at the same time. Increasing the number of rungs in the ladder would make it harder for the ladder to complete with a small number of phase indicators, decreasing the probability of crosstalk.

3.2.1.3 Candidate Resolution 3. Unary Ladder-based Clock without Binary Counter

The last alternative we define is a unary counter created from a single ladder. A single ladder creates a delay. It takes time for each rung reaction to happen; i.e., it takes time to climb up the ladder. By adding or removing rungs, we can increase or decrease the delay respectively. Upon reaching the top rung, the ladder initiates the alarm. When a heartbeat is input into the system, it is consumed as it resets the ladder to the first rung, starting the delay over from the beginning. The reactions for the unary ladder-based clock are shown in Figure 3.3. As stated in 2.5, the L_i represent each rung of the ladder and U forces the ladder to climb upward. Y is used to represent the alarm and H is the signal released by the heartbeat to reset the ladder to L_1.

3.2.1.4 Resolution and Rationale for Incorrect Clock Value

The resolution to the incorrect clock value obstacle was a relatively easy choice. The clock designs provided in 3.2.1.1 and 3.2.1.2 both have a non-trivial probability that the clock value is incorrect. The phase-based designs with a binary counter maintain the possibility that a phase indicator or a binary counter molecule will be "left behind" (does not take part in any reactions during its phase) which allows the value of the clock or the binary counter to become invalid. If a phase indicator is left behind, the binary counter may be able to perform multiple updates if a red-phase and a blue-phase indicator are both present. If a binary counter molecule is left behind, then there are two conflicting values for a bit of the counter, which invalidates the entire counter. An example of the left behind error is shown in Figure 3.4. Figure 3.4 shows a simulation of the 4-phase synchronous clock with the 2-bit binary counter. This simulation was performed in Simbiology, a Matlab package. The red and blue phase indicators are graphed along with the values of each bit. We observe that the binary counter works initially but, because of the "left behind" property, the value of each bit loses accuracy.

The unary counter (ladders used as a counter) does not have the same issue. Since there are no phases, there is no possibility that a phase indicator will be left behind. Since there is no binary counter, there is no possibility for a binary counter molecule to be left behind. The simplicity of the unary clock eliminates the possibility of the incorrect clock value obstacle.

3.2.2 Obstacle 2. False Positive Alarm

An obstacle that interferes with correctly alarming is initiating a false positive alarm. This is an obstacle to the leaf subgoal Avoid[Alarm if time<t]. If an alarm is initiated incorrectly, then the alarm is invalid. In order to satisfy the watchdog timer alarm requirements, we need to limit or eliminate any false positive alarms. We describe alternative alarm approaches and investigate their usefulness in avoiding the false positive alarm obstacle.

It is important to note that there is likely to be more than one device present in the system.

3.2.2.1 Candidate Resolution 1. Self-Propagating Alarm

The first alarm we consider is a self-propagating alarm, such that once an alarm molecule is in the solution, it will reproduce itself at a fast rate. The rapidly growing number of alarm molecules is easier to detect as time passes because a larger percentage of the solution is made up of the alarm molecules. Each watchdog timer releases a single alarm molecule upon completion, initiating the propagation of the alarm.

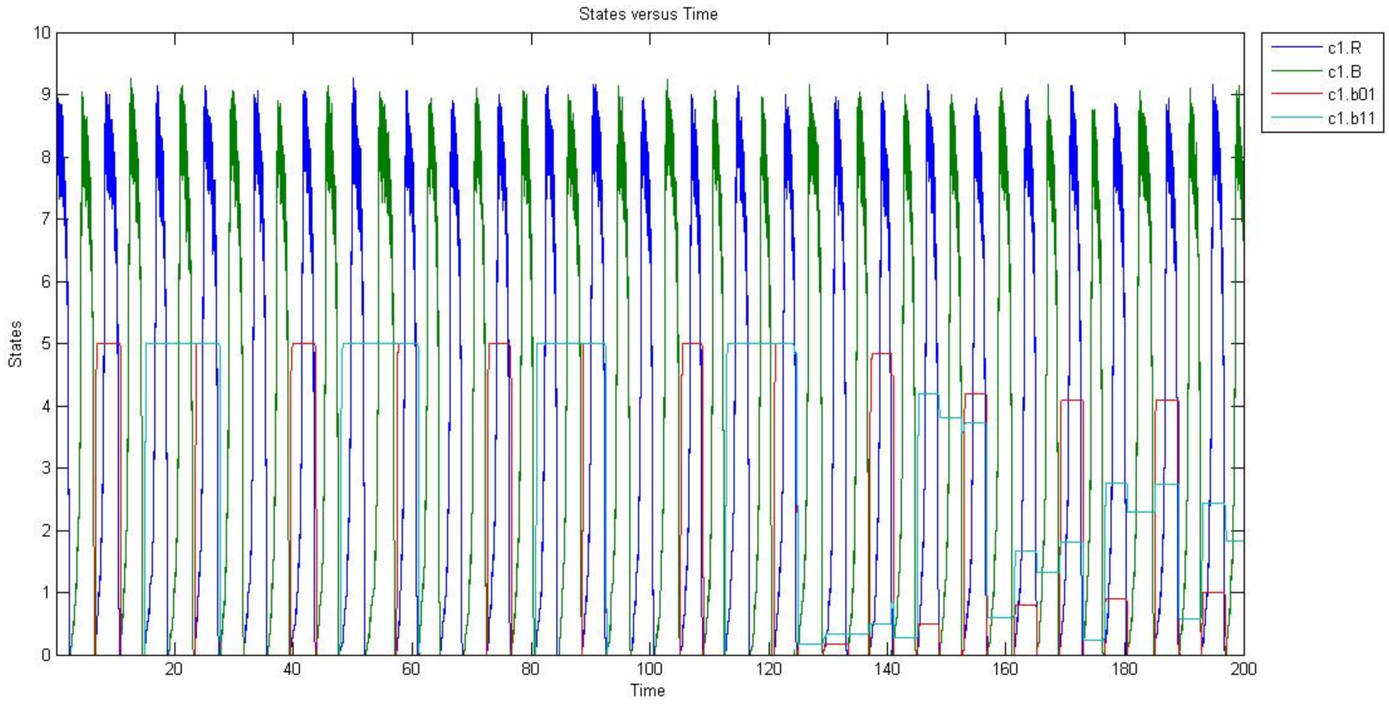

Figure 3.4 Example of a ”Left Behind” Error (Generated by SimBiology)

3.2.2.2 Candidate Resolution 2. Single Alarm per Watchdog Timer

The second alarm alternative is similar to the first in that each watchdog timer releases only a single alarm molecule. The alarm does not propagate; it remains in the solution until it is used in another reaction. The alarm is not considered to be on unless a large number of alarm molecules are present in the solution. In order for the alarm to be on, a large percentage of the watchdog timers in the solution must complete and initiate an alarm.

3.2.2.3 Resolution and Reasoning for False Positive Alarm

The resolution to the false positive alarm obstacle depends on the number of watchdog timers in the solution. If a single watchdog timer is present in the solution, design 3.2.2.1 is a better choice. Design 3.2.2.1 allows for a single watchdog timer to release an alarm that is easily detectable because of the propagation. Design 3.2.2.2 would produce only a single alarm molecule into the solution upon completion of a single watchdog timer. A single molecule will be very difficult to detect due to the large volume of the solution compared to the size of a single molecule.

If there are many watchdog timers present in the solution, design 3.2.2.2 is a better choice. With a large number of watchdog timers, design 3.2.2.1 becomes very fragile. In design 3.2.2.1 if a single watchdog timer alarms at an incorrect time, the alarm will propagate and can be detected. The detection of an incorrect alarm is a false positive alarm which is the obstacle we are trying to avoid. Design 3.2.2.2 is much more fault tolerant with a large number of watchdog timers. Since each watchdog timer releases a single alarm molecule, if a watchdog timer alarms incorrectly, only a single alarm molecule will be input into the solution. Since an alarm will not be considered on until a large number of alarm molecules are present in the solution, the faulty watchdog timer will not initiate an alarm. It takes a large number of watchdog timers releasing alarm molecules to initiate an alarm, so the likelihood of a false positive alarm is smaller than it is in design 3.2.2.1.

3.2.3 Obstacle 3. Non-Observability of the Alarm

A watchdog timer issues an alarm into the solution when it does not receive a heartbeat. This is an obstacle to the leaf subgoal Avoid[Alarm and time<t]. In order for an alarm to be useful, there must be a method of detecting the alarm. Failing to observe an existing alarm is an obstacle we found during the analysis process. We describe two different alarm methods to resolve the observability obstacle.

3.2.3.1 Candidate Resolution 1. Release an alarm molecule to be detected by an external system

The first alternative is to simply have the watchdog timer release an alarm molecule that can be detected by another system. The watchdog timers each release a single alarm molecule, and that is considered issuing an alarm. The watchdog timers work is complete when it issues the alarm.

3.2.3.2 Candidate Resolution 2. Use a Threshold Detecting Device

The second alternative is to create a device capable of detecting when a target threshold of alarm molecules is present in the solution. The threshold device is placed in the same solution as the watchdog timer. The threshold device has a configurable detection threshold which is used to determine how many molecules are needed for the threshold detector to trigger. Upon detecting the target threshold of molecules, the threshold detector releases a new molecule to begin a cascade of reactions or to be detected by another system.

3.2.3.3 Resolution and Reasoning to the Non-Observability of the Alarm

The first alternative solution does not solve the problem, it simply forces a different external system to resolve the obstacle. Therefore, we choose to use the second alternative solution. The threshold device will detect a specified concentration of the alarm molecules in the solution and releases a signal to notify external systems of the detection. A more detailed explanation of the threshold detector is found in Chapter 5.

3.2.4 Obstacle 4. Left behind Alarm

Introducing the threshold detector into the watchdog timer design introduced a new obstacle to the goal Avoid[Alarm if time<t]. Since there are multiple watchdog timers present in the solution, it is possible for a few of the watchdog timers to complete early and release an alarm molecule. Even though it may not be a large enough number to be considered an alarm, the alarm molecules remain present in the solution for the duration of the experiment. When multiple heartbeats pass, these left behind alarm molecules can build up and eventually the number of alarm molecules will be sufficient to issue an alarm, even though the monitored system is properly providing heartbeats.

3.2.4.1 Candidate Resolution 1. Reset early alarm molecules upon a heartbeat input

This alternative solution adds a single reaction to the watchdog timer.

$$Y + H \overset{\text{k}}{\to} L_1 \tag{3.2}$$

This reaction involves the heartbeat molecules H and the alarm molecules Y. When a heartbeat molecule meets an alarm molecule, it resets the alarm back to its initial state as L_1, the first rung of a watchdog timer.

3.2.4.2 Resolution and Reasoning to the Left Behind Alarm

Resetting the alarm molecules that were created in between heartbeats eliminates any potential buildup of the alarm. Each heartbeat essentially resets the watchdog timer to its initial state, all rungs at L_1 and any watchdog timers that completed start over. This also means that an alarm can only be issued if the threshold number of watchdog timers complete during a single delay phase, since they are reset at the beginning of each new delay phase.

3.3 Final Design of the Watchdog Timer

The watchdog timer consists of a set of ladders that create a delay phase and release alarm molecules Y when the delay phase is complete. When a heartbeat, H, is input each ladder

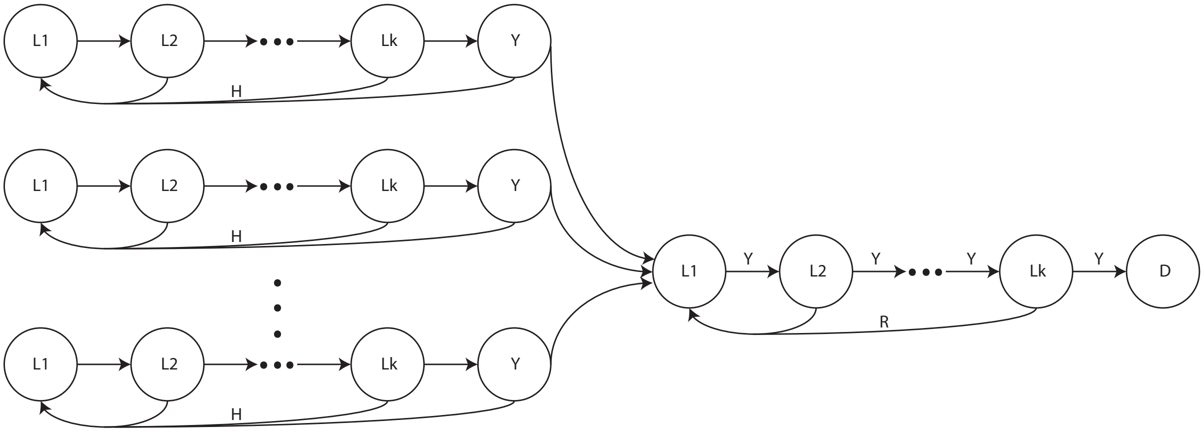

Figure 3.5 State-based representation of the watchdog timer.

is reset to the initial rung, even if it has released an alarm molecule. The alarm molecules then feed into threshold detectors which determine if enough alarm molecules are present to be considered an alarm. If there are enough alarm molecules, a threshold detector will complete and release a molecule, D, indicating that the alarm has been detected. Figure 3.5 provides a representation of the watchdog timer. On the left is the set of ladders that climb up to the top rung and release alarm molecules. On the right is a single threshold detector, only a single detector is shown to simplify the diagram. The threshold detector climbs up when it reacts with Y's and resets to the bottom rung when it reacts with R's.

3.4 Validation

3.4.1 Meeting the Requirements

Our goal is to create a watchdog timer. Such a device must satisfy the top-level goal Achieve[Watchdog Timer] defined in the goal diagram in 3.1. Since the goal diagram only has AND refinements, a device must satisfy all subgoals to satisfy the top level goal. For each goal in the diagram, we define a satisfaction argument of the form:

$$\{SubGoals, DomProp\} \models ParentGoal \tag{3.3}$$

SubGoals is the set of subgoals needed to satisfy the ParentGoal. DomProps is the set of domain properties needed to satisfy the set of goals. We chain satisfaction arguments bottom-up to show that the top level goal is satisfied [17].

We use the following domain properties to determine satisfaction:

1. Molecular reactions take an expected amount of time and have a probability of occurring, both of which can be calculated.

2. Reactants only interact with each other as defined by the set of reactions.

3. A reaction cannot occur in the absence of one or more of its reactants.

We also use the following environmental property:

1. An alarm is mutually exclusive, it is not possible for the alarm to be "on"and "off" concurrently.

Starting with the leaf subgoals, we show that the Unary Watchdog Timer, defined in 3.2, satisfies the parent goal, defined in 3.1, and is therefore a watchdog timer.

We start with the leftmost subgoal, bottom left in Figure 3.1, Achieve[Delay for t time]. Our molecular clock uses a cascade, or ladder, of reactions to create a time delay. By domain property 1, each reaction takes an expected amount of time. We can design our reactions to have an expected time of t until the cascade is complete. The cascade will take t time to complete which satisfies the subgoal.

The molecular alarm agent satisfies the next leaf subgoal Achieve[Initiate alarm]. It uses a single species as an alarm. When the species is present in the solution, the alarm has been issued. By domain property 2, each reactant follows the rules defined by the reaction network. Therefore, the alarm species is only released when the ladder completes and remains active. Once the alarm initiates it remains active since it is not involved in any other reactions. This subgoal also uses environmental property 1, meaning the alarm is considered "off" until it is "on" and once "on" cannot be considered "off". The device can initiate an alarm and the alarm cannot be shut down without external input.

The leaf subgoal Achieve[Detect HB] is satisfied by the molecular clock as follows. A heartbeat is introduced into the system as a single species. The heartbeat is detected when it takes part in a reaction. By domain property 3, no reactions involving the heartbeat species can occur in the absence of the heartbeat. Thus, when a heartbeat is input into the system and is involved in a reaction it is considered detected, and this detection satisfies the subgoal.

The leaf subgoal Avoid[False HB Detection] is satisfied for the same reason that Achieve[Detect HB] is satisfied. We cannot have a false detection by domain property 3. If the heartbeat is not present, it cannot take part in any reactions and therefore the subgoal is satisfied.

The leaf subgoal Achieve[Timer reset] is also satisfied by the molecular clock agent. The device has a set of reactions that reset the ladder. When these reactions occur, the ladder is reset to its initial state which resets the expected time until the ladder completes. The device satisfies the subgoal by having the ability to reset the delay period.

Moving up a layer, the leaf subgoal Avoid[Alarm if time<t] is satisfied by the molecular alarm agent. By domain property 1 we can design the device to have an expected alarm time t. By designing the reactions correctly, we give the device a probability distribution of alarming around time t. By environmental property 1, the alarm cannot be considered both "on" and "off", so the alarm is "off" until it is initiated. We cannot guarantee an alarm at exactly time t because we are working in a molecular domain.

The next goal on this layer is Achieve[Alarm if time>t]. It is satisfied if both of the subgoals, Achieve[Delay for t time] and Achieve[Initiate alarm], are satisfied. We have shown that the design of the molecular clock and the molecular alarm satisfy both of these subgoals. The device will delay for t time by the first subgoal. By the design of the device, the alarm is initiated after the delay is over. Therefore, the device releases an alarm after time t and satisfies the goal.

The goal Achieve[Reset time on HB] is satisfied if its subgoals are met: Achieve[Detect HB], Avoid[False HB Detection], and Achieve[Timer reset]. We have shown that the design of the molecular clock agent satisfies all three subgoals. It will detect a heartbeat and avoid false heartbeat detection errors. In addition, it will reset the delay period when a heartbeat is detected. This means that the goal of resetting the delay period when a heartbeat is input can be satisfied by the molecular clock.

Finally, we show that the molecular watchdog timer satisfies the top-level goal by satisfying each of its subgoals: Avoid[Alarm if time<t], Achieve[Alarm if time>t], and Achieve[Reset time on HB]. We have already shown that these subgoals are each satisfied. As described in 3.1, the top level goal is to initiate an alarm if a heartbeat is not received before time t and to not initiate an alarm if a heartbeat is received. The device satisfies this goal because its subgoals are satisfied. It does not alarm if time is less than t. It alarms if time is greater than t. It resets its time when a heartbeat is received. The satisfaction of its subgoals satisfies the top level goal; therefore, our device is a watchdog timer.

CHAPTER 4. TOOL BASED VERIFICATION

4.1 Background and Assumptions

Ultimately, we seek to accurately model the behavior of a watchdog timer constructed from DNA strands and, using this model, verify the correctness its planned behavior.

We make the following assumptions in our model:

- The volume of the solution remains constant during the experiment.
- A heartbeat is modeled as a single reaction with no reactants. It has a reaction rate derived from the intended input time.
- When a heartbeat reaction fires, it inputs a constant amount of heartbeat molecules.
- Each reaction rate is based on a rate constant for a toehold-mediated DNA strand displacement reaction. Rates constants are taken from [33] and are converted to rate constants for SCRNs by dividing them by Avogadros Constant.
- Strand displacement rates are simplified to model only the toehold binding, instead of the entire displacement reaction. This is a reasonable assumption because the time required for the toehold reaction, depending on the length of the toehold, is much larger than the time needed for displacement to occur [34].
- The volume of the solution is calculated based on the number of ladder molecules and the concentration of the ladder molecules. The formula for the volume is:

$$Volume = \frac{NumberLadders}{(ConcentrationLadders * Avogadro'sConstant)} \tag{4.1}$$

  Where Avogadros Constant, $6.02214x10^{23}mol^{-1}$, is defined as the number of particles in one Mole of a substance.

- Reactions are assumed to be irreversible. Since the watchdog timer is trying to delay for a specified time period, we only want reactions to progress forward. If reactions are stepping backward, then the time to completion changes. The alarm reaction cannot be reversible because once the alarm is present in the system, it must remain present until it is involved in a different reaction.

Limitations in computing power restrict the execution of the models to no more than a few dozen ladders. Beyond that, the state space becomes too large and the verifications do not complete. For example, with five ladders, a 4-rung watchdog timer model contains 252 states; with twenty ladders the number of states climbs to 23,789. With fifty ladders increases the number of states to 632,502 states.

There are an enormous number of molecules in a real biological system. Since we can only simulate a small number of devices, and there will be very many devices in solution, our simulations may not accurately model the interactions of a large number of devices. A model of a constant number of devices smaller than one hundred may not provide an accurate representation of a Watchdog Timer. Further work is needed to verify whether the small scale model accurately represents a real system.

The designs described in section 3.2 were modeled and verified using the PRISM model checker (see Appendix A for the PRISM model). A problem appeared when we tried to model the watchdog timer using rates consistent with DNA systems. A real DNA system has much smaller rate constants than those used in the models. PRISM is unable to handle numbers smaller than $1x10^{-15}$ as rates. Dividing the rate constants by Avogadros constant reduces the rate constants below the minimum threshold, resulting in the rate constants being rounded to zero. To avoid the rounding error, we scaled the rate constants outside of PRISM, performed the model checking, and corrected for the scaling.

External calculations create more opportunities for error and to avoid such possible errors, we used SMART to create a model in addition to the PRISM model (See Appendix B for the SMART model). SMART can handle the small rate constants needed, and therefore preserves automation of rate calculations.

We used the PRISM model to double check the accuracy of the SMART model and compare the two models to verify the correctness of the SMART model. The SMART model automates the rate calculations, removing the possibility for human error. We enter constants for the rates in the PRISM model to avoid the round off error. It is possible to generate the PRISM model with the rates as constants for a given system specification, but it requires the calculations to be performed outside of PRISM, leaving the possibility for error. We used PRISM to gain confidence in the results returned by our SMART model.

4.2 Comparison of the Models

We chose the following values for the parameters in each model:

- Rate Constant: We choose a rate constant of $1.3x10^{6}M^{-1}s^{-1}$, a Mass-Action rate constant for 5 nucleotide length toeholds, for all reactions which is then converted into a stochastic rate constant by dividing it by Avogadros Constant.
- Ladders: Five ladders are present in the solution at initialization.
- Concentration: The ladder molecules have a concentration of $100x10^{-9}M$.
- Heartbeat Input: Heartbeat input time is calculated by taking various fractions of the ladder completion time, which forces the heartbeat to be input at that fraction of the ladder completion time with high probability.
- Heartbeat Concentration: Heartbeat amounts are calculated as functions of the number of ladders, such as ten times the number of ladders in the system. These amounts are added when the heartbeat reactions fire. We use a single heartbeat reaction for simplicity, since the PRISM model is being used to show correctness of our SMART model.
- Alarm Threshold: We set the completion threshold, the number of ladders that need to complete to be considered an alarm, at 80%.

In order to add confidence to the accuracy of the SMART model, we defined the following property for each model:

- PRISM Property:

$$P =?\ [F = T\ Y >= (0.8 * MAX_L)] \tag{4.2}$$

- SMART Property:

$$prob\_at(tk(Y) >= (0.8 * n), T) \tag{4.3}$$

- These properties return the probability that the number of Y molecules in the solution is greater than or equal to 80% of the initial number of ladders at a specified time T, where Y is the number of alarm molecules.

The expected time to complete 80% of the ladders is generated in SMART using the property:

$$prob\_acc(!(tk(Y) > (0.8 * n)), 0, infinity) \tag{4.4}$$

- Returns the expected time between 0 and infinity that the number of Y molecules is less than or equal to 80% of the number of ladders. Where Y is the number of alarm molecules and n is the number of ladders at initialization.

We perform verification of these properties in both models using $0 \leq T \leq 100,000$ and the parameter values defined above. Using PRISM required scaling of the expected time and the rate constants to perform model checking by a factor of 1000. Figure 4.1 shows the PRISM results and Figure 4.2 shows the SMART results.

By comparing Figures 4.1 and 4.2, we can see that both returned the same probability of initiating an alarm. It is also noteworthy that the probability of achieving an alarm is much smaller given the presence of a heartbeat.

4.3 Correctness of the SMART Model

With confidence in the correctness of the SMART model, we use it to test different variations of the Watchdog Timer.

For our model to be correct, we need to check for the following properties:

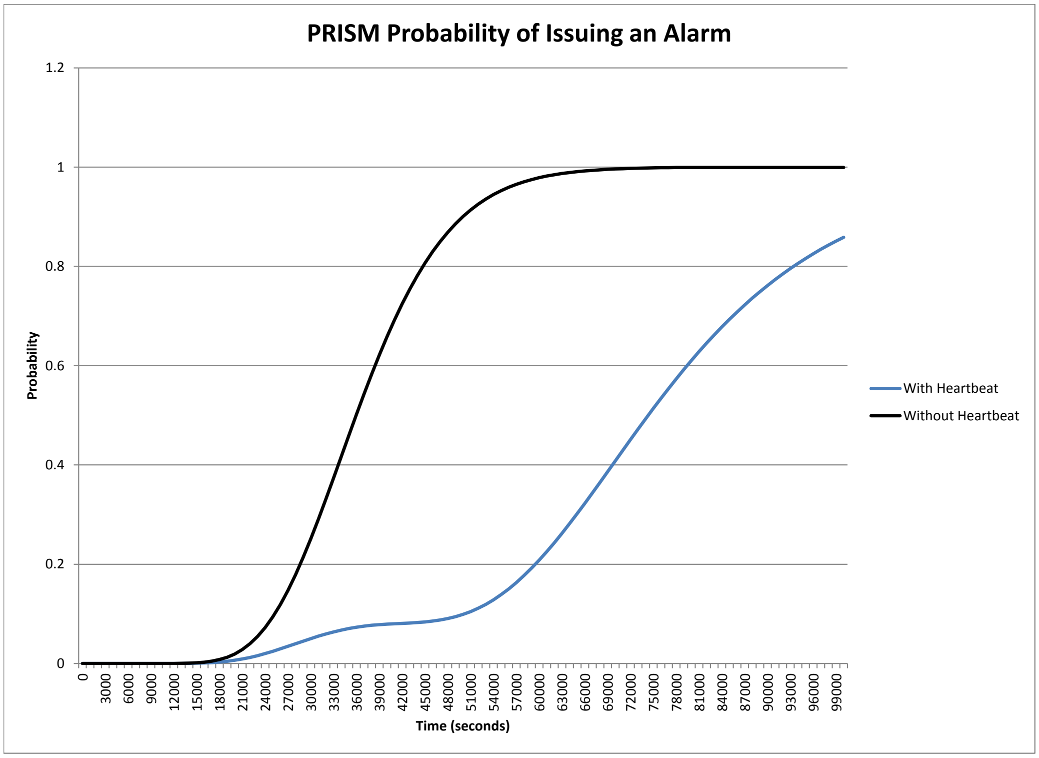

Figure 4.1 Probability of issuing an alarm over time with a single heartbeat input at half of the expected time (approximately 18,500 seconds) generated by PRISM.

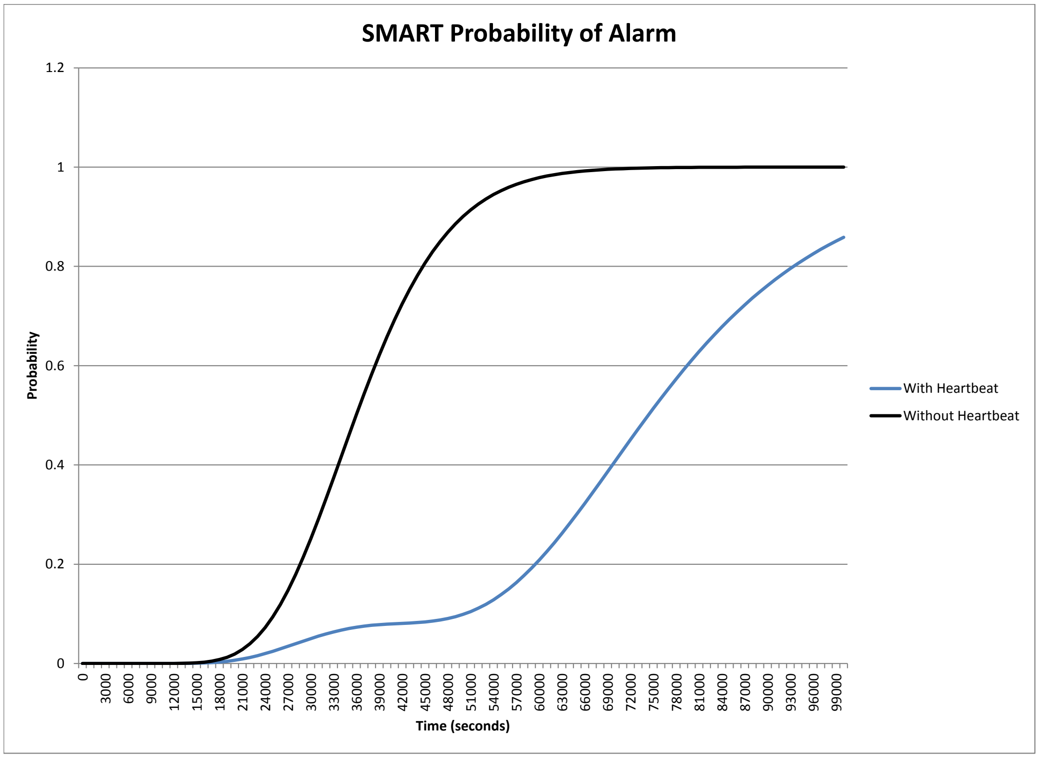

Figure 4.2 Probability of issuing an alarm over time with a single heartbeat input at half of the expected time (approximately 18,500 seconds) generated by SMART.

- Obstacles to avoid for a given p and t:
  1. $P \geq p : Alarm \wedge (Time < t)$
     - The probability is greater than or equal to p that the Alarm is active and Time is less than t.
  2. $P \geq p : \neg Alarm \wedge (Time \geq t)$
     - The probability is greater than or equal to p that the Alarm is not active and the time is greater than t.
- Requirements to meet for a given p and t:
  1. $P \leq p : Alarm \wedge (Time < t)$
     - The probability is less than or equal to p that the alarm is active and time is less than t.
  2. $P \geq p : Alarm \wedge (Time \geq t)$
     - The probability is greater than or equal to p that the alarm is active and the time is greater than or equal to t.

Simply put, we want to show that the probability of issuing an alarm is high in the absence of heartbeat inputs and low in the presence of heartbeat inputs.

That is, the probability of initiating an alarm should remain low as long as heartbeats are input. If the probability of issuing an alarm is not low, even in the presence of heartbeats, then the watchdog timer does not work. Model checking was performed with the SMART model. We define the expected time, t, as the time it takes for 80% of the ladders to complete in the absence of heartbeats. We modeled the probability of achieving an alarm between 0 and 100 seconds with the following heartbeat variations with a 4-rung watchdog timer:

- No Heartbeat inputs. A control case that shows how the ladders react without heartbeats.
- One Heartbeat input at 1/2 the expected time with heartbeat inputs ranging from 10*n to 40*n molecules, where n is the number of ladders. We input a large number of heartbeat molecules to ensure that all of the timers reset. Results are shown in Figure 4.3.

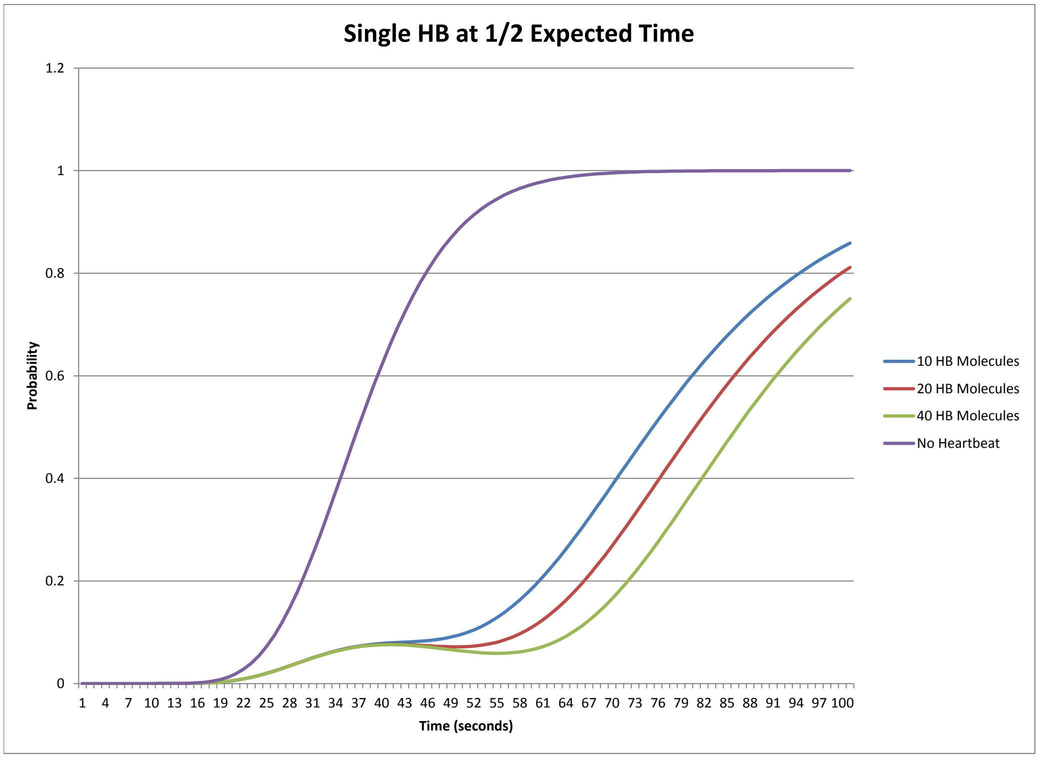

Figure 4.3 Probability of issuing an alarm over time for different heartbeat amounts and a single heartbeat at 1/2 expected time.

- One Heartbeat input at 3/4 the expected time with heartbeat inputs ranging from 10*n to 40*n molecules, where n is the number of ladders. Results are shown in Figure 4.4.
- Two Heartbeats, the first at 1/4 the expected time and the second at 3/4 the expected time with heartbeat inputs ranging from 10*n to 40*n molecules, where n is the number of ladders. Results are shown in Figure 4.5.

Varying the number of heartbeats, the heartbeat input time, and the number of molecules in a heartbeat event all change the probability of achieving an alarm by the expected time. A single heartbeat at half the expected time shows better results for earlier times than a single heartbeat at three quarters the expected time. This is not surprising because the heartbeat at three quarters time allows a longer period of time for ladders to complete, increasing its

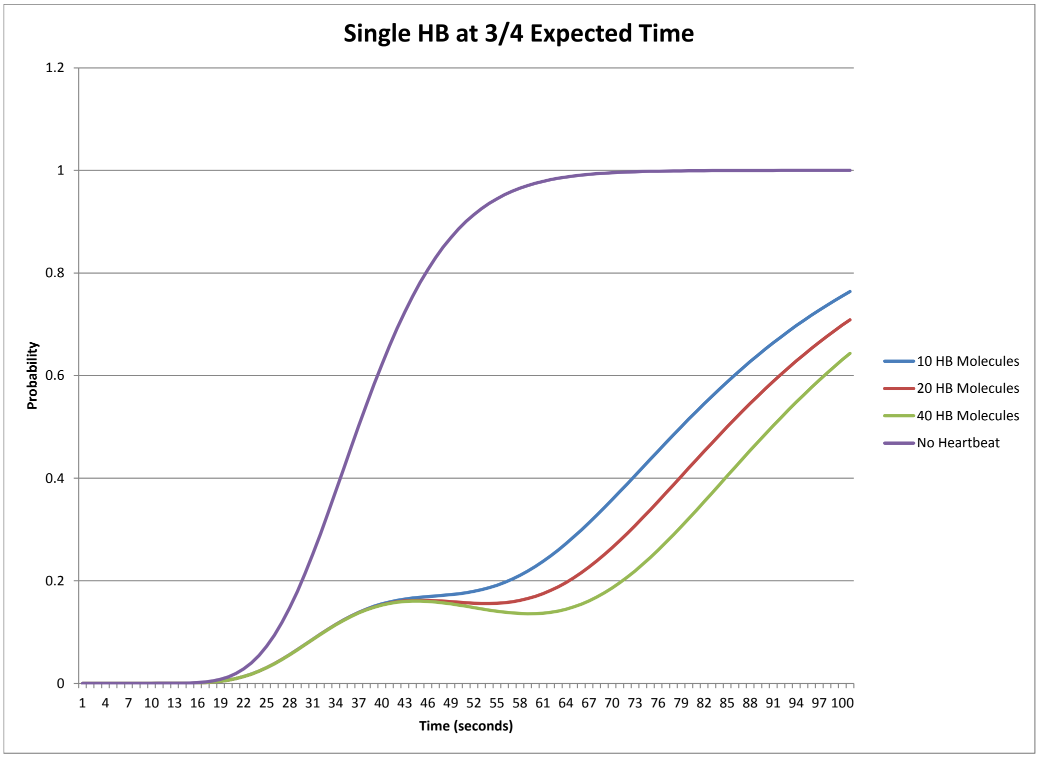

Figure 4.4 Probability of issuing an alarm over time for different heartbeat amounts and a single heartbeat at 3/4 expected time.

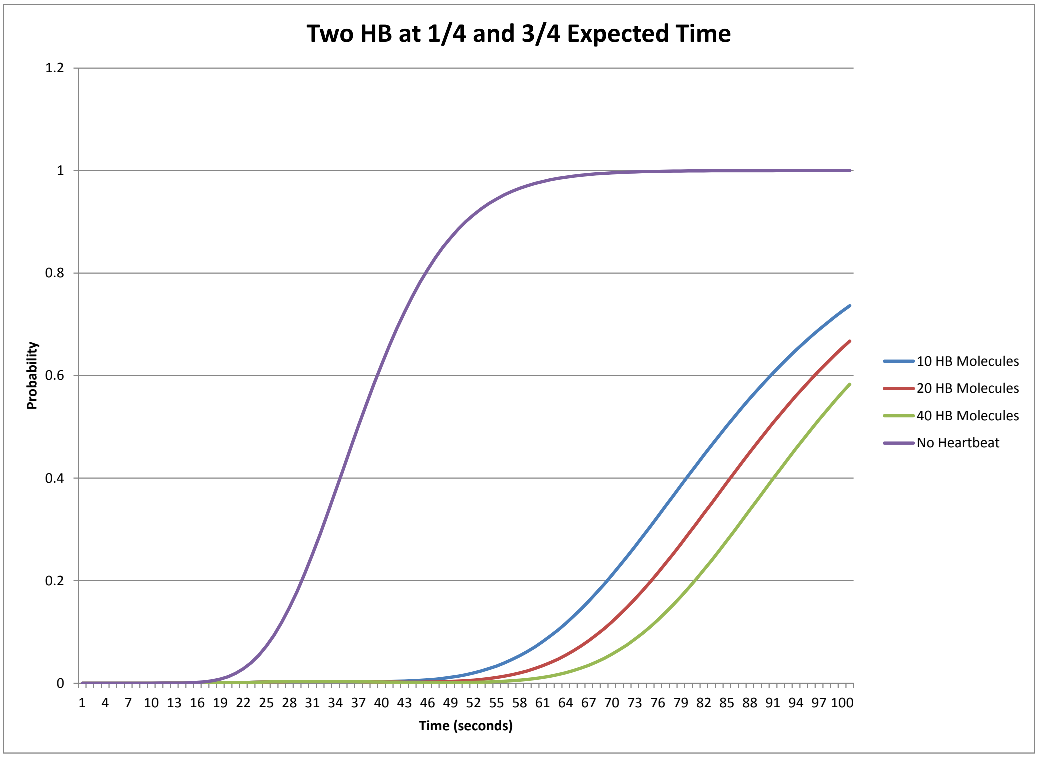

Figure 4.5 Probability of issuing an alarm over time for different heartbeat amounts and two heartbeats at 1/4 and 3/4 expected time.

probability of issuing an alarm earlier. The heartbeat at three quarters time has a lower probability of issuing the alarm for later times, due to the heartbeat being input later and resetting any watchdog timers that may have completed. All cases of two heartbeats show a lower probability of achieving an alarm when it shouldn't occur than the single heartbeat cases. These graphs also show that changing the input time and the number of heartbeats has a much larger influence on the probability than changing the number of molecules input with each heartbeat.

Verification using SMART reveals that adding heartbeats into the system correctly resets the watchdog timer because the probability of initiating an alarm dramatically decreases in the presence of heartbeats. The results shown in Figures 4.3, 4.4, and 4.5 suggest that with the correct heartbeat and ladder configuration, we can maintain a low probability of initiating an alarm. In the absence of those heartbeats, however, there is a high probability of initiating an alarm.

We also used SMART to investigate how increasing and decreasing the number of rungs in the ladder of the watchdog timer changes the probability of success. Using the 4-rung watchdog timer as a starting point, we modeled a 2, 6, and 8 rung watchdog timer. We performed model checking with 5 watchdog timers in the system, a heartbeat input of 50 molecules, and a single heartbeat input at half the expected time. We checked the probability of issuing an alarm between 0 and 150 seconds.

Figures 4.6, 4.7, 4.8, and 4.9 display the results of the model checking. We observe that having a larger number of rungs in the watchdog timer increases the overall time until an alarm is issued, with or without a heartbeat. The heartbeat is seen to delay the alarm in each watchdog timer, as designed. The number of rungs in the watchdog timer also determines how quickly the probability of issuing an alarm increases in the absence of a heartbeat. Comparing the 2-rung and the 4-rung watchdog timers, we see that the probability of issuing an alarm increases at a slower rate when there are more rungs in the watchdog timer. The same can be said for the 6 and 8-rung watchdog timers when compared with the 2 and 4-rung watchdog timers.

The ability to select the number and size of the heartbeats and the number of rungs provides

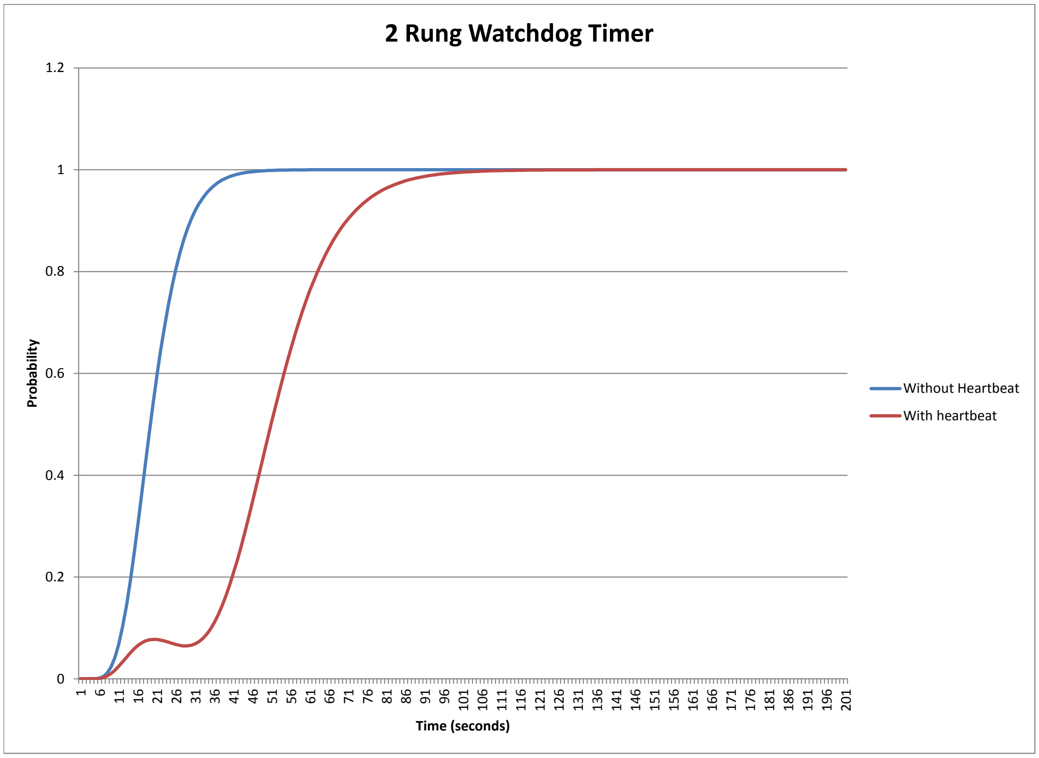

Figure 4.6 Probability of issuing an alarm over time for a 2-Rung watchdog timer.

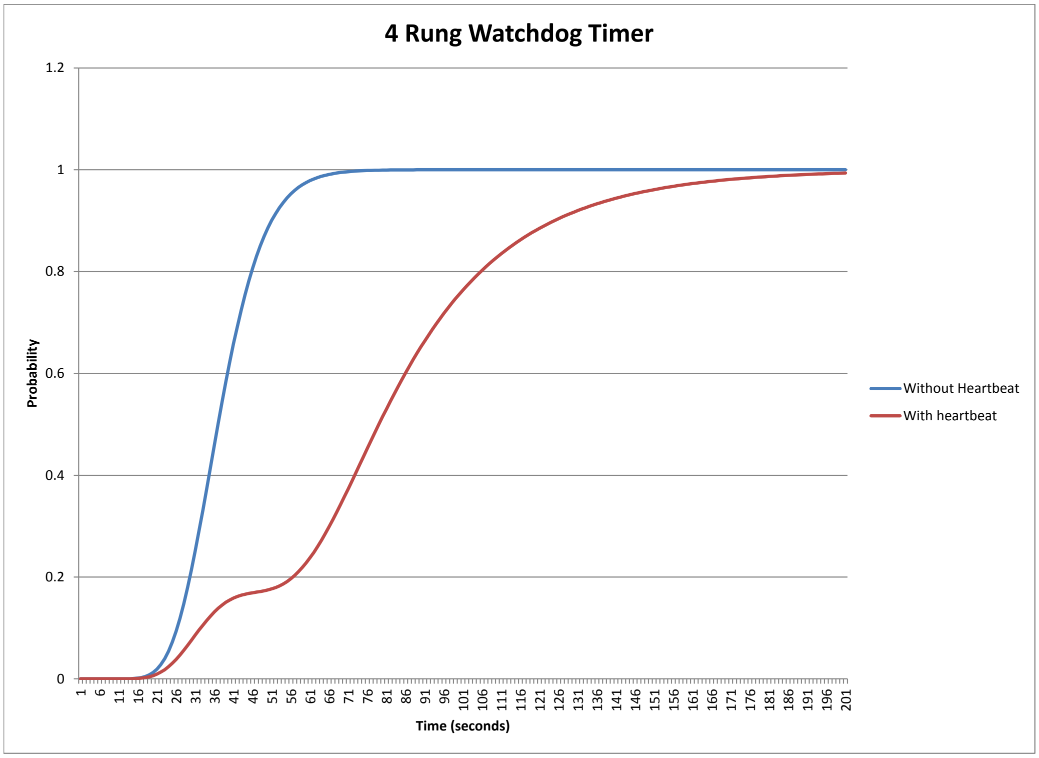

Figure 4.7 Probability of issuing an alarm over time for a 4-Rung watchdog timer.

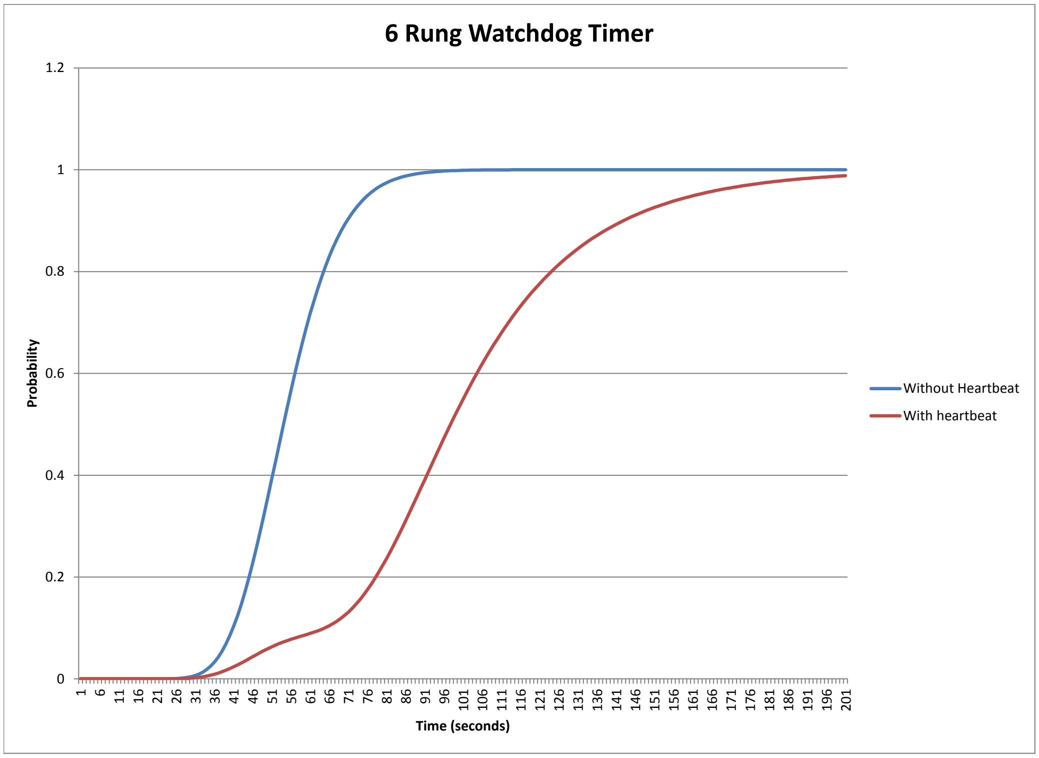

Figure 4.8 Probability of issuing an alarm over time for a 6-Rung watchdog timer.

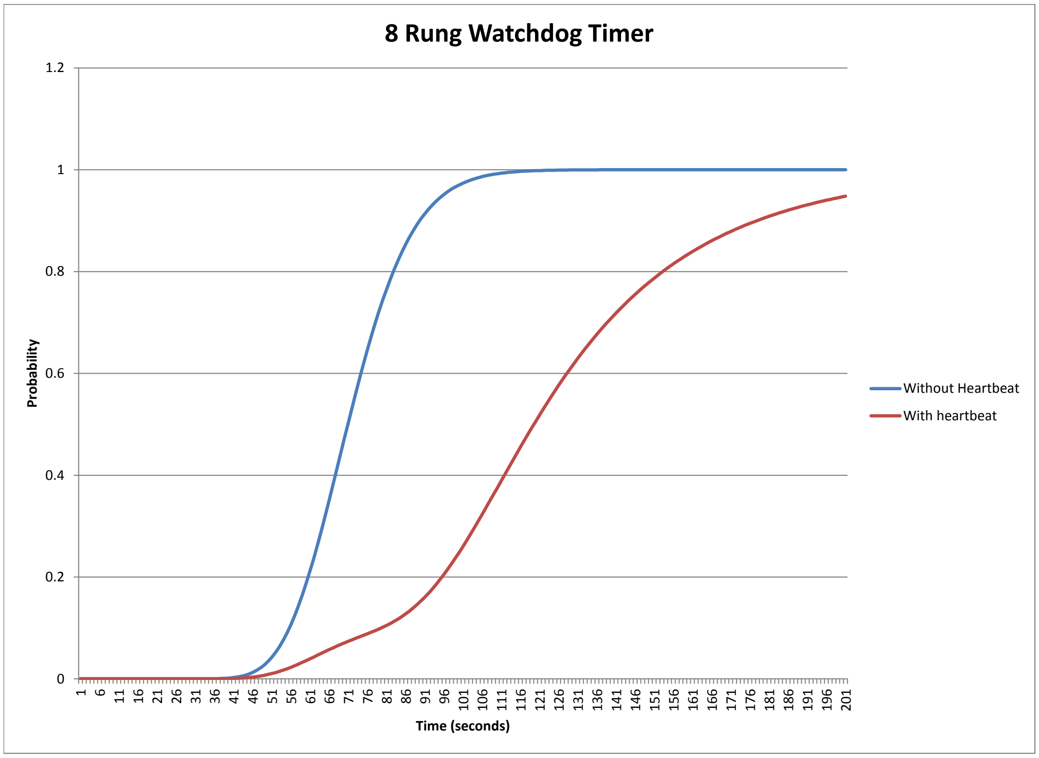

Figure 4.9 Probability of issuing an alarm over time for a 8-Rung watchdog timer.

a user with some control over the accuracy of a watchdog timer. The value of each parameter can be changed independently to increase or decrease the accuracy of a watchdog timer or to create a watchdog timer that meets the specific needs of a user. For example, if a user wants to monitor a slow system that will release heartbeat pulses with large time intervals in between them, they can increase the number of rungs in the watchdog timer to increase the time it takes to issue an alarm, allowing enough time to release a heartbeat pulse to reset the timer.

Future work will investigate to what extent there exists a watchdog timer design to meet various client needs.

CHAPTER 5. THRESHOLD DETECTION (MOLECULAR ALARM)

5.1 Motivation

A watchdog timer releases an alarm when a system does provide a heartbeat within a specified length of time. However, we need a means to detect the alarm instead of just releasing it. In this section we describe the molecular device, present in the solution with the watchdog timer, that is required to detect the alarm. The molecular alarm agent must determine that a target threshold number of watchdog timers have issued an alarm. We design a threshold detection device to detect when a specified number or concentration of a species is present in the solution.

5.2 Design of the Threshold Detector

We define a CRN to model the threshold detection device. We use a modified version of the ladder to create the detector. It consists of four different types of molecules:

- L_1, L_2, $\ldots$, L_n: represent the different rungs of the detector
- Y: the target molecule to be detected by the device.
- R: the threshold to be met, acts as the ladder reset species.
- D: the molecule released to represent detection.

Figure 5.1 lists the reactions in the threshold detector. We define a set of detection reactions which “climb the detection rungs” in the presence of the target molecule. We also define a set of restricting reactions to constantly reset the ladder. At initialization, we define a number of ladder molecules and a number of R molecules. Y’s are added into the system as each watchdog timer reaches the top of the ladder (see Figure 3.3).

$$\begin{array}{cc}
\underline{\textbf{Detection Reactions}} & \underline{\textbf{Restricting Reactions}} \\
L_n + Y \xrightarrow{\text{k}} D + Y & L_n + R \xrightarrow{\text{k}} L_1 + R \\
\vdots & \vdots \\
L_3 + Y \xrightarrow{\text{k}} L_4 + Y & L_3 + R \xrightarrow{\text{k}} L_1 + R \\
L_2 + Y \xrightarrow{\text{k}} L_3 + Y & L_2 + R \xrightarrow{\text{k}} L_1 + R \\
L_1 + Y \xrightarrow{\text{k}} L_2 + Y & L_1 + R \xrightarrow{\text{k}} L_1 + R
\end{array} \tag{5.1}$$

Figure 5.1 Molecular Threshold Detector Reactions

5.3 Analysis of the Threshold Detector

Essentially, the detector has two molecules working in competition. The Y molecules react to climb the ladder, while the R molecules react to reset ladders. To reach the top of a ladder, the number of Y must approach the number of R because the structure of the detector creates a competition. The detector releases a detection signal only when n, where n is the number of rungs in the ladder, detection reactions occur without a restricting reaction occurring. When multiple threshold detectors are present in the solution, it still requires n reactions to occur with a single ladder molecule to release a detection signal.

The detector has a set of parameters which can be manipulated to meet different user needs. Increasing or decreasing the number of rungs in the ladder increases and decreases the accuracy of the detector, respectively. This is because each additional rung adds another chance for a restricting reaction to reset the ladder, reducing the probability that a low number of Y's releases a detection signal. Increasing the number of R's means that the number of Y's required for the detection signal to be released also increases. The higher number of the two species, Y and R, has more control over the state of the ladder.

Changing different rate constants within the threshold detector has a number of effects on its behavior. Defining a higher rate constant for the detection reactions than for the restricting reactions makes it easier for a small number of Y to be detected because the "climbing" reactions occur at a faster rate than the restricting reactions. Similarly, if the rate constant for the restricting reactions is higher, it is harder for a small number of Y to be detected.

Manipulating the rate constants will also change the expected time until a detection signal is released. Faster rate constants decrease the expected time and slower rate constants increase the expected time.

5.4 Modeling the Threshold Detector

We model the threshold detector as a Probabilistic Petri-net using SMART (See Appendix C for the SMART model). Probabilistic Petri-nets consist of places, containers for tokens, and transitions, connections between places that consume and produce tokens. We define a place for each molecule Y, R, D and one for each L_i, and set the initial number of tokens based on the initial values of each variable. We define a transition for each reaction. The model takes a value for R, the number of ladders, and a value of Y, the number of alarm molecules. We verified the following property:

$$\mathit{probDatT} : \mathit{prob\_at}(tk(D)! = 0, t) \tag{5.2}$$

probDatT returns the probability that the number of tokens in place D is non-zero at time t. Figures 5.2, 5.3, 5.4, and 5.5 show the results for 5 detectors with 2, 4, 6, and 8 rungs with 20 R molecules.

Figures 5.2, 5.3, 5.4, and 5.5 have 4 plots which represent a value of Y equal to 5, 10, 15 and 20 (i.e., 25%, 50%, 75%, and 100% of the value of R). Looking at the different graphs, it is apparent that the 2 rung detector is too fragile. A 2 rung ladder has a high probability of incorrectly releasing a “threshold detected” molecule when only a small concentration of the target molecule is present. The 4 rung detector is a little better, in that a larger number of the target molecule is required to complete the ladder. The 6 and 8 rung detectors both have a low probability of releasing the “threshold detected” molecule with small numbers of the target molecule present, and only have a high probability of completing when the number of target molecules is about 75% of R or higher.

Figure 5.6 shows the probability of detecting a threshold with between 1 and 100 target molecules and an R of 100. All probabilities are measured at 150 seconds. This graph shows that with a small number of threshold molecules, detection is very unlikely. To decrease the

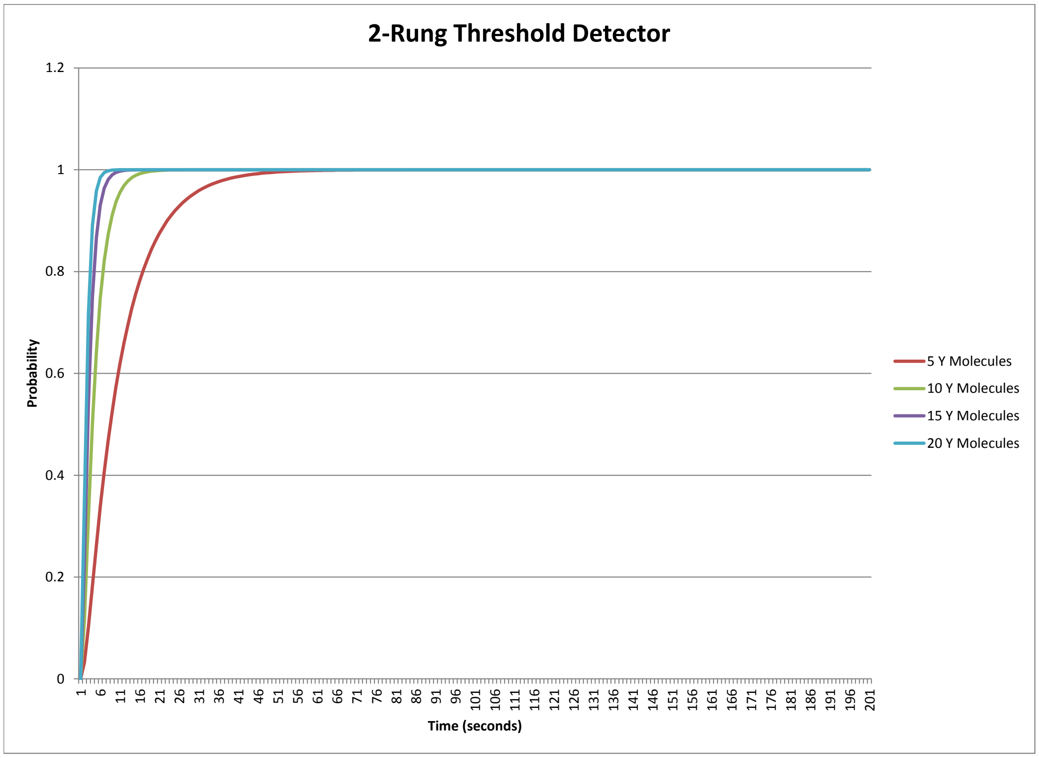

Figure 5.2 Probability of detecting an alarm over time for a 2-Rung threshold detector.

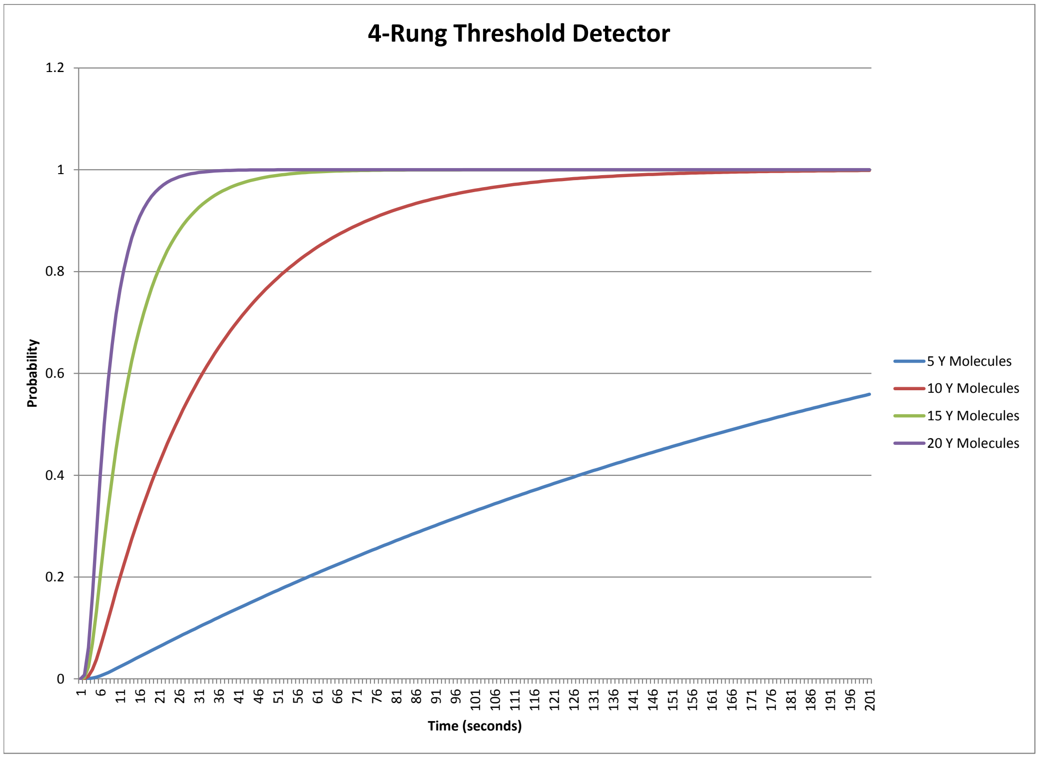

Figure 5.3 Probability of detecting an alarm over time for a 4-Rung threshold detector.

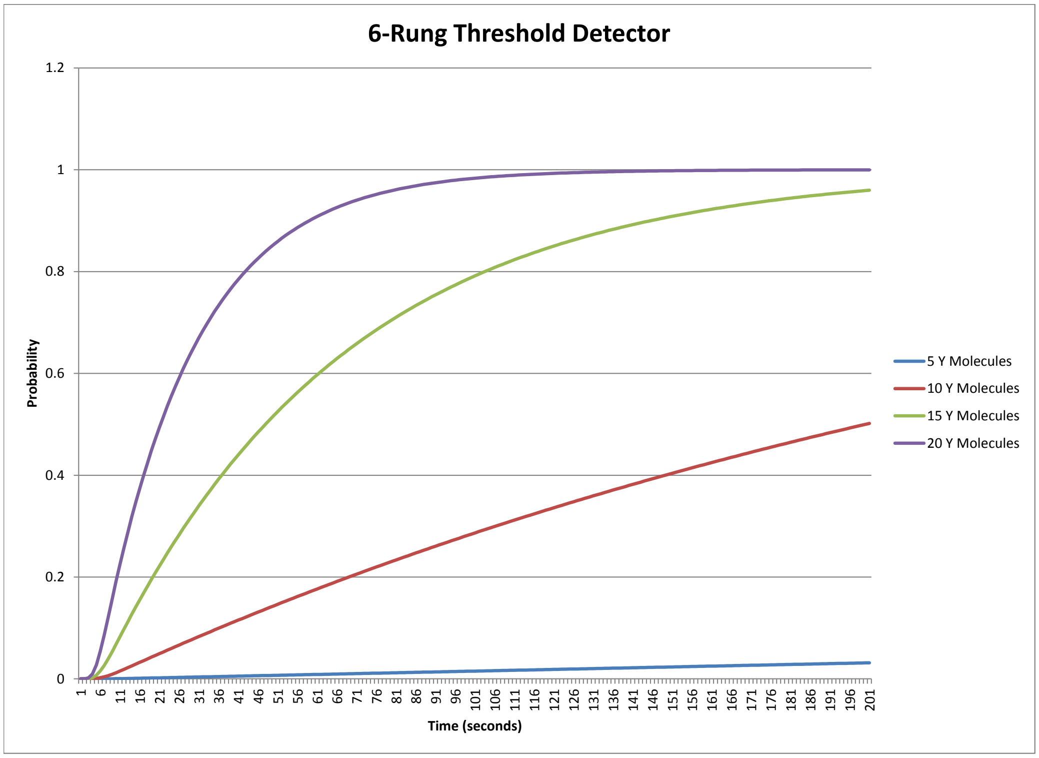

Figure 5.4 Probability of detecting an alarm over time for a 6-Rung threshold detector.

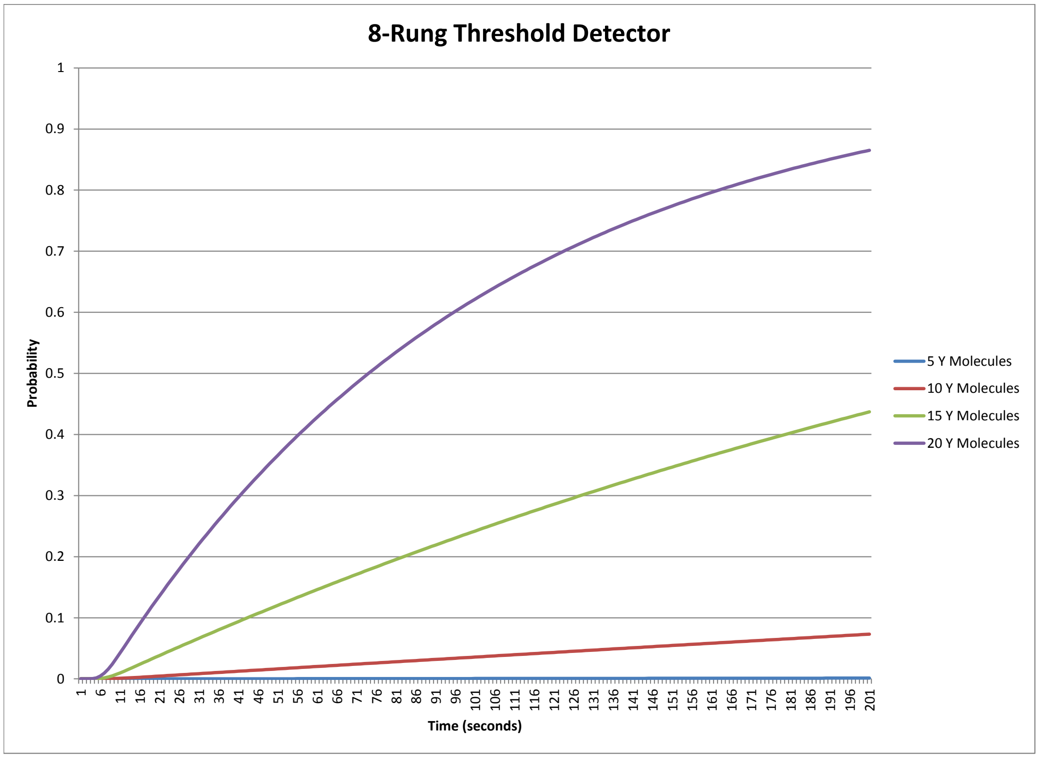

Figure 5.5 Probability of detecting an alarm over time for a 8-Rung threshold detector.

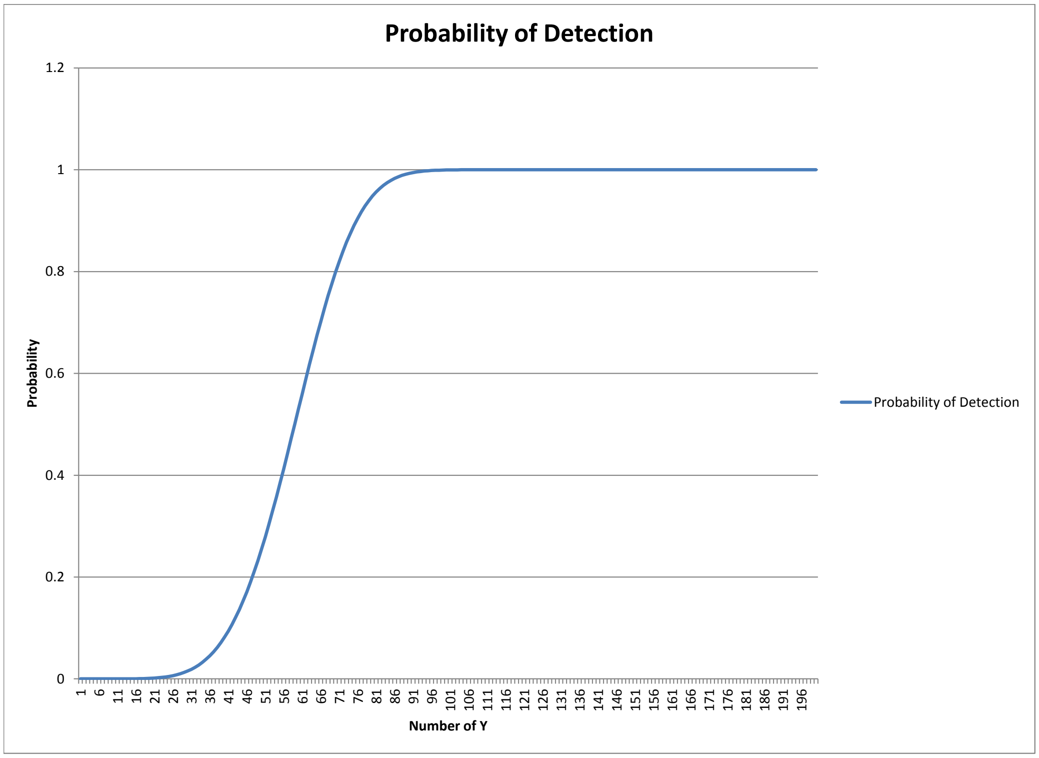

Figure 5.6 Probability of detecting an alarm with different alarm amounts.

probability of detecting a threshold under the target amount, R can be set at a multiple of the target threshold; e.g., set R to double the target threshold. It will take more time to achieve detection, but will lower the probability of false detection.

We combined the models for a 4-rung watchdog timer and an 8-rung threshold detector, to show how the combined system behaves. We used the following values for the model parameters:

- Rate constant: We used a rate constant of $1.3x10^6$ for all of the watchdog timer reactions and set the rate constant for the threshold detector reactions at ten times as high. Since the threshold detector has 8 rungs, it will take longer to complete than the watchdog timer if they have the same rate constant. To better show their interaction, we sped up the threshold detector.

- Number of Devices: We used 5 watchdog timers with 1 threshold detector and set the threshold amount at 4, 80% of the number of watchdog timers.

- Number of Heartbeats: We used 3 heartbeats to delay the alarm for a longer period of time. The heartbeats were input at half of the expected time, at the expected time, and at one and a half times the expected time.

- Size of Heartbeats: We used 10 times the number of watchdog timers, so 50 heartbeat molecules per heartbeat.

Figure 5.7 shows the probability of detecting an alarm in comparison to the probability of issuing an alarm. While the probability of issuing an alarm remains low, so does the probability of detecting the alarm. The probability of detecting the alarm does not become high until the probability of issuing an alarm exceeds 40%.

Figure 5.8 shows the probability of detecting an alarm in comparison to the expected number of alarm molecules, Y, present in the solution. The probability of detecting an alarm remains extremely low while the number of Y molecules is low. The probability of detection only becomes high once the number of alarm molecules exceeds the threshold number of molecules, 4.

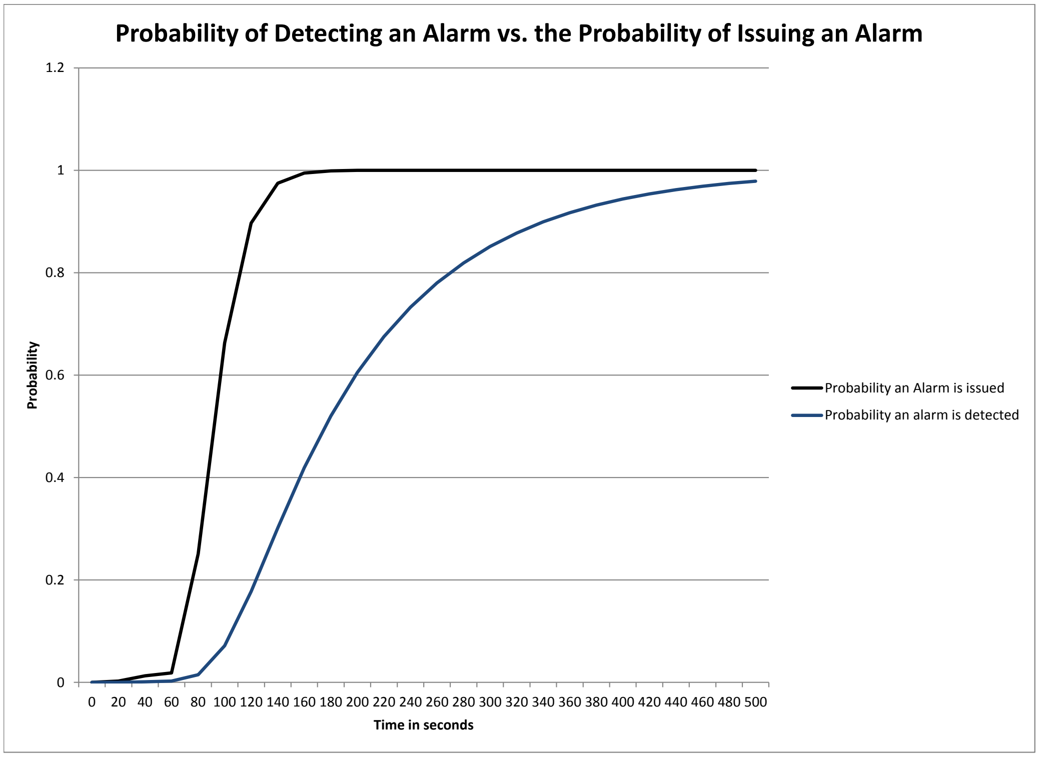

Figure 5.7 Probability of detecting an alarm vs the probability of issuing an alarm.

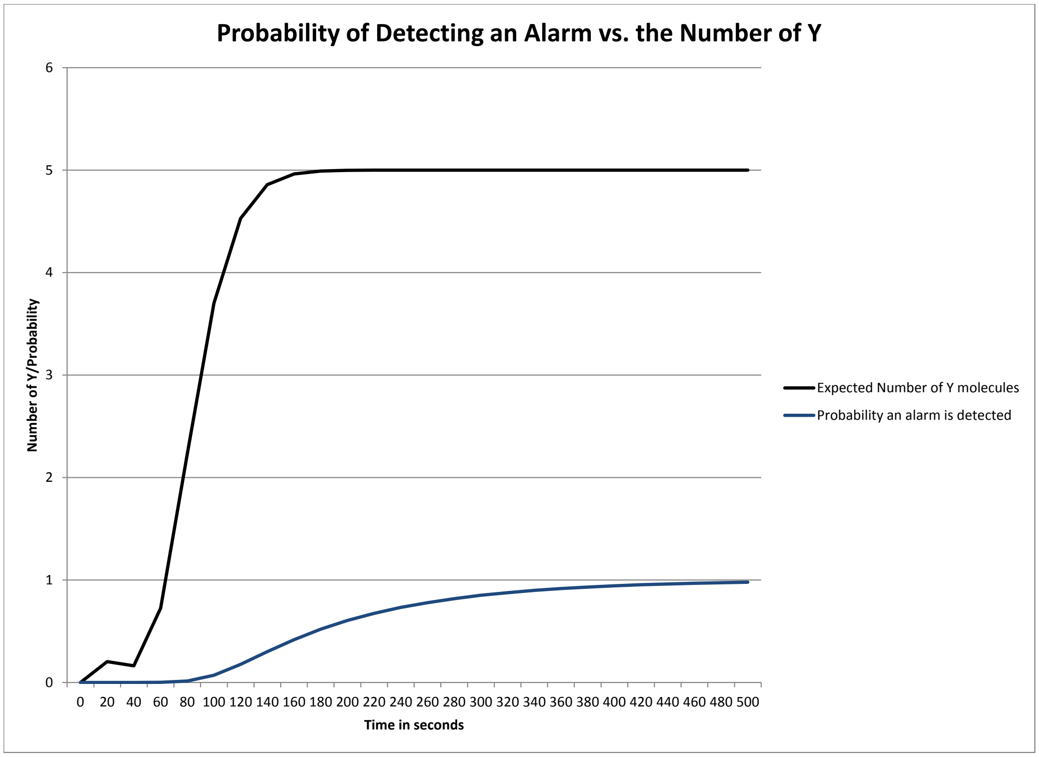

Figure 5.8 Probability of detecting an alarm vs the expected number of alarm molecules (Y).

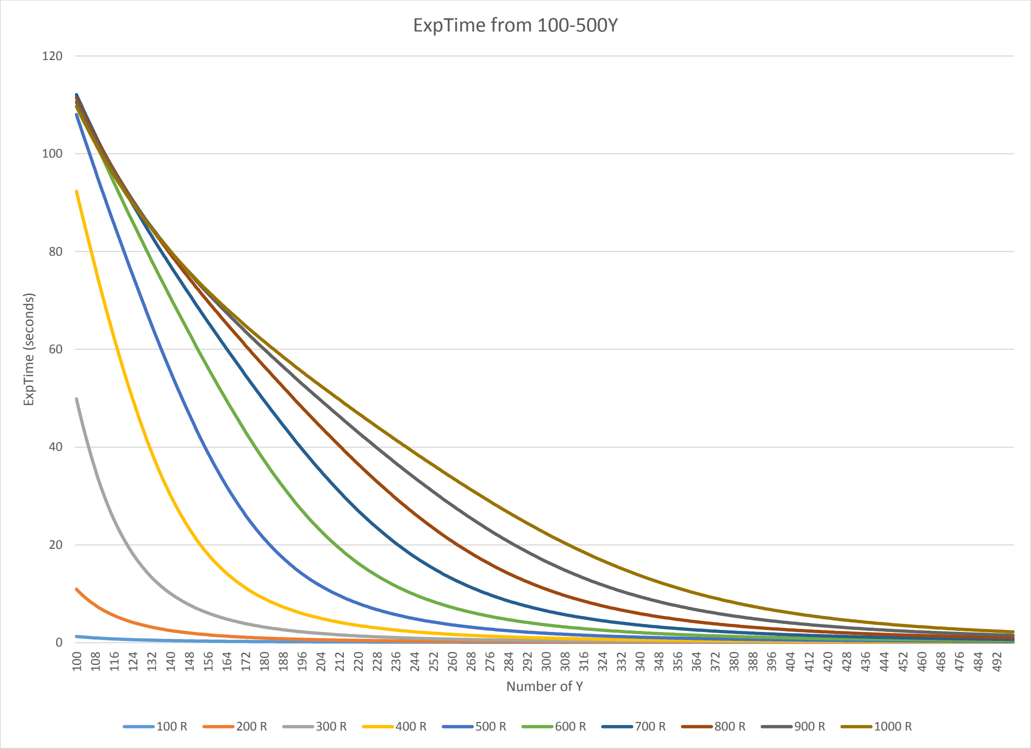

Figure 5.9 Expected Time of detecting an alarm with 100 to 500 Y molecules and varying numbers of R.

The expected time to detect an alarm needs to be long when the number of Y is smaller than the number of R in order to lower the probability of false detections. Figures 5.9, 5.10, 5.11, and 5.12 show the expected time until an alarm is detected by the threshold detector for varying numbers of R molecules and Y molecules. Observe that for each number of R, the expected time is large as the number of Y is very small.

Another important measure is the probability of detecting an alarm within a single heartbeat. In order for the threshold detector to release a false positive detection signal, the threshold ladder needs to reach the top rung and release a signal before any Y's present in the system are reset by the next heartbeat. Figure 5.13 shows the probability of detecting an alarm with an 8-rung detector and varying rate constants. All of the rate constants for the watchdog timer

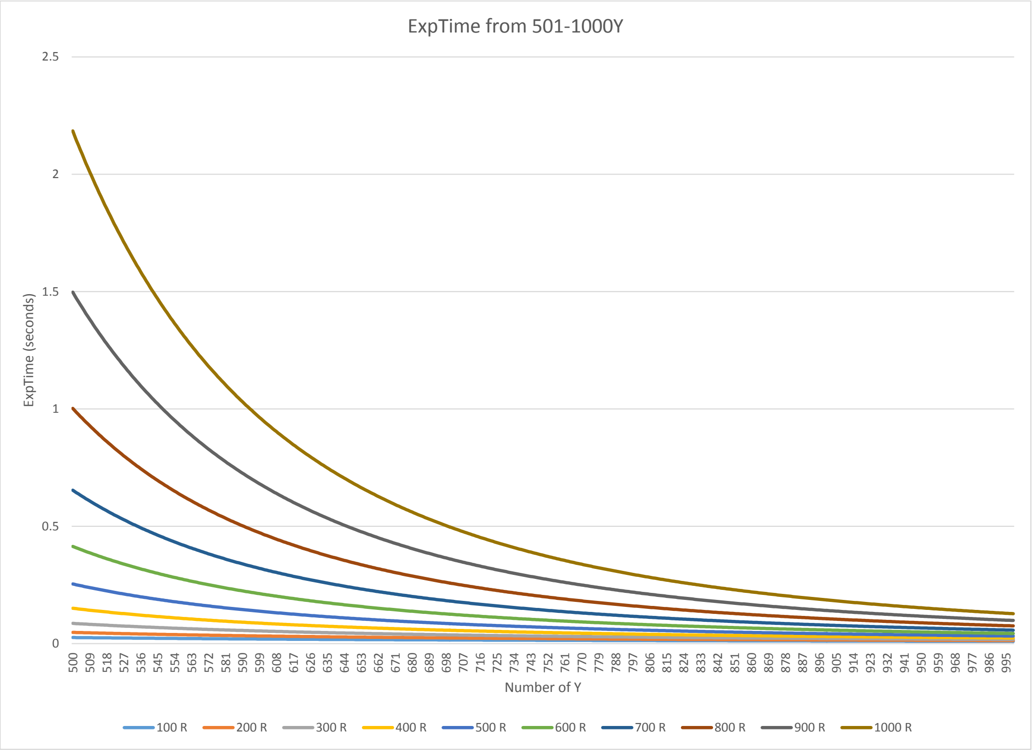

Figure 5.10 Expected Time of detecting an alarm with 501 to 1000 Y molecules and varying numbers of R.

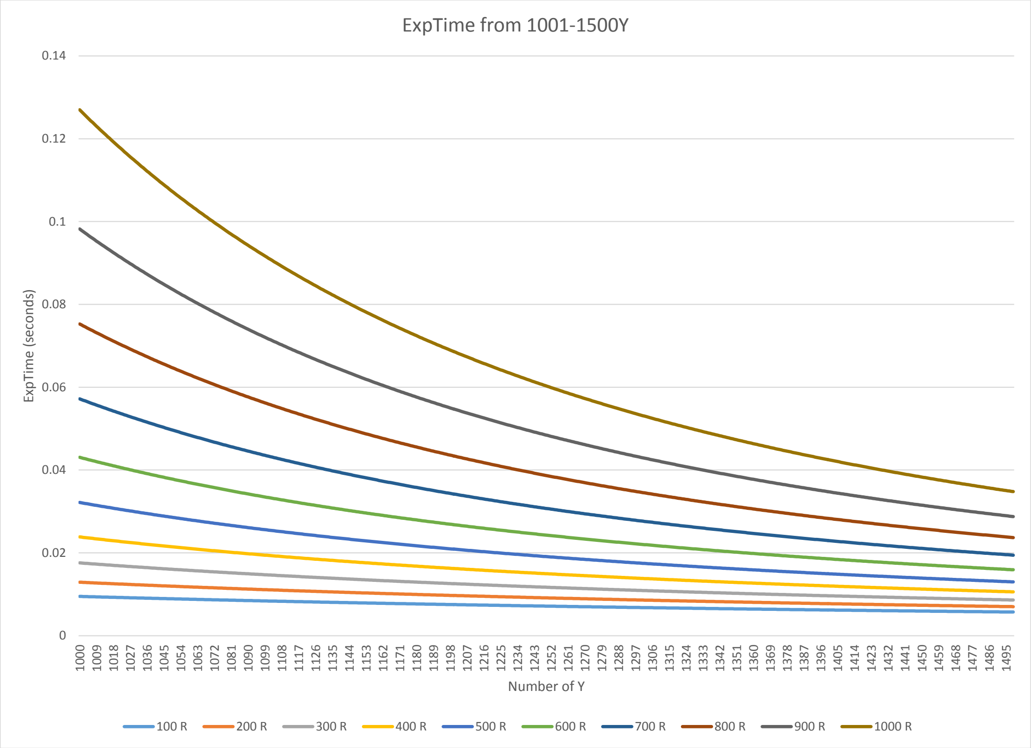

Figure 5.11 Expected Time of detecting an alarm with 1001 to 1500 Y molecules and varying numbers of R.

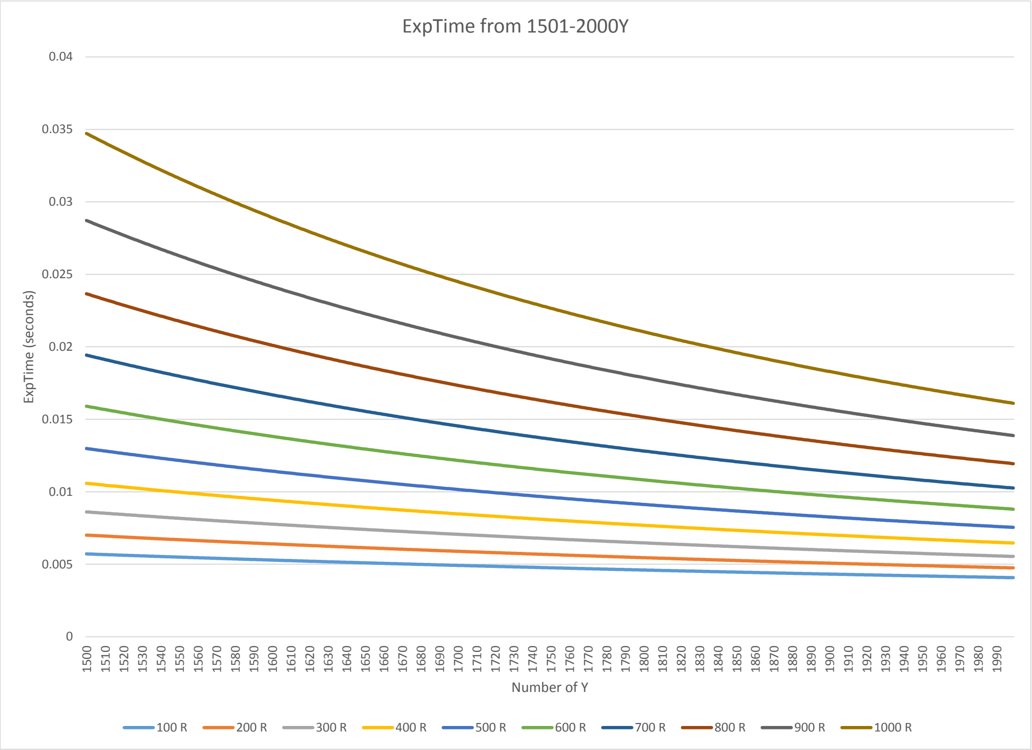

Figure 5.12 Expected Time of detecting an alarm with 1501 to 2000 Y molecules and varying numbers of R.

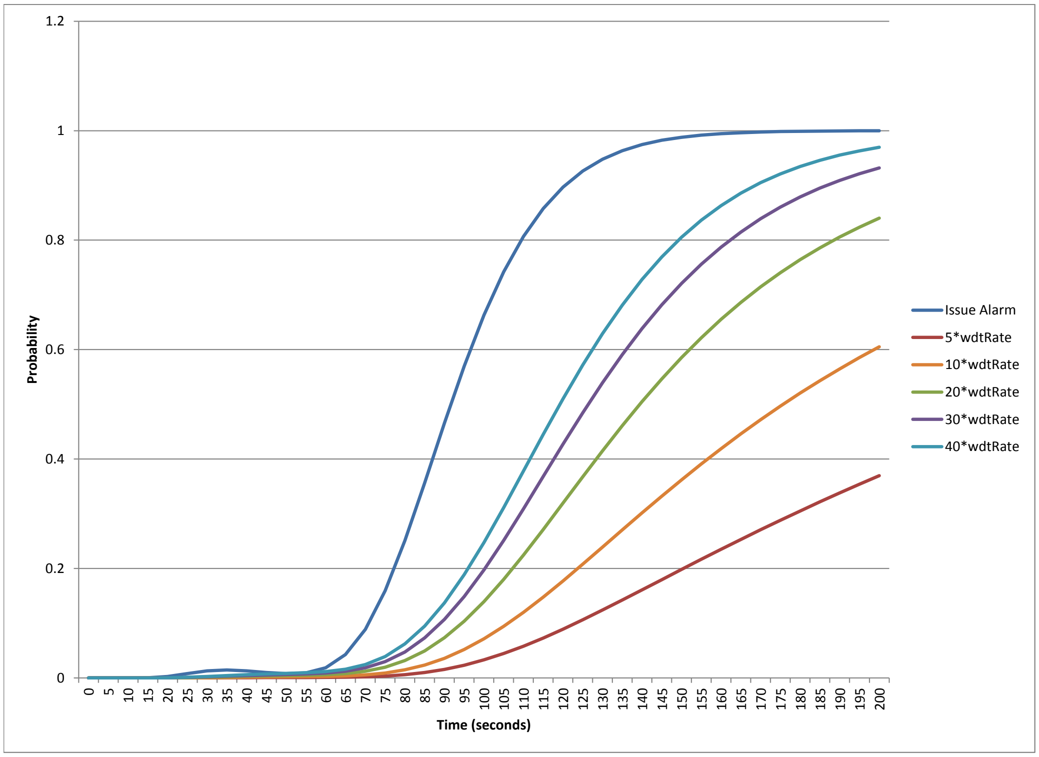

Figure 5.13 Probability of detecting an alarm with different threshold rate constants.

are set at $1.3x10^6$ as a mass action rate constant. The threshold detector rates are all equal and are set at a multiple of the watchdog timer rate constant.

Observe that increasing the rate constant of the threshold detector increases the probability of detection at earlier times. This is because the expected time for each reaction to happen is smaller as the rate constant increases.

Figure 5.14 shows a more focused version of Figure 5.13. Though increasing the rate constants decreases the amount of time it takes to detect an alarm, it also increases the probability of falsely detecting an alarm. Since the reactions occur faster, more reactions can occur in a small amount of time increasing the probability of falsely detecting an alarm. There is a trade off between accuracy and speed.

The model of the threshold detector exhibits the desired behavior. We can set the param-

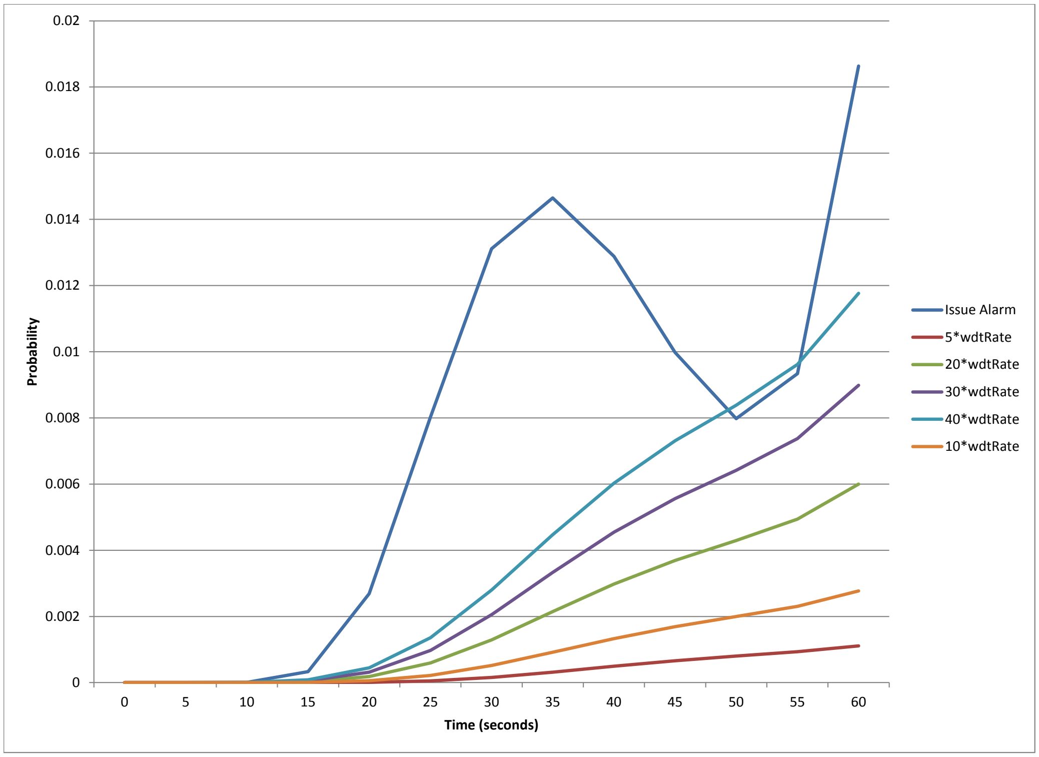

Figure 5.14 Probability of detecting an alarm with different threshold rate constants focused on early time.

eters to meet different threshold and accuracy needs. In the future, we hope to connect the threshold detector to a device that outputs a target molecule, such as the watchdog timer, and analyze its accuracy in an experimental laboratory.

CHAPTER 6. SIMPLE DSD EXPERIMENT

6.1 Motivation

We perform a simple DSD experiment as an initial step towards implementing a DNA watchdog timer. When learning a new programming language, the first program we write is usually a “Hello World” program which is a program that simply outputs “Hello World” to the standard output of the computer. We implement a simple DNA experiment for the same reason: to convince ourselves that we are able to model and perform DSD experiments.

6.2 Defining the Experiment

The simple DSD experiment was performed in collaboration with Eric Henderson’s Lab at Iowa State University and with Divita Mathur.

We define a single DSD reaction as a CRN:

$$AB + C \xrightarrow{\text{k}} AC + B \tag{6.1}$$

Briefly, two single stranded DNA strands A and B are initially bound together, forming species AB. Strand C attaches to a toehold on A and displaces strand B, leaving A and C bound together, forming species AC, and B alone. We created a model of this single reaction in both PRISM and in SMART; both model checkers are used to increase confidence in the accuracy of our models. We also create a set of DNA strands to implement the reaction in an experiment. We use the models to gain information about the experiment, in the hope of decreasing the amount of work needed for the biologists to create the DNA strands and perform the experiment.

6.3 Modeling the Experiment

The goal of our modeling effort is to generate information for our biologists to use in implementing the system. Specifically, we use the model to determine what length of toeholds to use in the implementation. Since the toehold length changes the rate and therefore expected time of a reaction, we want to specify a toehold length that allows enough time for sufficient observation of the experiment, but is short enough to minimize the overall length of the experiment. Simply put, we want the experiment to run for the shortest length of time where we can still perform enough observations. We also generate an expected time until a threshold of reactions has completed, which gives insight into how long the experiment should be run.

We create a model in PRISM and a model in SMART, and solve the Ordinary Differential Equations (ODEs) for the reaction [19]. Each model represents a single chemical reaction of the form:

$$A + B \overset{\text{k}}{\rightarrow} C \tag{6.2}$$

A represents the bound pair of strands, B represents the displacing strand and C represents the displaced state reached when an A and B have reacted, which is the final state of the reaction. We choose this simplification because the two strands are bound together at initialization acting as a single molecule.

We define the following variables for our models:

- *numDS*: The number of bound pairs (A, dual-stranded molecules) present in the solution at initialization.
- *numSS*: The number of displacer strands (B, single-stranded molecules) in the solution at initialization.
- *concDS*: The concentration of bound pairs in the solution at initialization.
- *volume*: The volume of the solution. The volume is derived from *numDS*, *concDS*, and Avogadro's constant, $6.02214129x10^{2}3mol^{-1}$, as follows:

$$Volume = \frac{numDS}{(concDS * Avogadro)} \tag{6.3}$$

- *maRC*: The mass action rate constant of the reaction.
- *rate*: The stochastic rate constant which is derived from *maRC* as follows:

$$\frac{maRC}{Avogadro} \tag{6.4}$$

We assign values to these constants to perform verification. For *maRC* we use a rate constant, $4.6773x10^4 M^{-1}s^{-1}$, taken from [33]. We choose *concDS* to be $100x10^{-9}M$, 100 nanomolars, based on advice from E. Henderson and D. Mathur. We can choose any integer for both *numDS* and *numSS*. Because volume is defined as a function of the number and concentration of bound pairs the expected runtime of the reaction will only slightly vary. Regardless of the value chosen for *numDS*, a larger number more closely represents a real system.

We use the following property to generate the expected time to 90% completion.

- $prob\_acc(tk(DS) > (0.1 * initDS), 0, infinity)$
  - Condition: The number of dual-stranded molecules (A) is greater than 10% of the initial concentration of dual-stranded molecules (A).
  - This returns the amount of measured in seconds that the condition is true.

Figures 6.1 and 6.2 show the expected time for the reaction to reach 90% completion with a *numDS* of 10,000 and a numSS of 50,000. Figure 6.1 shows the expected times for toeholds of length one through six. Figure 6.2 shows a closer view of toeholds four through six. The PRISM and SMART models return the same values and agree with the ODE solution [19].

Using these results, we choose a toehold length of four to implement the reaction. With a toehold of length four, the reaction takes approximately 112 seconds to reach 90% completion according to the SMART model, giving enough time to take multiple observations of the reaction, while being short enough that there is little downtime.

6.4 DSD Experiment

6.4.1 Creating the DNA Strands

D. Mathur designed and provided a set of DNA strands to implement the reaction:

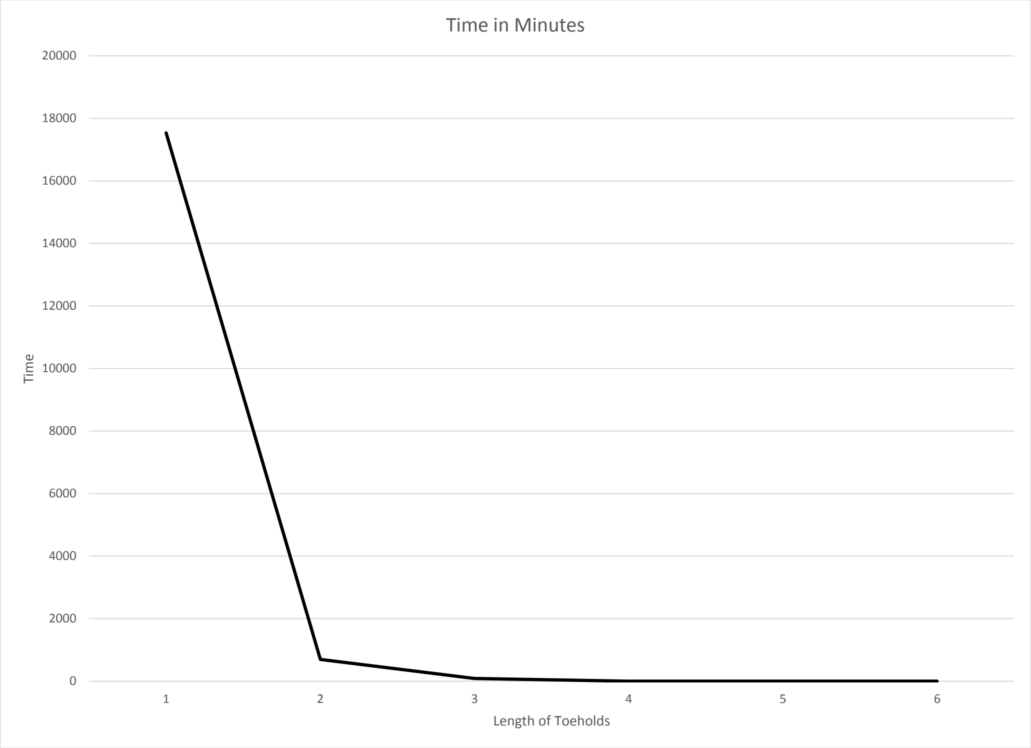

Figure 6.1 Expected time to 90% completion for all six toehold lengths.

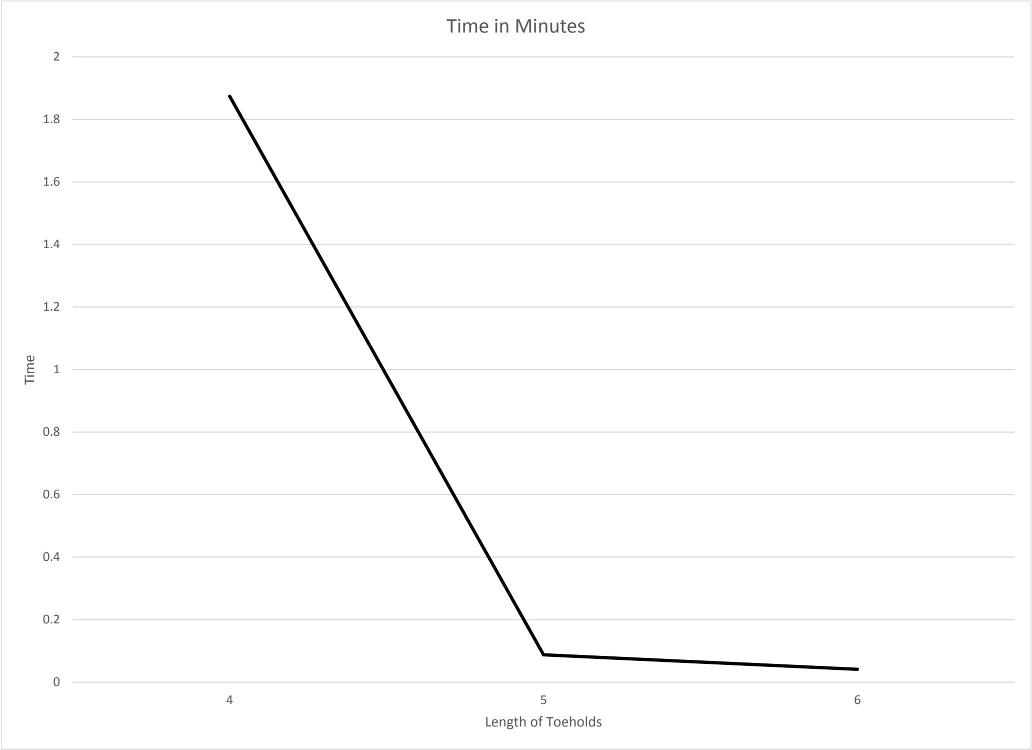

Figure 6.2 Expected time to 90% completion for four through six toehold lengths.

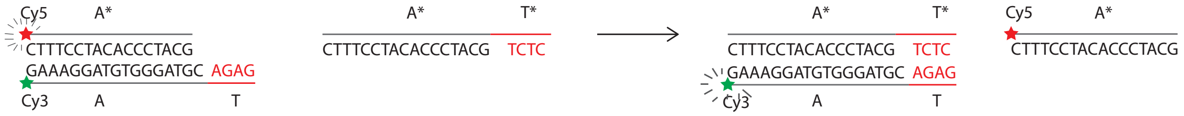

Figure 6.3 DNA strands used in the Simple DSD experiment (strands provided by D. Mathur)

- AT: Strand 1 is comprised of a domain A and a toehold T.
- A^*: Strand 2 is comprised of the complementary domain of A.
- A^*T^*: Strand 3 is comprised of the complementary domains of A and T.

The nucleotide sequences of the DNA strands can be seen in Figure 6.3. Cy3 and Cy5 are fluorophores that, when light of specific wavelengths are applied to them, give off different wavelengths of light. Cy3 and Cy5 are attached to the strands as shown in Figure 6.3. Cy5 absorbs the light that Cy3 gives off to emit light. When Cy3 and Cy5 are close, as they are when AT and A^* are attached, we only see light of Cy5's wavelength. By measuring the amount of light given off by Cy3 and Cy5, we can observe how many A^* have been displaced from AT, as the intensity of Cy5 light will go down, and Cy3 will go up.

6.4.2 Experimental Setup

The experiment is performed at room temperature with a concentration of 12.5 micromoles (uM) of magnesium. We use a concentration of 100 nanomoles (nM) of $AT - A*$ and 500 nM of A^*T^*. We use a Hyperspectral microscope to analyze the light emissions from the solution. To perform the experiment, we add 6 microliters (ul) of $AT - A^*$ at 200 nM and 6 ul of A^*T^* at 1 uM concentration. We then allow the reaction to sit for a number of seconds before measuring the light emissions.

6.4.3 Results

Figure 6.4 shows the intensity of light at different wavelengths. Cy3 emits light around the 570 nm wavelength and Cy5 emits light around the 670 nm wavelength, which can be seen as the left and right peaks, respectively. Each colored line represents a different time point at

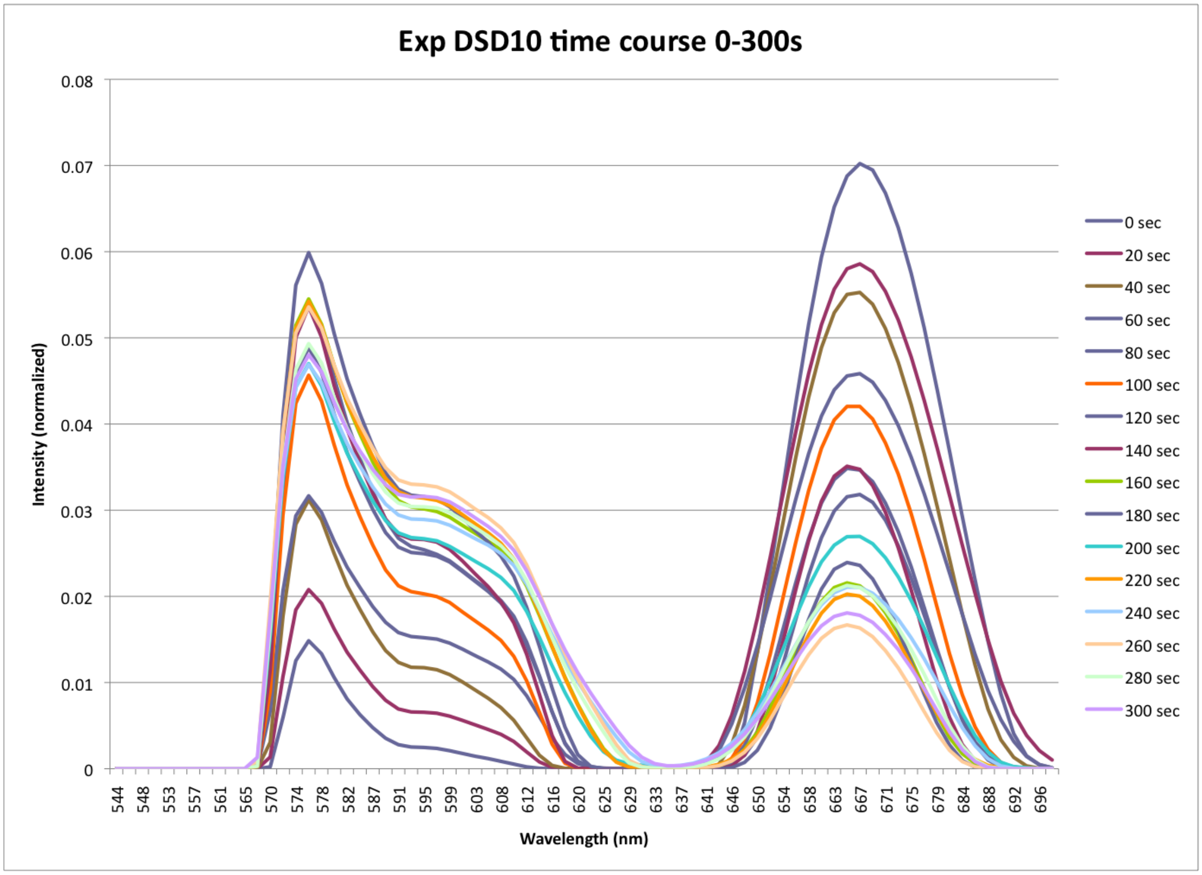

Figure 6.4 Results of DSD Experiment (provided by D. Mathur)

which a measurement was taken. As shown by the graph, the light emitted by Cy5 decreases over time while the light emitted by Cy3 increases. The change in light emissions shows the progression of the reaction.

The results indicate that the reaction progressed as intended. However, we are not yet certain the results match our model. The fluorophores attached to the DNA strands suffer from photo-bleaching; when they are exposed to light multiple times, they start to give off smaller intensities of light. To avoid photo-bleaching, we created a new sample for each observation time and waited the specified amount of time before exposing the sample to light. We were able to eliminate the photo-bleaching but this also introduces the possibility of error because we are using multiple samples.

Compared to initial plans to run the experiment for a few hours, the models show that we only need to run the experiment for a few minutes. Time estimates are based on the length of the toeholds which, through modeling, we determined to be the correct length to suit our needs. Even with an experiment as simple as this, performing some basic model checking saved a lot of time. We reduced trial and error to find the right toehold length. We lowered the experimental cost through saving laboratory time and reducing the number of DNA strands we needed to order.

CHAPTER 7. CONCLUSION

Goal-oriented requirements engineering [17, 18] proved to be useful in developing a molecular watchdog timer. The requirements engineering process helped to identify key failures in the initial watchdog timer design and provided insight into what verification was needed to ensure the watchdog timer satisfies all defined goals. Goal refinement and agent assignment defined a set of components needed in order to satisfy each of the goals, streamlining the design process.

The unary ladder-based watchdog timer satisfies all of the goals defined in Figure 3.1. The unary ladder-based watchdog timer is simple, which reduces the probability that errors occur. Maintaining a low probability of errors is important since the molecular watchdog timer is designed for safety-critical molecular systems. A user of the molecular watchdog timer must have control over the rate of success and failure of the watchdog timer in order to customize it to their needs. We have shown through model checking that the parameters in the unary ladder-based design can be used to customize the performance of the watchdog timer to meet the needs of a wide variety of molecular systems. Similarly, the threshold detector can be programmed to accommodate the watchdog requirements of various molecular systems. We also have shown through model checking that the probability of failure of a threshold detector can be controlled through the manipulation of its parameter values.

Goal-oriented obstacle analysis on the design of the molecular watchdog timer led to the development of the threshold detector. Further obstacle analysis provided insight into the difficulties of combining the delay component of the watchdog timer with the threshold detector, leading to more design changes. Future work will investigate how to avoid and overcome the difficulties involved in composing two or more molecular systems.

APPENDIX A.   PRISM MODEL OF THE WATCHDOG TIMER

```
ctmc
///////////////////////////////////
// Constant Variables
///////////////////////////////////
const int MAX_Y = 5;
const int MAX_H = 50;
const int MAX_L = 5;
const int NUM_HB = 1;
const int added_H = 10*MAX_L;

const double avo = 6.0221413e23;
const double maLR = 1.3e6;
//Volume is : MAX_L/(concLadd*AVO)*1000 (to allow prism to not round to 0)
const double volume = 8.30269e-14; //Calculated outside of PRISM
//const double ladderRate = maLR/(avo*volume);
const double ladderRate = 2.6e-5; // Calculated as (maLR/(avo*volume))
const double HBrate = ladderRate;

//const double expecTime = 4*(1/ladderRate);
const double expecTime = 36965.8;//Calculated outside of PRISM
const double expHBTime = expecTime/2;
const double hbinputRate = 1/expHBTime;
```

```
///////////////////////////////////////////////////////
// Initial Configuration
///////////////////////////////////////////////////////
const int INIT_L1 = MAX_L;
const int INIT_U = MAX_L;

// Ladder Rungs
module reactions
L1 : [0..MAX_L] init INIT_L1;
L2 : [0..MAX_L] init 0;
L3 : [0..MAX_L] init 0;
L4 : [0..MAX_L] init 0;
u : [0..MAX_L] init INIT_U;
Y : [0..MAX_Y] init 0;
HB : [0..NUM_HB] init NUM_HB;
H : [0..MAX_H] init 0;

[L1_U_to_L2_U] (L1 > 0) & (L2 < MAX_L) & (u > 0) ->
            (L1' = L1 - 1) & (L2' = L2 + 1);
[L2_U_to_L3_U] (L2 > 0) & (L3 < MAX_L) & (u > 0) ->
            (L2' = L2 - 1) & (L3' = L3 + 1);
[L3_U_to_L4_U] (L3 > 0) & (L4 < MAX_L) & (u > 0) ->
            (L3' = L3 - 1) & (L4' = L4 + 1);
[L4_U_to_Y_U] (L4 > 0) & (Y < MAX_Y) & (u > 0) ->
            (L4' = L4 - 1) & (Y' = Y + 1);

[L1_H_to_L1] (L1 > 0) & (H > 0) -> (H' = H - 1);
[L2_H_to_L1] (L2 > 0) & (H > 0) & (L1 < MAX_L) ->
            (H' = H - 1) & (L2' = L2 - 1) & (L1' = L1 + 1);
```

```
[L3_H_to_L1] (L3 > 0) & (H > 0) & (L1 < MAX_L) ->
            (H’ = H - 1) & (L3’ = L3 - 1) & (L1’ = L1 + 1);
[L4_H_to_L1] (L4 > 0) & (H > 0) & (L1 < MAX_L) ->
            (H’ = H - 1) & (L4’ = L4 - 1) & (L1’ = L1 + 1);
[Y_H_to_L1] (Y > 0) & (H > 0) & (L1 < MAX_L) ->
            (H’ = H - 1) & (Y’ = Y - 1) & (L1’ = L1 + 1);

[HBinput] (HB > 0) & (H <= MAX_H - added_H) ->
            (HB’ = HB - 1) & (H’ = H + added_H);
endmodule

// Reaction Rates Module
module reaction_rates
[L1_U_to_L2_U] L1 * u > 0 -> (ladderRate * L1 * u) : true;
[L2_U_to_L3_U] L2 * u > 0 -> (ladderRate * L2 * u) : true;
[L3_U_to_L4_U] L3 * u > 0 -> (ladderRate * L3 * u) : true;
[L4_U_to_Y_U] L4 * u > 0 -> (ladderRate * L4 * u) : true;

[L1_H_to_L1] L1 * H > 0 -> (HBrate * L1 * H) : true;
[L2_H_to_L1] L2 * H > 0 -> (HBrate * L2 * H) : true;
[L3_H_to_L1] L3 * H > 0 -> (HBrate * L3 * H) : true;
[L4_H_to_L1] L4 * H > 0 -> (HBrate * L4 * H) : true;
[Y_H_to_L1] Y * H > 0 -> (HBrate * Y * H) : true;

[HBinput] HB > 0 -> (hbinputRate) : true;
endmodule
```

APPENDIX B. SMART MODEL OF THE WATCHDOG TIMER

```
real avogadro := 6.0221413e23;
int HBladratio := 10;

//2-Rung Ladder no HB
pn noHB2(int n, real lrate, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
real ladder_rate := lrate/(avogadro); //ladder climb rate constant

place L1, L2, U, Y;
init(L1:n, U:n);
trans T0, T1;
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:Y:1, U:T1:1, T1:U:1
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume)
);

real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
```

```
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};

//2-Rung WDT
pn wdt2(int n, real lrate, real hrate, real expecTime, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
int m := HBladratio*n; //heartbeat in 10 times number of ladders
//int m := 0; //no heartbeat
real ladder_rate := lrate/(avogadro); //ladder climb rate constant
real hreset_rate := hrate/(avogadro); //hb reset rate constant

real expected_time := expecTime;
real hb_input2_rate := 1/(expected_time*(1/2));

place L1, L2, U, Y, H, HB;
init(L1:n, U:n, HB:m);
trans T0, T1, T2, T3, T4, T5; // For 1 HB
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:Y:1, U:T1:1, T1:U:1,
L1:T2:1, H:T2:1, T2:L1:1,
L2:T3:1, H:T3:1, T3:L1:1,
H:T5:1, Y:T5:1, T5:L1:1,
HB:T4:m, T4:H:m //For 1 HB
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),
T2 : expo(hreset_rate * tk(H) * tk(L1) / volume),
T3 : expo(hreset_rate * tk(H) * tk(L2) / volume),
T5 : expo(hreset_rate * tk(H) * tk(Y) / volume),
T4 : expo(hb_input2_rate)
);

real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};
//End 2-rung WDT



//4-Rung Ladder no HB
pn noHB4(int n, real lrate, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
real ladder_rate := lrate/(avogadro); //ladder climb rate constant

place L1, L2, L3, L4, U, Y;
init(L1:n, U:n);
trans T0, T1, T2, T3;
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:L3:1, U:T1:1, T1:U:1,
L3:T2:1, T2:L4:1, U:T2:1, T2:U:1,
L4:T3:1, T3:Y:1, U:T3:1, T3:U:1
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),
T2 : expo(ladder_rate * tk(L3) * tk(U) / volume),
T3 : expo(ladder_rate * tk(L4) * tk(U) / volume)
);

real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};

//4-Rung WDT
pn wdt4(int n, real lrate, real hrate, real expecTime, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
int m := HBladratio*n; //heartbeat in 10 times number of ladders
real ladder_rate := lrate/(avogadro); //ladder climb rate constant
real hreset_rate := hrate/(avogadro); //hb reset rate constant

real expected_time := expecTime;
//real hb_input2_rate := 1/(expected_time*(1/4));
real hb_input3_rate := 1/(expected_time*(1/2));

place L1, L2, L3, L4, U, Y, H, HB1, HB2;
```

```
//init(L1:n, U:n, HB1:1, HB2:1);

init(L1:n, U:n, HB2:1);

trans T0, T1, T2, T3, T4, T5, T6, T7,/* T9,*/ T11, T10; // For 1 HB

arcs(

L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,

L2:T1:1, T1:L3:1, U:T1:1, T1:U:1,

L3:T2:1, T2:L4:1, U:T2:1, T2:U:1,

L4:T3:1, T3:Y:1, U:T3:1, T3:U:1,

L1:T4:1, H:T4:1, T4:L1:1,

L2:T5:1, H:T5:1, T5:L1:1,

L3:T6:1, H:T6:1, T6:L1:1,

L4:T7:1, H:T7:1, T7:L1:1,

Y:T11:1, H:T11:1, T11:L1,

//HB1:T9:1, T9:H:m,

HB2:T10:1, T10:H:m //For 1 HB

);

firing(

T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),

T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),

T2 : expo(ladder_rate * tk(L3) * tk(U) / volume),

T3 : expo(ladder_rate * tk(L4) * tk(U) / volume),

T4 : expo(hreset_rate * tk(H) * tk(L1) / volume),

T5 : expo(hreset_rate * tk(H) * tk(L2) / volume),

T6 : expo(hreset_rate * tk(H) * tk(L3) / volume),

T7 : expo(hreset_rate * tk(H) * tk(L4) / volume),

T11 : expo(hreset_rate * tk(H) * tk(Y) / volume),

//T9 : expo(hb_input2_rate),

T10 : expo(hb_input3_rate)

);
```

```
real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};
//End 4-Rung WDT

//6-Rung Ladder no HB
pn noHB6(int n, real lrate, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
real ladder_rate := lrate/(avogadro); //ladder climb rate constant

place L1, L2, L3, L4, L5, L6, U, Y;
init(L1:n, U:n);
trans T0, T1, T2, T3, T4, T5;
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:L3:1, U:T1:1, T1:U:1,
L3:T2:1, T2:L4:1, U:T2:1, T2:U:1,
L4:T3:1, T3:L5:1, U:T3:1, T3:U:1,
L5:T4:1, T4:L6:1, U:T4:1, T4:U:1,
L6:T5:1, T5:Y:1, U:T5:1, T5:U:1
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),
T2 : expo(ladder_rate * tk(L3) * tk(U) / volume),
T3 : expo(ladder_rate * tk(L4) * tk(U) / volume),
T4 : expo(ladder_rate * tk(L5) * tk(U) / volume),
T5 : expo(ladder_rate * tk(L6) * tk(U) / volume)
);

real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};

//6-Rung WDT
pn wdt6(int n, real lrate, real hrate, real expecTime, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
int m := HBladratio*n; //heartbeat in 10 times number of ladders
//int m := 0; //no heartbeat
real ladder_rate := lrate/(avogadro); //ladder climb rate constant
real hreset_rate := hrate/(avogadro); //hb reset rate constant

real expected_time := expecTime;
real hb_input2_rate := 1/(expected_time*(1/2));

place L1, L2, L3, L4, L5, L6, U, Y, H, HB;
init(L1:n, U:n, HB:m);
trans T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13;
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:L3:1, U:T1:1, T1:U:1,
L3:T2:1, T2:L4:1, U:T2:1, T2:U:1,
L4:T3:1, T3:L5:1, U:T3:1, T3:U:1,
L5:T4:1, T4:L6:1, U:T4:1, T4:U:1,
L6:T5:1, T5:Y:1, U:T5:1, T5:U:1,
L1:T6:1, H:T6:1, T6:L1:1,
L2:T7:1, H:T7:1, T7:L1:1,
L3:T8:1, H:T8:1, T8:L1:1,
L4:T9:1, H:T9:1, T9:L1:1,
L5:T10:1, H:T10:1, T10:L1:1,
L6:T11:1, H:T11:1, T11:L1:1,
Y:T13:1, H:T13:1, T13:L1:1,
HB:T12:m, T12:H:m //For 1 HB
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),
T2 : expo(ladder_rate * tk(L3) * tk(U) / volume),
T3 : expo(ladder_rate * tk(L4) * tk(U) / volume),
T4 : expo(ladder_rate * tk(L5) * tk(U) / volume),
T5 : expo(ladder_rate * tk(L6) * tk(U) / volume),
T6 : expo(hreset_rate * tk(H) * tk(L1) / volume),
T7 : expo(hreset_rate * tk(H) * tk(L2) / volume),
T8 : expo(hreset_rate * tk(H) * tk(L3) / volume),
T9 : expo(hreset_rate * tk(H) * tk(L4) / volume),
T13 : expo(hreset_rate * tk(H) * tk(Y) / volume),
T10 : expo(hreset_rate * tk(H) * tk(L5) / volume),
T11 : expo(hreset_rate * tk(H) * tk(L6) / volume),
T12 : expo(hb_input2_rate)
);
```

```
real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};
//End 6-Rung WDT

//8-Rung Ladder no HB
pn noHB8(int n, real lrate, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
real ladder_rate := lrate/(avogadro); //ladder climb rate constant

place L1, L2, L3, L4, L5, L6, L7, L8, U, Y;
init(L1:n, U:n);
trans T0, T1, T2, T3, T4, T5, T6, T7;
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:L3:1, U:T1:1, T1:U:1,
L3:T2:1, T2:L4:1, U:T2:1, T2:U:1,
L4:T3:1, T3:L5:1, U:T3:1, T3:U:1,
L5:T4:1, T4:L6:1, U:T4:1, T4:U:1,
L6:T5:1, T5:L7:1, U:T5:1, T5:U:1,
L7:T6:1, T6:L8:1, U:T6:1, T6:U:1,
L8:T7:1, T7:Y:1, U:T7:1, T7:U:1
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),
T2 : expo(ladder_rate * tk(L3) * tk(U) / volume),
T3 : expo(ladder_rate * tk(L4) * tk(U) / volume),
T4 : expo(ladder_rate * tk(L5) * tk(U) / volume),
T5 : expo(ladder_rate * tk(L6) * tk(U) / volume),
T6 : expo(ladder_rate * tk(L7) * tk(U) / volume),
T7 : expo(ladder_rate * tk(L8) * tk(U) / volume)
);

real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};

//8-Rung WDT
pn wdt8(int n, real lrate, real hrate, real expecTime, real t) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);//*1000;
        //Volume generated by the concentration of ladders.
int m := HBladratio*n; //heartbeat in 10 times number of ladders
//int m := 0; //no heartbeat
real ladder_rate := lrate/(avogadro); //ladder climb rate constant
real hreset_rate := hrate/(avogadro); //hb reset rate constant

real expected_time := expecTime;
real hb_input2_rate := 1/(expected_time*(1/2));

place L1, L2, L3, L4, L5, L6, L7, L8, U, Y, H, HB;
init(L1:n, U:n, HB:m);
```

```
trans T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13,
            T14, T15, T16, T17; // For 1 HB
arcs(
L1:T0:1, T0:L2:1, U:T0:1, T0:U:1,
L2:T1:1, T1:L3:1, U:T1:1, T1:U:1,
L3:T2:1, T2:L4:1, U:T2:1, T2:U:1,
L4:T3:1, T3:L5:1, U:T3:1, T3:U:1,
L5:T4:1, T4:L6:1, U:T4:1, T4:U:1,
L6:T5:1, T5:L7:1, U:T5:1, T5:U:1,
L7:T6:1, T6:L8:1, U:T6:1, T6:U:1,
L8:T7:1, T7:Y:1, U:T7:1, T7:U:1,

L1:T8:1, H:T8:1, T8:L1:1,
L2:T9:1, H:T9:1, T9:L1:1,
L3:T10:1, H:T10:1, T10:L1:1,
L4:T11:1, H:T11:1, T11:L1:1,
L5:T12:1, H:T12:1, T12:L1:1,
L6:T13:1, H:T13:1, T13:L1:1,
L7:T14:1, H:T14:1, T14:L1:1,
L8:T15:1, H:T15:1, T15:L1:1,
Y:T17:1, H:T17:1, T17:L1:1,
HB:T16:m, T16:H:m //For 1 HB
);
firing(
T0 : expo(ladder_rate * tk(L1) * tk(U) / volume),
T1 : expo(ladder_rate * tk(L2) * tk(U) / volume),
T2 : expo(ladder_rate * tk(L3) * tk(U) / volume),
T3 : expo(ladder_rate * tk(L4) * tk(U) / volume),
T4 : expo(ladder_rate * tk(L5) * tk(U) / volume),
T5 : expo(ladder_rate * tk(L6) * tk(U) / volume),
T6 : expo(ladder_rate * tk(L7) * tk(U) / volume),
T7 : expo(ladder_rate * tk(L8) * tk(U) / volume),

T8 : expo(hreset_rate * tk(H) * tk(L1) / volume),
T9 : expo(hreset_rate * tk(H) * tk(L2) / volume),
T10 : expo(hreset_rate * tk(H) * tk(L3) / volume),
T11 : expo(hreset_rate * tk(H) * tk(L4) / volume),
T12 : expo(hreset_rate * tk(H) * tk(L5) / volume),
T13 : expo(hreset_rate * tk(H) * tk(L6) / volume),
T14 : expo(hreset_rate * tk(H) * tk(L7) / volume),
T15 : expo(hreset_rate * tk(H) * tk(L8) / volume),
T17 : expo(hreset_rate * tk(H) * tk(Y) / volume),
T16 : expo(hb_input2_rate)
);

real time := prob_acc(!(tk(Y)>=(0.8*n)), 0, infinity);
        //Time less then 80% of ladders have completed
real probAlarmatT := prob_at(tk(Y)>=(0.8*n), t);
};
//End 8-Rung WDT
```

APPENDIX C. SMART MODEL OF THE THRESHOLD DETECTOR

```
real avogadro := 6.0221413e23;
pn thresh2(int n, int m, real lrate, real hrate, real t, int r) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);
        //Volume generated by the concentration of ladders.
real climb_rate := lrate/(avogadro); //ladder climb rate constant
real reset_rate := hrate/(avogadro); //hb reset rate constant

//For 2+ rungs
place L1, L2, Y, R, D;
init(L1:n, R:r, Y:m);
trans T0, T1, T4, T5;

arcs(
L1:T0:1, T0:L2:1, Y:T0:1, T0:Y:1,
L2:T1:1, T1:D:1, Y:T1:1, T1:Y:1,

L1:T4:1, R:T4:1, T4:L1:1, T4:R:1,
L2:T5:1, R:T5:1, T5:L1:1, T5:R:1
);
firing(
T0 : expo(climb_rate * tk(L1) * tk(Y) / volume),
T1 : expo(climb_rate * tk(L2) * tk(Y) / volume),
T4 : expo(reset_rate * tk(R) * tk(L1) / volume),
T5 : expo(reset_rate * tk(R) * tk(L2) / volume)
);

real probDatT := prob_at(tk(D)!=0, t);
        //probability D is not 0 at specified time
real expTime := prob_acc(tk(D)==0, 0, infinity);
        //expected time until D appears
};

pn thresh4(int n, int m, real lrate, real hrate, real t, int r) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);
        //Volume generated by the concentration of ladders.
real climb_rate := lrate/(avogadro); //ladder climb rate constant
real reset_rate := hrate/(avogadro); //hb reset rate constant

//For 4+ rungs
place L1, L2, L3, L4, Y, R, D;
init(L1:n, R:r, Y:m);
trans T0, T1, T2, T3, T4, T5, T6, T7;

arcs(
L1:T0:1, T0:L2:1, Y:T0:1, T0:Y:1,
L2:T1:1, T1:L3:1, Y:T1:1, T1:Y:1,
L3:T2:1, T2:L4:1, Y:T2:1, T2:Y:1,
L4:T3:1, T3:D:1, Y:T3:1, T3:Y:1,

L1:T4:1, R:T4:1, T4:L1:1, T4:R:1,
L2:T5:1, R:T5:1, T5:L1:1, T5:R:1,
L3:T6:1, R:T6:1, T6:L1:1, T6:R:1,
L4:T7:1, R:T7:1, T7:L1:1, T7:R:1
);
firing(
T0 : expo(climb_rate * tk(L1) * tk(Y) / volume),
T1 : expo(climb_rate * tk(L2) * tk(Y) / volume),
T2 : expo(climb_rate * tk(L3) * tk(Y) / volume),
T3 : expo(climb_rate * tk(L4) * tk(Y) / volume),
T4 : expo(reset_rate * tk(R) * tk(L1) / volume),
T5 : expo(reset_rate * tk(R) * tk(L2) / volume),
T6 : expo(reset_rate * tk(R) * tk(L3) / volume),
T7 : expo(reset_rate * tk(R) * tk(L4) / volume)
);

real probDatT := prob_at(tk(D)!=0, t);
        //probability D is not 0 at specified time
real expTime := prob_acc(tk(D)==0, 0, infinity);
        //expected time until D appears
};

pn thresh6(int n, int m, real lrate, real hrate, real t, int r) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);
        //Volume generated by the concentration of ladders.
real climb_rate := lrate/(avogadro); //ladder climb rate constant
real reset_rate := hrate/(avogadro); //hb reset rate constant

place L1, L2, L3, L4, L5, L6, Y, R, D;
```

```
init(L1:n, R:r, Y:m);
trans T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11;


arcs(
//Climb up
L1:T0:1, T0:L2:1, Y:T0:1, T0:Y:1,
L2:T1:1, T1:L3:1, Y:T1:1, T1:Y:1,
L3:T2:1, T2:L4:1, Y:T2:1, T2:Y:1,
L4:T3:1, T3:L5:1, Y:T3:1, T3:Y:1,
L5:T8:1, T8:L6:1, Y:T8:1, T8:Y:1,
L6:T9:1, T9:D:1, Y:T9:1, T9:Y:1,


//Fall Back
L1:T4:1, R:T4:1, T4:L1:1, T4:R:1,
L2:T5:1, R:T5:1, T5:L1:1, T5:R:1,
L3:T6:1, R:T6:1, T6:L1:1, T6:R:1,
L4:T7:1, R:T7:1, T7:L1:1, T7:R:1,
L5:T10:1, R:T10:1, T10:L1:1, T10:R:1,
L6:T11:1, R:T11:1, T11:L1:1, T11:R:1
);
firing(
T0 : expo(climb_rate * tk(L1) * tk(Y) / volume),
T1 : expo(climb_rate * tk(L2) * tk(Y) / volume),
T2 : expo(climb_rate * tk(L3) * tk(Y) / volume),
T3 : expo(climb_rate * tk(L4) * tk(Y) / volume),
T8 : expo(climb_rate * tk(L5) * tk(Y) / volume),
T9 : expo(climb_rate * tk(L6) * tk(Y) / volume),
T4 : expo(reset_rate * tk(R) * tk(L1) / volume),
T5 : expo(reset_rate * tk(R) * tk(L2) / volume),
T6 : expo(reset_rate * tk(R) * tk(L3) / volume),
T7 : expo(reset_rate * tk(R) * tk(L4) / volume),
T10 : expo(reset_rate * tk(R) * tk(L5) / volume),
T11 : expo(reset_rate * tk(R) * tk(L6) / volume)
);

real probDatT := prob_at(tk(D)!=0, t);
        //probability D is not 0 at specified time
real expTime := prob_acc(tk(D)==0, 0, infinity);
        //expected time until D appears
};


pn thresh8(int n, int m, real lrate, real hrate, real t, int r) := {
real concLad := 100e-9;
real volume := n/(concLad*avogadro);
        //Volume generated by the concentration of ladders.
real climb_rate := lrate/(avogadro); //ladder climb rate constant
real reset_rate := hrate/(avogadro); //hb reset rate constant

//8+ rungs
place L1, L2, L3, L4, L5, L6, L7, L8, Y, R, D;
init(L1:n, R:r, Y:m);
trans T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15;

arcs(
//Climb up
L1:T0:1, T0:L2:1, Y:T0:1, T0:Y:1,
L2:T1:1, T1:L3:1, Y:T1:1, T1:Y:1,
L3:T2:1, T2:L4:1, Y:T2:1, T2:Y:1,
L4:T3:1, T3:L5:1, Y:T3:1, T3:Y:1,
L5:T8:1, T8:L6:1, Y:T8:1, T8:Y:1,
L6:T9:1, T9:L7:1, Y:T9:1, T9:Y:1,
L7:T12:1, T12:L8:1, Y:T12:1, T12:Y:1,
L8:T13:1, T13:D:1, Y:T13:1, T13:Y:1,

//Fall Back
L1:T4:1, R:T4:1, T4:L1:1, T4:R:1,
L2:T5:1, R:T5:1, T5:L1:1, T5:R:1,
L3:T6:1, R:T6:1, T6:L1:1, T6:R:1,
L4:T7:1, R:T7:1, T7:L1:1, T7:R:1,
L5:T10:1, R:T10:1, T10:L1:1, T10:R:1,
L6:T11:1, R:T11:1, T11:L1:1, T11:R:1,
L7:T14:1, R:T14:1, T14:L1:1, T14:R:1,
L8:T15:1, R:T15:1, T15:L1:1, T15:R:1



);
firing(
T0 : expo(climb_rate * tk(L1) * tk(Y) / volume),
T1 : expo(climb_rate * tk(L2) * tk(Y) / volume),
T2 : expo(climb_rate * tk(L3) * tk(Y) / volume),
T3 : expo(climb_rate * tk(L4) * tk(Y) / volume),
T8 : expo(climb_rate * tk(L5) * tk(Y) / volume),
T9 : expo(climb_rate * tk(L6) * tk(Y) / volume),
T12 : expo(climb_rate * tk(L7) * tk(Y) / volume),
T13 : expo(climb_rate * tk(L8) * tk(Y) / volume),
T4 : expo(reset_rate * tk(R) * tk(L1) / volume),
T5 : expo(reset_rate * tk(R) * tk(L2) / volume),
T6 : expo(reset_rate * tk(R) * tk(L3) / volume),
T7 : expo(reset_rate * tk(R) * tk(L4) / volume),
T10 : expo(reset_rate * tk(R) * tk(L5) / volume),
T11 : expo(reset_rate * tk(R) * tk(L6) / volume),
T14 : expo(reset_rate * tk(R) * tk(L7) / volume),
T15 : expo(reset_rate * tk(R) * tk(L8) / volume)
);


real probDatT := prob_at(tk(D)!=0, t);
        //probability D is not 0 at specified time
real expTime := prob_acc(tk(D)==0, 0, infinity);
        //expected time until D appears
real expTimetoL8 := prob_acc(tk(L8)==0, 0, infinity);
        //expected time until L8 appears
};
```

BIBLIOGRAPHY

[1] Athreya, K., and Lahiri, S. *Measure Theory and Probability Theory*. Springer Texts in Statistics. Springer, 2006.

[2] Ciardo, G., Jones, R., Miner, A., and Siminiceanu, R. Logical and stochastic modeling with smart. In *Computer Performance Evaluation. Modelling Techniques and Tools*, P. Kemper and W. Sanders, Eds., vol. 2794 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 78–97.

[3] Cook, M., Soloveichik, D., Winfree, E., and Bruck, J. Programmability of chemical reaction networks. *Algorithmic Bioprocesses* (2009), 543–584.

[4] Delbrck, M. Statistical fluctuations in autocatalytic reactions. *The Journal of Chemical Physics 8*, 1 (1940).

[5] Doll, T., Raman, S., Dey, R., and Burkhard, P. Nanoscale assemblies and their biomedical applications. *Journal of The Royal Society*, January (2013).

[6] Douglas, S. M., Bachelet, I., and Church, G. M. A logic-gated nanorobot for targeted transport of molecular payloads. *Science (New York, N.Y.) 335*, 6070 (Feb. 2012), 831–4.

[7] Érdi, P., and Tóth, J. *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models*. Nonlinear science : theory and applications. Manchester University Press, 1989.

[8] Gillespie, D. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry 81* (1977), 2340–2361.

[9] Gillespie, D. T. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications 188*, 1-3 (Sept. 1992), 404–425.

[10] Hjelmfelt, A., Weinberger, E., and Ross, J. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Acadamy of Science 88* (1991), 10983–10987.

[11] Jiang, H., Riedel, M., and Parhi, K. Synchronous sequential computation with molecular reactions. *Proceedings of the 48th Design Automation Conference on - DAC '11*, 1 (2011), 836–841.

[12] Jiang, H., Riedel, M., and Parhi, K. Digital signal processing with molecular reactions. *IEEE Design & Test of Computers 29*, 3 (June 2012), 21–31.

[13] Jiang, H., Salehi, S. A., Riedel, M. D., and Parhi, K. K. Discrete-time signal processing with DNA. *ACS synthetic biology 2*, 5 (May 2013), 245–54.

[14] Kharam, A., Jiang, H., Riedel, M., and Parhi, K. Binary counting with chemical reactions. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* (Jan. 2011), 302–13.

[15] Knight, J. *Fundamentals of Dependable Computing for Software Engineers*, 1st ed. Chapman & Hall/CRC, 2012.

[16] Kwiatkowska, M., Norman, G., and Parker, D. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)* (2011), G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806 of *LNCS*, Springer, pp. 585–591.

[17] Lamsweerde, A. V. Requirements engineering: from craft to discipline. *Proceedings 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (2008), 238–249.

[18] Lamsweerde, A. V. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons Ltd, Chichester, West Sussex, England, 2009.

[19] Lathrop, J. I., and Klinge, T. H. Personal Communication.

[20] Leveson, N. G. *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company, USA, 1995.

[21] Lutz, R., Lutz, J., Lathrop, J., Klinge, T., Henderson, E., Mathur, D., and Sheasha, D. A. Engineering and verifying requirements for programmable self-assembling nanomachines. *2012 34th International Conference on Software Engineering (ICSE)* (June 2012), 1361–1364.

[22] Lutz, R. R., Lutz, J. H., Lathrop, J. I., Klinge, T. H., Mathur, D., Stull, D. M., Bergquist, T. G., and Henderson, E. R. Requirements analysis for a product family of DNA nanodevices. *2012 20th IEEE International Requirements Engineering Conference (RE)* (Sept. 2012), 211–220.

[23] Magnasco, M. Chemical Kinetics is Turing Universal. *Physical Review Letters 78*, 6 (Feb. 1997), 1190–1193.

[24] McQuarrie, D. Stochastic approach to chemical kinetics. *Journal of Applied Probability 4*, 3 (1967), 413–478.

[25] Qian, L., Winfree, E., and Bruck, J. Neural network computation with DNA strand displacement cascades. *Nature 475*, 7356 (July 2011), 368–372.

[26] Smith, L. Nanotechnology: Molecular robots on the move. *Nature 465*, May (2010), 167–168.

[27] Soloveichik, D. Robust stochastic chemical reaction networks and bounded tau-leaping. *Journal of Computational Biology*, 1998 (2009), 1–29.

[28] Soloveichik, D., Cook, M., Winfree, E., and Bruck, J. Computation with finite stochastic chemical reaction networks. *Natural Computing 7*, 4 (Feb. 2008), 615–633.

[29] Soloveichik, D., Seelig, G., and Winfree, E. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences of the United States of America 107*, 12 (Mar. 2010), 5393–5398.

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

[31] Yurke, B. Using DNA to Power Nanostructures. 111–122.

[32] Yurke, B., Turber, A. J., Jr, A. P. M., Simmel, F. C., and Neumann, J. L. A DNA-fuelled molecular machine made of DNA. 605–608.

[33] Zhang, D. Y., and Seelig, G. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry 3*, 2 (Feb. 2011), 103–13.

[34] Zhang, D. Y., and Winfree, E. Control of DNA strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society 131*, 47 (Dec. 2009), 17303–14.