Automated Requirements Analysis for a Molecular Watchdog Timer

Samuel J. Ellis, Eric R. Henderson, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Divita Mathur, and Andrew S. Miner

Iowa State University
Ames, IA 50011, U.S.A.
{sjellis, telomere, tklinge, jil, lutz, rlutz, divita, asminer}@iastate.edu

ABSTRACT

Dynamic systems in DNA nanotechnology are often programmed using a chemical reaction network (CRN) model as an intermediate level of abstraction. In this paper, we design and analyze a CRN model of a watchdog timer, a device commonly used to monitor the health of a safety critical system. Our process uses incremental design practices with goal-oriented requirements engineering, software verification tools, and custom software to help automate the software engineering process. The watchdog timer is comprised of three components: an absence detector, a threshold filter, and a signal amplifier. These components are separately designed and verified, and only then composed to create the molecular watchdog timer. During the requirements-design iterations, simulation, model checking, and analysis are used to verify the system. Using this methodology several incomplete requirements and design flaws were found, and the final verified model helped determine specific parameters for biological experiments.

Keywords

probabilistic model checking; requirements engineering; molecular programming; chemical reaction networks

1. INTRODUCTION

Molecular programming, also called DNA nanotechnology, uses the information processing capabilities of DNA to engineer the self-assembly of nanoscale structures and devices. Born in the pioneering research of Seeman in the 1980s [42], molecular programming is now a large, multifaceted field in which teams of investigators from computer science, chemistry, molecular biology, mathematics, physics, and various engineering disciplines collaborate to design ever more elaborate nanosystems. Many envisioned DNA molecular programming applications will be safety critical. Examples include nanoscale bio-sensors to detect the presence of disease markers in the human body and of dangerous pollutants in air or water, and nanoscale drug therapy capsules that release their contents only when they encounter and bind to an associated tumor cell [31]. One such advance is a barrel-shaped DNA nanorobot programmed to autonomously navigate to a tissue sample, unlock its cargo compartment when it encounters the proper antigen key, and dispense appropriate antibody fragments [14]. Other applications to medicine, computer electronics, and biological instrumentation are anticipated, and many of these applications will be safety critical.

This paper concerns the reliable molecular programming–programming in the literal sense of computer science–of nanosystems that are *dynamic* and *nonstructural*. By "dynamic" we mean that the objective is not the self-assembly of static structures, but rather the self-assembly of devices that carry out desired *processes* at molecular scales. By "nonstructural" we mean that the devices are not internally connected structures, but are rather diffuse collections of molecules that carry out their tasks in well mixed solutions. To make these distinctions concrete by example, we do not in this paper consider the use of DNA tiles or DNA origami to create (static, not dynamic) two- and three-dimensional nanostructures [40, 41, 19, 51, 24, 35], and we do not consider the creation of (dynamic, but structural) molecular robots that walk on DNA origami tracks or deliver molecular payloads [32, 14]. We instead focus our attention on nanosystems that, like recently engineered logic circuits [38, 39] and many natural biological circuits [8], operate amorphously and probabilistically according to the laws of chemical kinetics.

A dynamic, nonstructural molecular process in which the presence or absence of very small numbers of certain types of molecules (e.g, a single copy of a viral genome in a living cell) may be significant is mathematically modeled by a (*stochastic*) *chemical reaction network* or, briefly, a *CRN*. (All CRNs in this paper are stochastic, so we omit "stochastic" from the terminology.) The CRN model, which goes back at least to 1940 [12], has three desirable features. First, it is mathematically simple. A CRN is a finite collection of *reactions*, each of which has a simple form such as $A + C \xrightarrow{k} B + D$, where the letters A,B, etc., represent abstract molecules that hide all other properties of the molecular species that they represent. Such a reaction says that a molecule of A and a molecule of C may collide and be consumed to produce a molecule of B and a molecule of D, where the rate at which the reaction occurs is determined by the positive real number k, the *rate constant* of the reaction. A *state* of a CRN is a vector specifying the number of each species present, and the dynamics of the CRN proceed as a continuous time Markov process with rates derived from the rate constants [2]. (See Appendix A for more details) A second feature of CRNs, discovered only recently, is that they are very general. Every algorithm can, in at least one sense, be simulated by a CRN [45].

A third desirable feature of CRNs, also discovered recently, is that they can be implemented in a uniform way using DNA strand displacement reactions [46]. This is fortuitous, because dynamic systems in DNA nanotechnology, including DNA walkers and logic circuits, are typically implemented using DNA strand displacement reactions [44, 54]. The details of strand displacement reactions are not crucial for this paper, but it is relevant to note that they are relatively easy to implement in the laboratory, and that they are easy to specify. In fact, there is a programming language, DSD, in which a large, expressive class of such reactions can be specified and compiled into explicit DNA sequences [37, 28]. Moreover, CRNs have recently been used as a higher level programming language that can be compiled into DSD [9].

Monitoring a dynamic, nonstructural molecular process for the occurrence of an event is relatively easy, because matters can be arranged so that molecules that are indicative of the event react in chance encounters with molecules of the monitoring process, thereby triggering a desired response to the event. In sharp contrast, monitoring such a molecular process for the *nonoccurrence* of an event involves timing issues that can be very hard to resolve [13]. This is why the above-mentioned logic circuits and many others use a "dual rail" scheme in which the presence of a signal is represented by the presence of one species, and the absence of this signal is represented by the *presence* of another species. Such dual rail schemes are useful and convenient for the ordinary operation of a process, but they cannot be relied upon to monitor the process for failures.

A *watchdog timer* is a device often used to monitor a safety-critical system and to issue an alarm if the system fails. In simple but typical instances, the watchdog timer monitors the system for expected, periodic "heartbeats" that indicate the system's well-being. If a heartbeat does not occur for too long, the system has failed, and the watchdog timer detects this and trips an alarm (i.e., it "barks") [25, 30].

In this paper we engineer a *molecular watchdog timer*, which is a chemical reaction network that monitors an external system for a regular heartbeat, indicated by the periodic introduction of "H" molecules into the ambient solution, and trips an alarm if a heartbeat does not occur for too long a time. Since the absence of a heartbeat is indicative of a system failure, it is *not* reasonable to assume that the system is representing its heartbeats in dual rail form. (If the systems fails to produce a heartbeat, it is also apt to fail to produce an indicator of the absence of the heartbeat.) Hence our molecular watchdog timer must, despite the above-mentioned difficulties, monitor for the *prolonged absence* of a heartbeat and respond accordingly.

The design of our molecular watchdog timer entailed a rigorous process of specifying the high-level reliability requirements (including very low probabilities of both false positives and false negatives); refining these high-level requirements into a detailed requirements model; proving the implications implicit in the requirements model; designing the molecular watchdog timer to achieve the low-level requirements; and model checking that it does so. In addition, the design of our molecular watchdog timer is modular, with each low-level requirement satisfied by one of the three components (absence detector, threshold filter, and signal amplifier); and the overall process was iterative and incremental, with both the requirements model and the design repeatedly revised in response to defects discovered by the process. Moreover, our entire process was adapted to the novel "computing in soup" environment of dynamic, nonstructural DNA nanotechnology. The final product, a fully verified, CRN-level design for a reliable molecular watchdog timer *together* with the process that achieved it, demonstrates the crucial role that rigorous software engineering processes and automated software engineering tools will play in the coming age of nanotechnology.

2. GOAL MODELING

The requirements for the molecular watchdog timer (MWT) were identified, refined, and validated incrementally using a goal-oriented requirements engineering approach, KAOS, in concert with automated formal analysis tools [15, 49]. The goal-based requirements engineering framework that we used produces as its initial artifact an AND/OR goal graph in which the top-level node describes the high-level functional requirement for the system to be built.

2.1 Goal graph refinement

Figure 1 shows an initial goal graph for the MWT, with the top-level goal of *Achieve[Alarm iff no Heartbeat is provided within t time]*. This goal describes the prescriptive intent that: *If and only if the monitored application does not provide the appropriate signal within a specified time, then the system shall issue an alarm.* The top-level goal is AND-refined into two subgoals, since it can be satisfied if and only if both of the two second-level goals are met. These subgoals are subsequently refined until each subgoal can be assigned to an agent. An agent is a system component with responsibility for satisfying the goal(s) assigned to it [49]. The goal graph was refined incrementally, both from the top down and from the bottom up, as our understanding of what was needed grew. As described in the next section, automated analysis tools (especially MATLAB's SimBiology package and the probabilistic model checker PRISM [26]) played essential roles in evaluating the realizability of the goals for the MWT, given the dynamic, non-deterministic CRN environment.

There are four environmental assumptions that must hold for satisfaction of the subgoals to mean that the parent goal is satisfied. These environmental properties are: (1) the environment controls when heartbeats are present; (2) a heartbeat is a "dose" of molecules of species "H", and the size of the heartbeat dose must be within a certain range; (3) no molecules in the environment other than the heartbeats interact with our system; and (4) the solution is well mixed. In other words, the MWT molecules do not react with other molecules in the environment. The correct formalization of the first and third of these properties emerged from the process of proving that satisfaction of the subgoals implies the satisfaction of the top-level goal. The allowable range of the size of the heartbeat dose in the second property depends on the values assigned to several other parameters, such as

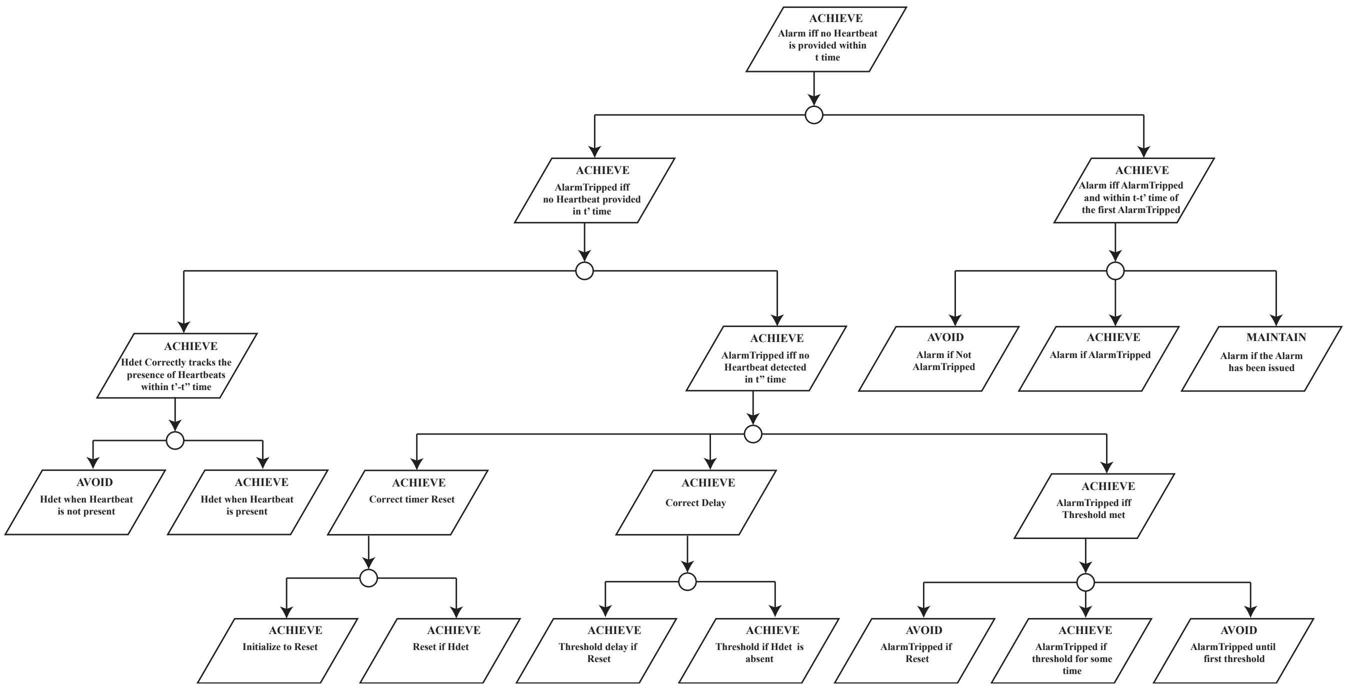
ACHIEVE
Alarm iff no Heartbeat is provided within t time
ACHIEVE
AlarmTripped iff no Heartbeat provided in t' time
ACHIEVE
Alarm iff AlarmTripped and within t-t' time of the first AlarmTripped
ACHIEVE
Hdet Correctly tracks the presence of Heartbeats within t'-t" time
ACHIEVE
AlarmTripped iff no Heartbeat detected in t" time
AVOID
Alarm if Not AlarmTripped
ACHIEVE
Alarm if AlarmTripped
MAINTAIN
Alarm if the Alarm has been issued
AVOID
Hdet when Heartbeat is not present
ACHIEVE
Hdet when Heartbeat is present
ACHIEVE
Correct timer Reset
ACHIEVE
Correct Delay
ACHIEVE
AlarmTripped iff Threshold met
ACHIEVE
Initialize to Reset
ACHIEVE
Reset if Hdet
ACHIEVE
Threshold delay if Reset
ACHIEVE
Threshold if Hdet is absent
AVOID
AlarmTripped if Reset
ACHIEVE
AlarmTripped if threshold for some time
AVOID
AlarmTripped until first threshold

Figure 1: Goal graph refinement.

the number of each species of molecule in the model, and was investigated and confirmed using model checking. Another early environmental assumption, that at initialization there is no heartbeat in the system, was later removed as the MWT design evolved to become more robust. The current goal graph and design handle both the presence and the absence of the heartbeat as the MWT begins execution.

2.2 Tool-assisted obstacle analysis

To analyze the risks of not being able to achieve the intended system, we used obstacle analysis [49]. This technique challenges the goal graph by identifying candidate obstacles to the satisfaction of the goals, assessing the likelihood and criticality of those obstacles, and controlling or resolving them, e.g., by adding or modifying goals. We found in previous work on a product family of DNA nanopliers that the use of a goal oriented requirements engineering approach, with a focus on early analysis of the risks to satisfying the goals [50], worked well in helping find and remedy missing and unrealistic requirements [33, 34].

With the obstacle analysis we sought to determine whether the goals are robust and realizable in a CRN. In fact, we will see below that the automated formal analysis performed to assess the likelihood and impact of candidate obstacles showed that our initial requirements for the MWT were both inaccurate and incomplete. The next section gives examples of how SimBiology and PRISM helped identify goals that were missing and goals that were not realizable by the agents to which they were initially assigned.

The gaps in the requirements arose largely from the nondeterministic nature of CRN reactions, from the very large number of molecular agents operating in parallel, and from the uncertainties of the probabilistic environment in which the MWT operates. Other, related solutions to specifying goals in uncertain environments formalize the required degree of goal satisfaction, as in [29], or the required probability of goal satisfaction, as in [7]. The MWT differs in that all possible failures inevitably will occur in some proportion of the individual agents, but the system must be robust enough to keep the occurrence of failures acceptably low and to operate correctly with probability approaching 1, even in the presence of some component faults. The watchdog timer that we design must be one of which we can say with confidence that if the application it is monitoring fails, that the MWT will detect and notify us, and that if the MWT notifies us that the application has failed, that we can trust it.

2.3 Formal specification of goals in CSL

The goals in the goal graph are formally specified in *continuous stochastic logic* (CSL) [5, 6]. CSL is expressive enough to reason about the continuous time Markov chains (CTMCs) that form the semantics of our chemical reaction networks. As an example, the goal *Achieve [Reset if Hdet]* is specified in CSL as

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

where *Reset* is a boolean that is true when the absence detector's timer is reset, and H_{det} is a boolean that is true when the absence detector has detected that a heartbeat is present. In other words, the goal is satisfied if every time a heartbeat is detected, then with probability $1-\lambda_1$ the timer will reset within $w_{r_{on}}$ time.

The formal specification and proofs were developed incrementally as the goal graph was refined and corrected. We have proven that satisfaction of the conjunctive subgoals satisfies the CSL specification of the top-level goal in the goal diagram. The complete CSL specifications for the goal graph appear in Appendix B.

2.4 Setting the Goal Parameters

Constructing a model satisfying all the constraints on the parameters provided in the goal diagram would have been prohibitively time-consuming without the aid of automation. We developed custom MATLAB scripts that both generated models of arbitrary paramters and allowed smooth integration with the model checkers PRISM [26] and SMART [10]. These scripts along with some features provided by PRISM automated the exploration of the parameter space to discover models that provably satisfy our requirements.

3. DESIGN OF THE MOLECULAR WATCHDOG TIMER

Our molecular watchdog timer consists of three separate components, which are the agents to which leaf goals are assigned. These three components—the absence detector, the threshold filter, and the signal amplifier—are mathematically modeled by stochastic chemical reaction networks (CRNs). CRNs serve as a high-level programming language to specify the behavior for a computational subsystem implemented in DNA. For example, the absence detector component is assigned responsibility for the goal *Achieve [Reset if Hdet].*

3.1 The Absence Detector

This component detects when a heartbeat signal has not been present for a specified period of time. The heartbeat is a "dose" of a specific molecular species "H" that is expected to be periodically output by the molecular application that is being monitored by the MWT. If the heartbeat is not detected by the MWT for an extended period of time, we can conclude that the molecular application being monitored has failed. The absence detectors are assigned responsibility for achieving the leaf goals *Avoid [Hdet when Heartbeat is not present]*, *Achieve [Hdet when Heartbeat is present]*, *Achieve [Initialize to Reset]*, *Achieve [Reset if Hdet]*, *Achieve [Threshold delay if Reset]*, and *Achieve [Threshold if Hdet is absent].*

The CRN model for the absence indicator component is:

$$\begin{aligned} L_i + U &\to L_{i+1} + U \\ L_k + U &\to Y + U \\ L_i + H &\to L_1 \\ Y + H &\to L_1 \end{aligned}$$

The CRN specifies a cascade, or "ladder" such that, at any rung, "L_i", binding of the rung molecule with a "U" causes the cascade to climb a rung, to "L_{i+1}", and binding with a heartbeat molecule "H" knocks the cascade back down to the bottom rung. In the absence of binding of the rung with a heartbeat, the ladder climbs to the next rung. (This is an implementation of the "frog in the well" Markov process [4].) Upon reaching the top rung, the ladder releases a "Y" molecule into solution. The "Y" molecule can then reach and bind to the threshold filter unless it first encounters a heartbeat molecule. In the latter case, the ladder is returned to the bottom rung.

3.2 The Threshold Filter

This component detects when a target number of absence detectors have each reached the "Y" state, releasing a signal that is received by the threshold filter. If and only if enough absence detectors are in a "Y" state, will the threshold filter trip an alarm. The threshold filters are assigned to the leaf goals *Avoid [AlarmTripped if Reset]*, *Achieve [AlarmTripped if threshold for some time]*, and *Avoid [AlarmTripped until first threshold].*

The CRN for the threshold filter component is:

$$\begin{aligned} T_i + Y &\to T_{i+1} + Y \\ T_k + Y &\to T_k + Y + D \\ T_i + R &\to T_1 + R \end{aligned}$$

The threshold filter CRN creates a cascade similar to that of the absence detector. Binding of a "T_i" molecule with a "Y" molecule causes the cascade to climb a rung to "T_{i+1}", and binding with a "R" molecule causes the cascade to fall to the bottom rung. Unlike the absence detectors, all reactions involved with the threshold filters are catalytic, meaning that they do not consume any molecules. When a threshold filter is at its top rung, reacting with "Y" molecules releases "D" molecules, which are consumed by the signal amplifiers to release alarms.

3.3 The Signal Amplifier

This component takes a tripped alarm and increases it to make it more noticeable by another external system. The signal amplifier cannot create an alarm by itself; it requires the threshold filters to trip the alarm. The signal amplifiers are assigned to the leaf goals *Avoid [Alarm if Not AlarmTripped]*, *Achieve [Alarm if AlarmTripped]*, and *Maintain [Alarm if the Alarm has been issued].*

The CRN for the signal amplifier component is:

$$\begin{aligned} D + Ca &\to Al + Ca \\ Al + Da &\to 2Al \end{aligned}$$

The signal amplifier CRN consumes "D" molecules to produce "Al" molecules, representing an alarm. "Al" molecules then interact with "Da" molecules to produce more "Al" molecules. "Da" is initialized to a finite number to place an upper bound on the number of "Al" molecules allowed in the model. Without a bound, the model contains an infinite number of possible states.

4. VERIFICATION OF THE MOLECULAR WATCHDOG TIMER

Many of the intended uses of the MWT system are in situations where the absence of a heartbeat indicates an unsafe state. In such applications it is critical that the MWT be robust, reliable, and correct. These requirements for the MWT drove the software process towards verification of the CRN MWT model. Ensuring that it is highly probable that the MWT will issue an alarm when it should (and not alarm when it should not) is complicated by the concurrency and large number of individual devices that comprise the MWT system, the exceedingly dynamic DNA environment, the fault-proneness of the individual devices, and the very small size of each MWT. Due to the small size and individual signal strength, any signal must be amplified to be externally observable, further complicating debugging and analysis.

Consistent with the assignment of responsibilities for satisfaction of the leaf subgoals in the goal diagram to the three

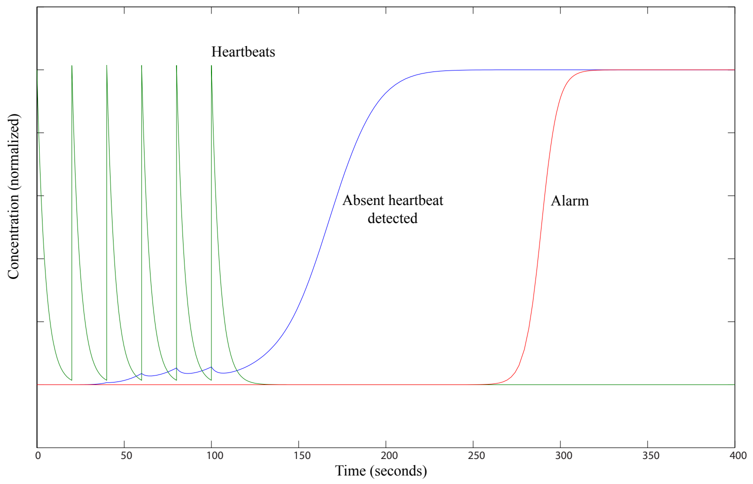

Figure 2: MWT ODE Simulation using MATLAB's SimBiology package

agents–the absence detector, the threshold filter, and the signal amplifier–the system model describes the three agents in the CRN language. The properties to be checked against the CRN model are the CSL properties that are the formal specification of the goals and subgoals.

To build confidence in the correctness and robustness of the planned system, we followed an incremental development process of simulation of the system model for sanity checks (see Figure 2) and selection of likely parameter value ranges, followed by model checking of the goal diagram leaf-node properties for which the agents were responsible. Since we had previously proven that the high-level goals were implied by the lower-level goals, we did not need to verify the higher-level goals. We also injected faults that had been previously discovered by analytical reasoning to confirm that the model checking found them.

We describe below some specific examples of how incorporating model-driven automated analysis into the development process helped identify missing requirements, explore design alternatives, and make the MWT more robust.

Clock Agent Cannot Achieve Assigned Goal: Our first implementation of absence detection for the MWT used a binary counting device introduced in [23]. This design seemed promising, but testing and simulation in MATLAB's SimBiology package revealed that this device fails to meet our specifications.

The problem with this binary counting design for absence detection is that it was designed to work in a setting in which it is assumed that all reactions are "fast" or "slow", and that all "fast" reactions occur before all "slow" reactions. The stochastic (and more realistic) model in which we work here violates this assumption, allowing slow reactions to interfere with the clock's function. Over time the accumulation of such violations inexorably leads to failure of the clock.

To resolve this obstacle, we investigated an alternative refinement in which the delay is achieved by a programmed cascade of interactions. In this case, the goal remained unchanged, and we simply changed the agent used to satisfy the goal.

Missing Domain Property: In refining a goal into two subgoals, we had to introduce the domain property that it takes an expected amount of time after a heartbeat arrives to detect its arrival. This is because the detection occurs via the chemical binding of the heartbeat molecules to the absence detectors. The subgoals did not satisfy the goal without this domain property. This stochastic reality was handled by introducing a "grace period" before the heartbeat detection is required. However, we initially failed to propagate the addition of the grace period back up to the parent goal. Model checking with PRISM was able to detect this omission, and it was corrected in the subsequent version.

Missing Initialization Case: Model checking revealed a failure mode that can occur just after the MWT begins execution. The initial intent was that execution of the MWT begin at time zero, i.e., when the MWT was "poured into the test tube". However, as specified, the MWT could violate this intent by alarming before the external application had a chance to input a heartbeat. To resolve this, we added a new CSL property to the high level goal specifying that alarm must be off for a period of time after initialization. This new goal trickled down through the goal diagram until it created the new leaf-goal *Achieve[Initialize to Reset]*, which specifies that the timer must be considered in a reset stage at initialization. Along with the leaf-goal *Achieve[Threshold delay if Reset]*, this implies that the alarm will not be active upon initialization. We proved manually that the implication holds after the change and used model checking to confirm that it was possible for this failure mode to occur before the change. This was confirmed using model checking by verifying that a model with an alarm in the initial state satisfies the goals.

Goal Too Strict: Automated verification of the goals assigned to the threshold filter revealed that they were insufficient to prove the correctness of their parent. The obstacle derived from the fact the threshold filter was required to prevent the alarm from tripping if the threshold number of absence detectors was not reaching the "Y" state. However, an insufficient threshold could be arbitrarily close to

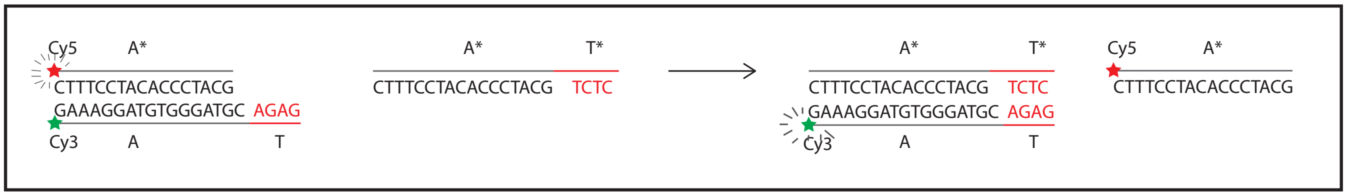

Figure 3: DNA strands used in the simple DNA experiment

a sufficient threshold. Therefore, the threshold filter could not both satisfy the requirement to prevent the alarm from tripping during an insufficient threshold and trip the alarm during a sufficient threshold.

The obstacle was resolved by tightening the constraint on the absence detector to not only enforce that the theshold was not high, but also to enforce that it was low. This enabled the threshold filter to satisfy both requirements and satisfy the parent goal. We verified in PRISM that our MWT satisfies the the new CSL specifications.

Goals Coupled Too Tightly: PRISM verification revealed an error in the specification of a leaf goal assigned to the absence detector that was responsible for maintaining a high threshold in the absence of a heartbeat. This goal enforced that the threshold be maintained until a heartbeat is detected. However, in the implementation of the absence detector, the mechanism for detecting the heartbeat was too tightly coupled to the thresholding process. Therefore, the threshold would not be maintained until a heartbeat was detected.

The obstacle was resolved by relaxing the CSL property, allowing the absence detector to turn the threshold off once a state is reached which has a high probability of detecting a heartbeat within a specified time. PRISM verification confirmed that the change to the property corrected the error.

5. TOOL-ASSISTED DESIGN OF PRELIMINARY EXPERIMENT

In this section we show how the final verified model helped determine specific parameters and molecules for initial laboratory experiments. Specifically, we used the SMART model to determine what length of DNA toeholds to use in the implementation. Here, a toehold is a short length of unpaired nucleotides at the end of a DNA strand. Since the toehold length changes the rate and therefore expected time of a reaction, we wanted to specify a toehold length that allowed enough time for sufficient observation of the experiment, but was short enough to minimize the overall length of the experiment. Simply put, we wanted the experiment to run for the shortest length of time where we could still perform enough observations. We also used SMART to generate an expected time until a threshold percentage of reactions completed, which gave insight into how long the experiment should be run. This tool-supported automatic analysis of the design for the experiment saved laboratory time and cost.

The work described in this section was a simple, preliminary experiment designed to check that we could achieve observable climbing of a single ladder rung. This experiment is in no way novel, but the experimental setup was new to us, which motivated the effort. What is interesting in the context of this paper is that, even with this very simple experiment, the model checking proved useful in correcting two inadvertent errors in our original design.

We defined a single, dynamic strand displacement reaction as a CRN: $AB + C \longrightarrow AC + B$. 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, with strand B now unbound. We created a model of this single reaction in PRISM and in SMART; both model checkers were used to increase confidence in the accuracy of our models. We defined the DNA strands needed to implement the reaction in an experiment but before ordering the strands from the commercial provider, we used the models to gain information about the experiment.

The automated analysis provided two unique benefits. First, the automated analysis of design alternatives helped answer the question of what the DNA strands' toehold length should be for the reaction behavior to be slow enough that it would be observable with the equipment we were using (described below). Analysis with the SMART model checker showed that we needed to decrease the planned length of the DNA toeholds in order for the experiment to be observable. The 6-nucleotide (nt) DNA toeholds that we had initially planned to use would cause the experiment to complete too rapidly for samples to be taken during its progression. The PRISM and SMART models return the same values and agree with the ODE solution [15]. Using these results, we chose a toehold length of 4 nt to implement the reaction.

Second, the automated analysis of design alternatives helped answer the question of what the time-to-imaging should be for good observability. We were unsure when the reactions would complete and, based on our reading of the literature, expected to run the experiment for a few hours. Automated analysis showed that 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. Based on the modeling results, 4-nt toehold strands were ordered and worked well in the experiments. Sampling intervals for the experiment also were chosen based on the information from the model.

Figure 3 shows the DNA strands used in the experiment. Here, Strand 1 is a domain A and a toehold T; Strand 2 is the complementary domain of A; and Strand 3 is the complementary domains of A and T. Cy3 and Cy5 are fluorophores that absorb and emit light at wavelengths of 550/570 nm and 649/670 nm, respectively. Importantly, there is sufficient overlap in the emission spectrum of Cy3 and the absorbance spectrum of Cy5 that proximity-dependent fluorescence resonance energy transfer (FRET) can occur between the two fluorophores. Thus, if the two dyes are separated by less than about 10 nm, irradiation of Cy3 with 550 nm light will

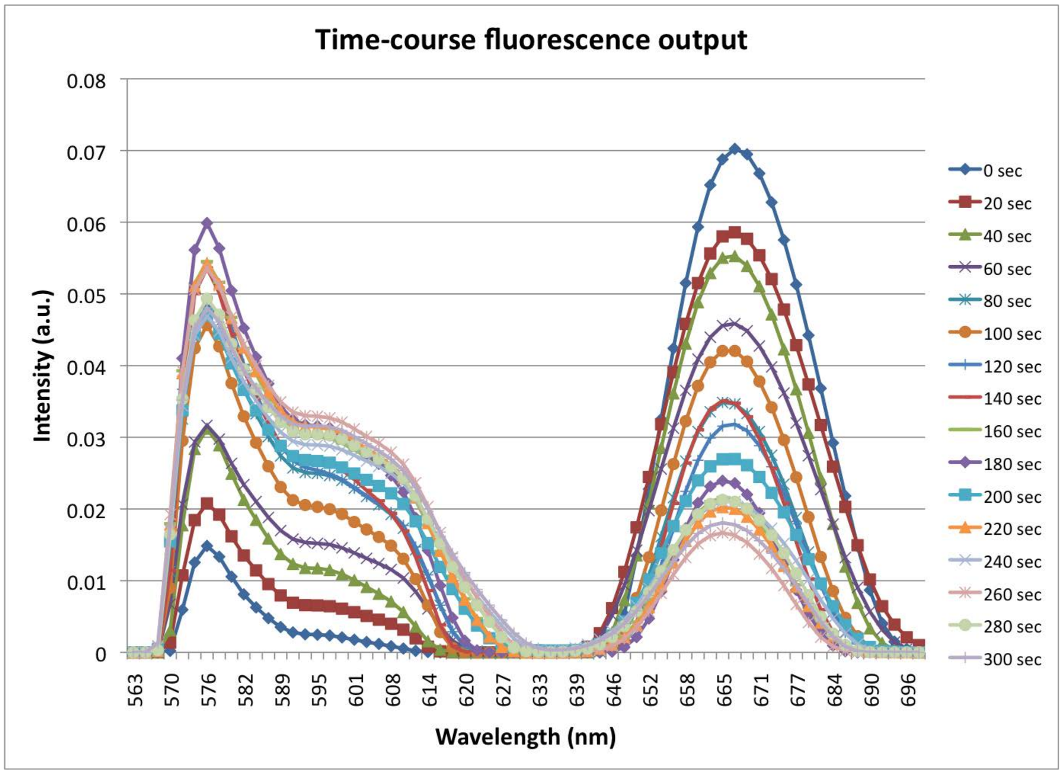

Figure 4: Results of simple, initial experiment.

result in emission of photons from Cy5 ($\tilde{6}70$ nm). A FRET signal between Cy3 and Cy5 is, therefore, a very useful measure of the proximity of DNA strands to which the dyes are chemically coupled. A change in FRET signal was used to determine the rate of displacement of one strand by another in the experiments.

Initial results (see Figure 4) indicate that the reaction progressed as intended. However, we are not yet certain that 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 less light. To avoid photo-bleaching, we created a new sample for each observation and waited the specified amount of time before exposing the sample to light. This eliminates the photo-bleaching, but also introduces the possibility of error due to the multiple samples.

Compared to the initial longer expected time to equilibrium, the model checking indicated that we only needed to run the experiment for a few minutes. The time estimates are based on the length of the toeholds. Through modeling, we were able to determine before experimentation began what was likely to be an appropriate length to enable observability. It is telling that even with an experiment as simple as this, early model checking reduced the overall development effort. We reduced the need to do trial and error experiments in the laboratory to find the right toehold length. This lowered the development cost by saving laboratory time and by reducing the number of different DNA strands that had to be ordered.

6. RELATED WORK

Probabilistic model checking permits the analysis and formal verification of systems that exhibit stochastic behavior. Nanosystems, such as the molecular watchdog timer, are inherently stochastic. Probabilistic model checkers allow the verification of the correctness of the modeled behavior at either the individual device level or at the higher level of the behavior of the assemblage (population) of devices. In our work on the MWT we have used the open-source, probabilistic model checkers PRISM [26] and SMART [10], to analyze the behavioral and performance requirements for the MWT.

PRISM has been used previously to model a variety of biological case studies, including DNA circuits and DNA nanorobotic walkers [11, 27]. PRISM also is used widely to model probabilistic protocols, e.g., in distributed sensor networks, and stochastic multi-player games [26]. PRISM interfaces with Visual DSD, a design tool for DNA strand displacement [28].

SMART has been used to model other types of stochastic systems, e.g., in a NASA-funded study to evaluate an airport runway safety monitor protocol for false alarms [43]. Stochastic Petri net formalisms have also been used to model signal transduction in biological pathways [20]. A good overview of the formalisms used in population models, including the stochastic petri nets used in SMART, appears in [21]. Model checking in support of the preliminary experimental work reported here was performed using SMART. To the best of our knowledge, this was the first time that SMART was used in molecular programming [15]. The rest of the model checking reported in this paper was done using PRISM in order to take advantage of its CSL support.

Our work differs from most of the previous work in probabilistic model checking by focusing more on the identification, specification and analysis of requirements. An exception is work by Filieri, Ghezzi and Tamburrelli to use probabilistic model checking to verify reliability requirements [16]. Non-probabilistic model checkers also have been used for goal-oriented obstacle analysis during the requirements phase. For example, Alrajeh et al. have recently used model checking and machine learning to automatically generate obstacles for the safety-critical London Ambulance Service from model checking counterexamples [1].

Our work also differs from previous computational approaches on molecular systems in that we formally specify and validate the requirements. Goal oriented requirements engineering, including KAOS [49] and TROPOS [3], repeatedly has shown benefits in eliciting, specifying and validating requirements for a wide variety of safety-critical, secure, and high-availability systems. However, despite the difficulty of "getting the requirements right" for a new molecular device, the requirements engineering of molecular programming is rarely considered as a separate activity from design.

More broadly, there has been significant recent progress in modeling biological or chemical systems. Yordanov et al. have formalized and encoded DNA computing to allow use of Satisfiability Modulo Theories (SMT) [53]. Fisher, Harel and Henzinger have done computational modeling of biological systems as reactive systems [17]. Hetherington et al. and Sumner et al. have composed an advanced computational model of a biological system from sub-models describing its different aspects [22, 48].

Our experience developing the molecular watchdog timer is consistent with the "Twin Peaks" idea, described by Nuseibeh [36], that requirements and architecture should coevolve. We used the probabilistic model checker extensively and incrementally to check the alignment of goals with design alternatives. The obstacle analysis gave us a framework for selecting among alternative designs [49]. It was hard to correctly specify the requirements, and the automated analysis of the formal models revealed multiple instances of infeasible goals or inadequate agent assignments, given the realities of the stochastic environment in which the DNA device operates. Often, the physical constraints drove our revision (de-idealization) of the requirements. Whalen et al., report similar experience with large avionic systems, in which the requirements were as likely to be wrong as the models [52]. They found, as we did, that the analysis of formal models was effective in uncovering inconsistencies between the requirements and implicit assumptions about the environment in which the system was to be deployed.

7. DISCUSSION

The tool-supported, incremental, requirements-design process described here was developed to be general enough to be applicable to other molecular programmed systems. The intent is for this to support the development of other new nanosystems by formally specifying an intended system and applying automated tool support to identify unforeseen risks to its achievement early on. Early, computational resolution of missing and unrealizable goals saves time and cost by reducing the number of experiments needed. The process also appears to be applicable beyond molecular programmed systems to other systems in which the system goal is achieved by the interactive behavior of a very large number of distributed, autonomous agents that execute in a probabilistic environment and must be shown to satisfy key properties. Examples of such systems include distributed, adaptive sensor networks and coalitions of systems that display probabilistic, nondeterministic behavior [47].

The framework we used is incremental in that the obstacle analysis, including of the feasibility of the requirements given the agents operating in the probabilistic environment, forces consideration of the physical (here, chemical) limitations, and the associated design constraints. The approach provides a growing understanding of the feasibility or infeasibility of the design alternatives, and this drives the refinement of the requirements to match reality.

8. CONCLUSION

The work reported here shows how programmed molecular systems can benefit from automated requirements analysis and verification. During incremental goal-oriented requirements/design refinements, the use of model-based simulation, probabilistic model checking, and formal analysis identified requirements flaws and latent design constraints and helped us explore better alternatives. The final, verified model of the molecular watchdog timer was then used to help select parameter values for initial laboratory experiments. The verified model provides a baseline for a future family of molecular watchdog timers whose robustness must be assured in their envisioned uses.

9. ACKNOWLEDGMENTS

We thank Samik Basu, Gianfranco Ciardo, Anthony Finkelstein, Carlo Ghezzi, Axel van Lamsweerde, and the reviewers for useful suggestions. This work was supported by NSF Grant 1247051. Part of this work was performed while the fifth author was on sabbatical at Caltech and the Isaac Newton Institute for Mathematical Sciences at the University of Cambridge and the sixth author was on sabbatical at Caltech and the Open University.

10. REFERENCES

[1] D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo, and S. Uchitel. Generating obstacle conditions for requirements completeness. In *ICSE*, pages 705–715, 2012.

[2] D. F. Anderson and T. G. Kurtz. Continuous time Markov chain models for chemical reaction networks. In H. Koeppl, G. Setti, M. di Bernardo, and D. Densmore, editors, *Design and Analysis of Biomolecular Circuits*, pages 3–42. Springer New York, 2011.

[3] Y. Asnar, P. Giorgini, and J. Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requir. Eng.*, 16(2):101–116, 2011.

[4] K. B. Athreya and S. N. Lahiri. *Measure Theory and Probability Theory.* Springer, New York, NY, USA, 2006.

[5] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Model-checking continous-time Markov chains. *ACM Trans. Comput. Log.*, 1(1):162–170, 2000.

[6] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[7] A. Cailliau and A. van Lamsweerde. Assessing requirements-related risks through probabilistic goals and obstacles. *Requir. Eng.*, 18(2):129–146, 2013.

[8] L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Sci. Rep.*, 2, 2012.

[9] Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Soloveichik, and G. Seelig. Programmable chemical controllers made from DNA. *Nat Nano*, 8:755–762, 2013.

[10] G. Ciardo, R. J. *III*, A. S. Miner, and R. Siminiceanu. Logic and stochastic modeling with SMART. *Perform. Eval.*, 63(6):578–608, 2006.

[11] F. Dannenberg, M. Z. Kwiatkowska, C. Thachuk, and A. J. Turberfield. DNA walker circuits: Computational potential, design, and verification. In *DNA*, pages 31–45, 2013.

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

[13] D. Doty. Timing in chemical reaction networks. In *SODA*, pages 772–784, 2014.

[14] S. M. Douglas, I. Bachelet, and G. M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070):831–834, 2012.

[15] S. J. Ellis. Designing a molecular watchdog timer for safety critical systems. Master's thesis, Iowa State University, 2014.

[16] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *ICSE '11*, pages 341–350, 2011.

[17] J. Fisher, D. Harel, and T. A. Henzinger. Biology as reactivity. *Commun. ACM*, 54:72–82, Oct. 2011.

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

[19] D. Han, S. Pal, J. Nangreave, Z. Deng, Y. Liu, and H. Yan. DNA origami with complex curvatures in three-dimensional space. *Science*, 332:342–346, 2011.

[20] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, pages 215–264. Springer Berlin Heidelberg, 2008.

[21] T. A. Henzinger, B. Jobstmann, and V. Wolf. Formalisms for specifying Markovian population models. *Int. J. Found. Comput. Sci.*, 22(4):823–841, 2011.

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

[23] H. Jiang, M. Riedel, and K. Parhi. Synchronous sequential computation with molecular reactions. In *Proceedings of the 48th Design Automation Conference*, pages 836–841. ACM, 2011.

[24] Y. Ke, L. Ong, W. M. Shih, and P. Yin. Three-dimensional structures self-assembled from DNA bricks. *Science*, 338(6111):1177–1183, 2012.

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

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

[27] M. R. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of The Royal Society Interface*, pages 1470–1485, 2012.

[28] M. R. Lakin, S. Youssef, F. Polo, S. Emmott, and A. Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.

[29] E. Letier and A. van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *SIGSOFT FSE*, pages 53–62, 2004.

[30] N. G. Leveson. *Safeware: System Safety and Computers*. ACM, New York, NY, USA, 1995.

[31] X. Liu, Y. Liu, and H. Yan. Functionalized DNA nanostructures for nanomedicine. *Isr. J. Chem*, 53:555–556, 2013.

[32] K. Lund, A. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. Stojanovic, N. Walter, E. Winfree, and H. Yan. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.

[33] R. R. Lutz, J. H. Lutz, J. I. Lathrop, T. Klinge, E. Henderson, D. Mathur, and D. A. Sheasha. Engineering and verifying requirements for programmable self-assembling nanomachines. In *ICSE*, pages 1361–1364, 2012.

[34] R. R. Lutz, J. H. Lutz, J. I. Lathrop, T. Klinge, D. Mathur, D. M. Stull, T. Bergquist, and E. Henderson. Requirements analysis for a product family of DNA nanodevices. In *RE*, pages 211–220, 2012.

[35] D. Mathur and E. R. Henderson. Complex DNA nanostructures from oligonucleotide ensembles. *ACS Synthetic Biology*, 2(4):180–185, 2013.

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

[37] A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 6:S419–S436, 2009.

[38] L. Qian and E. Winfree. A simple DNA gate motif for synthesizing large-scale circuits. *J Royal Soc Interface*, 8:1281–1297, 2011.

[39] L. Qian, E. Winfree, and J. Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 45:368–372, Oct. 2011.

[40] P. W. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12), 2004.

[41] P. W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440:297–302, 2006.

[42] N. Seeman. Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.

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

[44] D. Soloveichik. *Molecules computing : self-assembled nanostructures, molecular automata, and chemical reaction networks*. PhD thesis, California Institute of Technology, 2008.

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

[46] D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proc Natl Acad Sci*, 107(12):5393–5398, 2010.

[47] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. McDermid, and R. Paige. Large-scale complex IT systems. *Commun. ACM*, 55(7):71–77, July 2012.

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

[49] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, Chicester, England, 2009.

[50] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Software Eng.*, 26(10):978–1005, 2000.

[51] B. Wei, M. Dai, and P. Yin. Complex shapes self-assembled from single-stranded DNA tiles. *Nature*, 485(7400):623–626, 2012.

[52] M. W. Whalen, A. Gacek, D. D. Cofer, A. Murugesan, M. P. E. Heimdahl, and S. Rayadurgam. Your "What" is my "How": Iteration and hierarchy in system design. *IEEE Software*, 30(2):54–60, 2013.

[53] B. Yordanov, C. M. Wintersteiger, Y. Hamadi, and H. Kugler. SMT-based analysis of biological computation. In *NASA Formal Methods*, pages 78–92, 2013.

[54] D. Y. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, 3:103–113, 2011.

APPENDIX

A. CHEMICAL REACTION NETWORKS

We very briefly review the CRN abstraction. Let Λ be a finite set, whose elements we call *(molecular) species*. A *state* over Λ is an element $\mathbf{c} \in \mathbb{N}^\Lambda$, i.e., a function $\mathbf{c} : \Lambda \to \mathbb{N}$. Intuitively, for each species $S \in \Lambda$, the value $\mathbf{c}(S)$ is the number of molecules of S that are present in the solution in state $\mathbf{c}$. We often think of a state $\mathbf{c} \in \mathbb{N}^\Lambda$ as a vector whose components are the natural numbers $\mathbf{c}(S)$.

A *reaction* over Λ is an expression α of the form

$$A_1 + A_2 \xrightarrow{k} B_1 + \cdots + B_m, \qquad (*)$$

where A_1, A_2, B_1, …, B_m are (not necessarily distinct) elements of Λ, and k is a positive real number. We call A_1 and A_2 the *reactants* of α, $B_1, \cdots, B_m$ the *products* of α, and k the *rate constant* of α. Note that (*) requires all reactions in this paper to be *bimolecular*, i.e., to have exactly two reactants. We freely use convenient, obvious abbreviations such as writing $2A \xrightarrow{k} B + C$ for $A + A \xrightarrow{k} B + C$.

A reaction α is *applicable* in a state $\mathbf{c}$ if the reactants of α are present in $\mathbf{c}$. In this case, the reaction α may at some time be *applied* to $\mathbf{c}$, resulting in the new state $\mathbf{c}'$ obtained from $\mathbf{c}$ by subtracting the appropriate numbers of reactants and adding the appropriate numbers of products. For example, if there are initially a_1 molecules of A_1, a_2 molecules of A_2, and no molecules of B in solution, then repeated applications of a reaction $A_1 + A_2 \xrightarrow{k} B$ will eventually increase the number of molecules of B to $\min\{a_1, a_2\}$.

A *chemical reaction network* (*CRN*) is an ordered pair $\mathcal{C} = (\Lambda, R)$, where Λ is a finite set of species and R is a finite set of reactions on Λ. The *rate* of a reaction $\alpha \in R$ as in (*) *in* a state $\mathbf{c}$ is $k_{\mathbf{c}}(\alpha) = \frac{kN}{v}$, where v is the volume of the solution, and $N = \textbf{if } A_1 \neq A_2 \textbf{ then } \mathbf{c}(A_1) \cdot \mathbf{c}(A_2) \textbf{ else } \mathbf{c}(A_1) \cdot (\mathbf{c}(A_1) - 1)/2$. (Note that $k_{\mathbf{c}}(\alpha) > 0$ if and only if α is applicable in $\mathbf{c}$.) The CRN $\mathcal{C}$ operates probabilistically as a continuous time Markov process with these rates [18, 2]. This determines, for any state $\mathbf{c}$ the probability distribution on the next reaction and the expected time until this reaction.

The informed reader may notice that the CRN model is closely related to three other models, namely Petri nets, vector addition systems, and population protocols. These connections are useful in many contexts, but they need not concern us here.

B. CSL FORMALIZATIONS

Below are the CSL formalizations of the goal diagram in Figure 1. These goals are ordered in breadth first order following the order of the figure.

Our specification is designed such that domain specialists can specify parameters according to their needs that will then constrain the goal diagram. The parameters the user must provide are:

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

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

ϵ - Probability of error allowed by the u-delay

δ - Probability of error allowed by the v-alarm

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

$$(1 - \epsilon_1)(1 - \epsilon_2) = (1 - \epsilon)$$
$$(1 - \delta_1)(1 - \delta_2) = (1 - \delta)$$

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

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

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

ACHIEVE: AlarmTripped iff no HB provided in t' time

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

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

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

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

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

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

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

ACHIEVE: AlarmTripped iff no Heartbeat detected

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

AVOID: Alarm if not AlarmTripped

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

ACHIEVE: Alarm if AlarmTripped

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

MAINTAIN: Alarm if AlarmTripped

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

AVOID: Heartbeat Detected if Heartbeat not present

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

ACHIEVE: Heartbeat Detected if Heartbeat present

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

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

$$\begin{aligned}
(1-\gamma_1)(1-\gamma_2) &= (1-\epsilon)\\
(1-\gamma_1)(1-\lambda_1)(1-\lambda_2)(1-\lambda_3)(1-\lambda_4) &= (1-\epsilon_1')(1-\epsilon_2')\\
(1-\eta_1)(1-\eta_2)(1-\eta_3) &= (1-\delta_1')\\
w_{r_{off}} &\leq g - w_h
\end{aligned}$$

ACHIEVE: Correct timer reset

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

ACHIEVE: Correct delay

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

ACHIEVE: AlarmTripped iff Threshold met

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

ACHIEVE: Initialize to Reset

$$Reset$$

ACHIEVE: Reset if Heartbeat detected

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

ACHIEVE: Threshold delay if Reset

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

ACHIEVE: Threshold if Heartbeat Detected is absent

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

AVOID: AlarmTripped if reset

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

ACHIEVE: AlarmTripped if threshold for some time

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

AVOID: AlarmTripped until first threshold

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