Stochastic Model Checking of Genetic Circuits

Curtis Madsen, Newcastle University
Zhen Zhang, University of Utah
Nicholas Roehner, University of Utah
Chris Winstead, Utah State University
Chris Myers, University of Utah

Synthetic genetic circuits have a number of exciting potential applications such as cleaning up toxic waste, hunting and killing tumor cells, and producing drugs and bio-fuels more efficiently. When designing and analyzing genetic circuits, researchers are often interested in the probability of observing certain behaviors. Discerning these probabilities typically involves simulating the circuit to produce some time series data and computing statistics over the resulting data. However, for very rare behaviors of complex genetic circuits, it becomes computationally intractable to obtain good results as the number of required simulation runs grows exponentially. It is, therefore, necessary to apply numerical methods to determine these probabilities directly. This paper describes how *stochastic model checking*, a method for determining the likelihood that certain events occur in a system, can by applied to models of genetic circuits by translating them into *continuous-time Markov chains* (CTMCs) and analyzing them using *Markov chain analysis* to check *continuous stochastic logic* (CSL) properties. The utility of this approach is demonstrated with several case studies illustrating how this method can be used to perform design space exploration of two genetic oscillators and two genetic state-holding elements. Our results show that this method results in a substantial speedup as compared with conventional simulation-based approaches.

Categories and Subject Descriptors: G.3 [**Mathematics of Computing**]: Probability and Statistics—*Markov processes, stochastic processes*; J.3 [**Computer Applications**]: Life and Medical Sciences—*biology and genetics*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*computer-aided design (CAD)*

General Terms: Design, Verification

Additional Key Words and Phrases: Stochastic model checking, design space exploration, synthetic genetic circuits, Markov chain analysis, continuous stochastic logic

ACM Reference Format:
Madsen, C., Zhang, Z., Roehner, N., Winstead, C., and Myers, C. 2014. Stochastic Model Checking of Genetic Circuits. ACM X, X, Article 1 (XXXX), 20 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Recently, biologists and engineers have begun to work together on *synthetic biology* [Arkin 2008; Endy 2005]. Indeed, the promise of synthetic biology is to design new, useful biological systems to

---

An earlier version of this paper entitled “Utilizing Stochastic Model Checking to Analyze Genetic Circuits" was presented at the 2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB) and was published in its Proceedings (http://dx.doi.org/10.1109/CIBCB.2012.6217255). This paper expands on the previous paper by allowing support for nested continuous stochastic logic properties in the stochastic model checking algorithm and by including a discussion on the intuition of how to select thresholds when performing the logical abstraction of a genetic circuit to a continuous-time Markov chain. Additionally, this paper presents new results of applying the stochastic model checking methodology to two genetic oscillator circuits, the repressilator and the dual-feedback genetic oscillator.
This work is supported by the National Science Foundation under Grants CCF-0916105 and CCF-0916042.
Author’s addresses: C. Madsen, School of Computing Science, Newcastle University; Z. Zhang and C. Myers, Department of Electrical and Computer Engineering, University of Utah; N. Roehner, Department of Bioengineering, University of Utah; C. Winstead, Department of Electrical and Computer Engineering, Utah State University.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

perform tasks such as consuming toxic waste [Cases and de Lorenzo 2005], destroying tumors [Anderson et al. 2006], and producing drugs [Ro et al. 2006] and bio-fuels [Atsumi and Liao 2008]. However, the design of *synthetic genetic circuits* (i.e., the collections of genes within DNA that interact to control the behavior of cells) is currently largely an ad hoc process of trial and error that can be both time consuming and costly. One should first develop models and analyze them *in silico* (on a computer) before testing the object of the model *in vivo* (within an organism). Indeed, this problem has already been tackled in electronic circuit design where methods for modeling and analyzing new designs of circuits have been developed to ensure that a circuit behaves as desired before it is fabricated. Crucial to making this methodology a reality for genetic circuits is the development of *genetic design automation* (GDA) tools. While some GDA tools are already under development [Beal and Bachrach 2008; Bilitchenko et al. 2011; Cai et al. 2010; Chandran et al. 2009; Marchisio and Stelling 2008; 2011; Myers et al. 2009; Pedersen and Phillips 2009; Yordanov et al. 2010], there is still a need for more efficient methods for the modeling, analysis, and design of genetic circuits. Therefore, the goal of this paper is to develop an analysis methodology that can greatly accelerate the exploration of the design space during genetic circuit design.

Biological systems can often be represented as a set of *chemical species* (different types of molecules) that are acted upon by *chemical reactions*. Such a chemical reaction network representation can be readily translated into an *ordinary differential equation* (ODE) model using the *law of mass action* [Waage and Guldberg 1864]. These models can then be simulated and analyzed using a variety of well known methods to obtain results on the behavior of the system [Press et al. 1992; Strogatz 1994]. When the numbers of each chemical species is large, which is the case in many biological systems, ODE models produce reasonably accurate results. Unfortunately, genetic circuits typically involve very small molecule counts leading to inaccurate ODE analysis results [McAdams and Arkin 1999]. Indeed, the behavior of genetic circuits is highly non-deterministic.

In order to more accurately reason about genetic circuits, it is essential that their inherent stochastic nature be carefully considered. To address this concern, stochastic analysis methods such as *Gillespie's stochastic simulation algorithm* (SSA) [Gillespie 1977] are typically applied to the simulation of genetic circuits. The SSA is a Monte Carlo simulation algorithm that deals effectively with the small, discrete amounts of species and the randomness of reactions while improving efficiency of simulation by stepping over useless time steps (time steps where no reactions occur). Over the years, the efficiency of the SSA has been improved in a variety of ways [Gibson and Bruck 2000; Cao et al. 2006; Gillespie and Petzold 2003; Kuwahara et al. 2006; Kuwahara 2007; Kuwahara and Mura 2008; Slepoy et al. 2008].

Unfortunately, stochastic simulation of genetic circuits can be extremely time consuming even with the various improvements. To address the simulation time problem in electronic circuits, one typically applies a logical abstraction in which rather than tracking the exact voltage of each signal wire, it is only considered whether the wire is at a high or low voltage level. Similarly, one can apply a logical abstraction to a genetic circuit. When properly applied, the resulting qualitative logical model only tracks whether each species is in a high or a low state rather than tracking the exact amount of each species in a system [Thieffry and Thomas 1995; Thomas 1991]. One problem with this type of abstraction is that it yields a model that is incapable of producing quantitative predictions of behavior. To address this problem, a quantitative logical model can be used to encode the infinite state space of a genetic circuit into a *continuous-time Markov chain* (CTMC) [Kuwahara et al. 2006; Kuwahara 2007; Hahn et al. 2009; Ciocchetta et al. 2009; Burrage et al. 2006; Munsky and Khammash 2006; Henzinger et al. 2009].

In order to perform computations over the infinite state space of a CTMC, techniques have been developed that effectively transform the Markov chain into a discrete-time finite-state model. Many methods utilize a *state projection* to explore the chain up to a specified depth as they try to approximate the system [Hahn et al. 2009; Ciocchetta et al. 2009]. Others perform this exploration but are capable of dynamically exploring more of the chain if the explored states do not satisfy a desired level of precision [Burrage et al. 2006; Munsky and Khammash 2006]. Finally, some methods attempt to explore the infinite chain in sections and use a sliding window to move to different sec-

tions of the chain as necessary [Henzinger et al. 2009]. The state projection abstraction works very well for smaller systems (i.e., systems of two or three species). However, since they try to maintain as much of the full CTMC as possible, state projection methods often do not scale well for more complex systems of several species with greater ranges of possible values.

After a circuit has been abstracted into the logical representation of a computationally tractable Markov chain, it can be efficiently analyzed using *stochastic model checking* [Kwiatkowska et al. 2007] to quickly compare the robustness of alternative circuit designs and evaluate different design trade-offs. Stochastic model checking is a technique that utilizes either statistical (simulation) or numerical (*Markov chain analysis*) techniques to determine the probability that a system satisfies a specified property [Younes et al. 2006]. State projection methods lend themselves well to stochastic model checking where the probability of a *transient* property can be easily checked using either statistical or numerical approaches. However, in order to also compute how much error they introduce, state projection methods include absorbing states that are used to collect error probability during analysis. By including these states, they are incapable of being used for computations regarding the *steady-state* behavior of a system due to steady-state computations moving the probability density to recurrent parts of the CTMC such as the absorbing error states.

This paper presents a new methodology for the design space exploration of genetic circuits that leverages logical abstraction and stochastic model checking techniques. In particular, this methodology translates a representation of the genetic circuit into a quantitative logical model encoded by a CTMC. Unlike previous methods, the presented approach uses more abstractions to collapse states of the CTMC together. By performing additional abstractions, the reduced state space can be analyzed more efficiently allowing the method to scale to larger and more complex systems. Although these abstractions introduce some unquantifiable error into the CTMC, the presented method can be used in conjunction with stochastic model checking to analyze not only transient properties but also steady-state properties of a genetic circuit design specified using *continuous stochastic logic* (CSL). Finally, this paper illustrates utilizing this methodology to perform design space exploration of several circuits using genetic technology including two genetic oscillators and two state-holding elements. In particular, it highlights how the efficiency of this method allows it to be used as an engineering tool to enable design trade-offs to be quickly evaluated in order to select robust designs for the analyzed circuits.

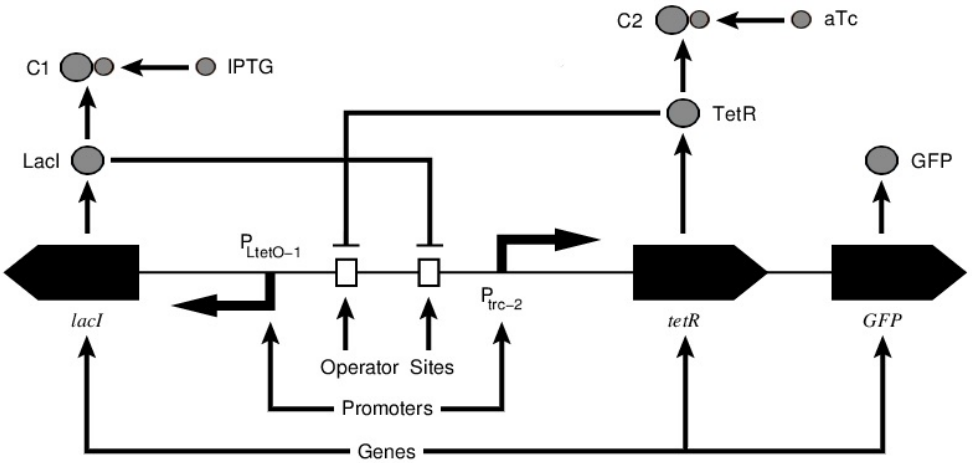
Section 2 presents background on genetic circuits. Section 3 describes our methodology to produce and analyze a logical representation of a genetic circuit. Section 4 presents the case studies. Finally, Section 5 discusses future research that is needed to support the design of genetic circuits.

2. BACKGROUND

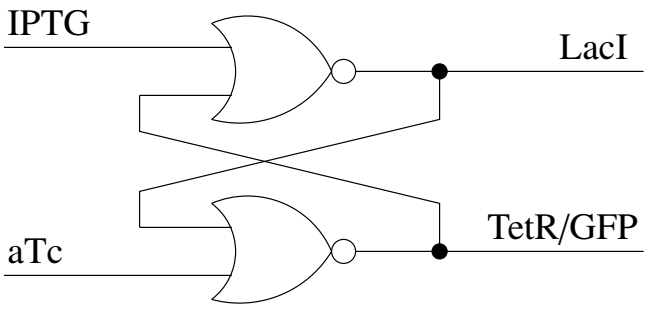
A genetic circuit is composed of, among other things, *genes*, *operator sites*, and *promoters* on a strand of DNA. An example of a genetic circuit for a toggle switch which was constructed by Gardner et al. [Gardner et al. 2000] is shown in Figure 1(a). This diagram was constructed using symbols from the the SBOL Visual standard [Quinn et al. 2013]. Genes are the portion of the DNA that code for *proteins*, the basic building blocks for nearly all molecular machinery within a cell. Proteins can also regulate cellular function by binding to an operator site in order to increase, *activate*, or decrease, *repress*, the associated promoter's affinity. Promoters are the regions on DNA where *RNA polymerase* (RNAP) binds to start the transcription of a gene to produce *messenger RNA* (mRNA), which in turn is translated by a *ribosome* into a protein. In Figure 1(a), the LacI protein binds to the operator site associated with the P_{trc-2} promoter to repress the production of TetR and *green fluorescent protein* (GFP), a reporter that causes the cell to glow. Similarly, the TetR protein binds to the operator site associated with the $P_{LtetO-1}$ promoter to repress the production of LacI. The molecules IPTG and aTc are known as *chemical inducers*. IPTG can bind with LacI to form a complex C1 that prevents LacI from being able to repress TetR production. Similarly, aTc can bind with TetR to form a complex C2 that prevents TetR from being able to repress LacI production.

A logical model that summarizes the behavior of the genetic toggle switch is shown in Figure 1(b). Digital designers should recognize this circuit as a *set-reset* (SR) *latch*, a simple asynchronous

state-holding gate. The chemical inducer IPTG is the *set* input that is used to set the output, GFP, in the high state. The chemical inducer aTc is the *reset* input that is used to set the output in the low state. When neither input is applied, the latch retains its previous state. Applying both inputs simultaneously, just like in an electronic SR latch, is illegal as it would make the circuit oscillate erratically. While this diagram indicates a logical behavior, one should keep in mind that the actual behavior of this latch is extremely noisy due to the inherent stochastic nature of genetic circuits described earlier.



(a)



(b)

Fig. 1: (a) The genetic toggle switch circuit drawn using symbols from the SBOL Visual standard [Quinn et al. 2013] where LacI and TetR repress each other (denoted by the ⊣ arrows). In this circuit, LacI can be sequestered by IPTG, TetR can be sequestered by aTc, and GFP is the reporter protein causing the cell to glow indicating whether the toggle is in the on or off state. While this figure shows the operator site and promoter as distinct regions of the DNA, these sequences typically overlap. This figure was partly constructed using Pigeon [Bhatia and Densmore 2013], an SBOL Visual-compliant tool. (b) A digital circuit representation of the genetic toggle switch.

Figure 2 presents the average of performing 100 stochastic simulation runs for 25,000 seconds of the genetic toggle switch using Gillespie's SSA. These simulations are started in an initial state in which LacI is 60 molecules while TetR, GFP, IPTG, and aTc are all 0 molecules. At time 5,000, 60 molecules of IPTG are provided which causes the state of the toggle switch to change resulting in the production of GFP. At time 10,000, the IPTG is removed, but the toggle switch on average

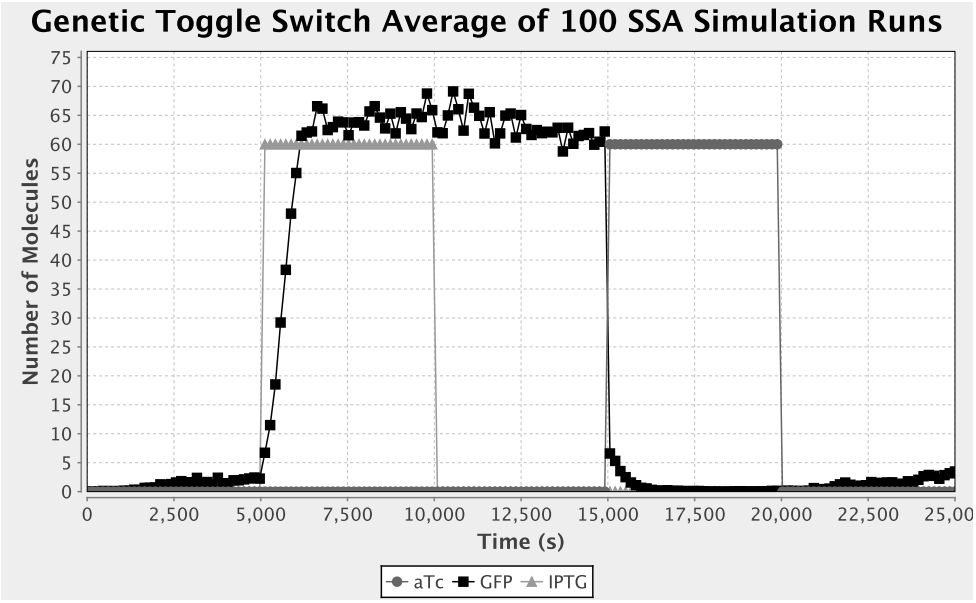


Fig. 2: The average of 100 stochastic simulation runs of the genetic toggle switch. These results plot the molecule count of the GFP protein showing how the circuit is set and reset by IPTG and aTc, respectively. IPTG is added at time 5,000s to set the switch and then is removed at time 10,000s. aTc is later added at time 15,000s to reset the switch and then is removed at time 20,000s.

holds the high GFP state. At time 15,000, 60 molecules of aTc are provided which causes the toggle switch to change state again, and GFP degrades away. At time 20,000, the aTc is removed, and once again the toggle switch holds its state. It should be stressed though that this shows the average of 100 simulations runs. This circuit is noisy meaning that there is a chance that the circuit does not perform ideally and loses its state. One of the goals of this paper is to be able to efficiently determine the probability of this erroneous behavior.

3. METHODS

This section describes how stochastic model checking techniques can be applied to reason about synthetic genetic circuits. In particular, it presents a Markov chain analysis method for checking CSL properties of genetic circuits and a process to logically abstract a genetic circuit into a CTMC.

3.1. Stochastic Model Checking

Model checking has been successfully applied to reason about the correctness of various types of systems in many domains. Due to the inherently noisy nature of genetic circuits, any property that is checked would most likely fail to be true. Instead, for these systems, the more interesting question is the probability that a property is true. To address this problem, researchers have developed stochastic model checking techniques to compute the probability of being in each state at any given time.

There are two types of stochastic model checking used to compute the likelihood that a property is true: statistical and numerical based techniques [Kwiatkowska et al. 2007]. Both of these techniques have been utilized by many tools such as the probabilistic model checker PRISM [Hinton et al. 2006]. Statistical techniques involve simulating a system using a method such as Gillespie's SSA for a large number of runs and terminating whenever a property is shown to be true or false. When all of the simulations are complete, statistics are calculated on how many simulations satisfied the property in the time allotted versus the number of simulations that failed to do so. One downside of

using statistical techniques is that the more rare an event is, the more simulations that need to be run in order to observe it, and this may cause the time that it takes to compute a likelihood to become prohibitively expensive.

Numerical methods, on the other hand, attempt to determine these likelihoods in a more direct method. They usually attempt to find the state space of the model, abstract this state space as a CTMC, and then employ methods such as Markov chain analysis to compute the probability of reaching the states in the system at different times. Numerical methods are often more precise and efficient than statistical techniques; however, they require that the state space be computable and that the appropriate abstractions exist to consider the state space as a CTMC. Since models of genetic circuits typically have infinite state spaces, researchers have developed methods to abstract the state spaces into finite representations. For instance, one simple approach is to set a maximum and a minimum amount for each species in a circuit and only consider states in the CTMC where the species' amounts are within these bounds. Any probability density that leaves this bounded region can then be quantified as error [Hahn et al. 2009; Ciocchetta et al. 2009]. Other methods, such as the *finite state projection* (FSP) [Munsky and Khammash 2006], are capable of allowing a user to specify how much error he or she is willing to accept and iteratively expand the size of the bounds on the CTMC in order to obtain a more precise result. For systems where the majority of the probability density may shift to another portion of the state space, this bounding can be altered dynamically as in the sliding window abstraction [Henzinger et al. 2009].

Each type of stochastic model checking can be utilized to check a CSL property [Aziz et al. 2000; Kwiatkowska et al. 2007] which can be specified using the following grammar:

$$\begin{aligned} Prop &::= \mathtt{U}(\mathrm{T},\Psi,\Psi) \mid \mathtt{F}(\mathrm{T},\Psi) \mid \mathtt{G}(\mathrm{T},\Psi) \mid \mathtt{St}(\Psi) \\ \Psi &::= \mathtt{true} \mid \Psi \wedge \Psi \mid \neg\Psi \mid \phi \geq \phi \mid \phi > \phi \mid \phi = \phi \\ \phi &::= v_i \mid c_i \mid \phi + \phi \mid \phi - \phi \mid \phi * \phi \mid \phi / \phi \mid Prop \\ \mathrm{T} &::= \mathtt{true} \mid \mathrm{T} \wedge \mathrm{T} \mid \neg\mathrm{T} \mid t \geq c_i \mid t > c_i \mid t = c_i \end{aligned}$$

where v_i is a variable, c_i is a constant, and t stands for time in the system. In this grammar, Ψ represents a state formula that must be true in a given state and can be composed of other state formulae combined using logical connectives or comparisons between numerical expressions, ϕ. The formula $\mathtt{U}(T,\Psi_1,\Psi_2)$ represents the probability that an execution of the system satisfies the until formula $\Psi_1 \,\mathtt{U}^{\mathrm{T}}\, \Psi_2$ which means that Ψ_1 must remain true until Ψ_2 becomes true within the time frame that the time bound expression, T, evaluates to true. The eventually operator, F, is essentially used as a shorthand for describing an until property where the left-hand side of the formula is true. For example, the eventually formula $\mathtt{F}(\mathrm{T},\Psi)$ would simply require that Ψ becomes true before T evaluates to false. The globally true formula $\mathtt{G}(\mathrm{T},\Psi)$ requires that Ψ remains true during the time that T evaluates to true. This formula builds off of the eventually operator by requiring that $\neg\Psi$ does not eventually become true while T evaluates to true and returns one minus the resulting probability. The formula $\mathtt{St}(\Psi)$ represents the probability that once the system reaches its steady state, it is in a state where Ψ is satisfied. It should be noted that $Prop$ is a symbol in ϕ's grammar which allows for CSL properties to be nested within other CSL properties. As a shorthand, Ψ and T can also contain `false`, $\vee$, $<$, and $\leq$, which are easily derived.

Given an error bound, ϵ, a CSL property, Φ, and a genetic circuit model, M, Algorithm 1 can be used to determine the probability of an interesting event occurring. The first step of the algorithm converts the model into a CTMC, C, using the level set, L (line 1). Details of this conversion are given below. Next, the algorithm parses the CSL property and walks its expression tree looking for any nodes that represent nested properties (line 2). If any nested properties are found, the algorithm loops over each state in C (line 3) and alters M by setting its initial state to the Markov chain state, s (line 4). The algorithm then recursively calls itself for each altered model, M', using the nested property, Φ', as the new CSL property to be checked (line 5). The probability of each recursive call, p, is stored as a variable of each s where it can be referenced when determining if the state satisfies Φ (line 6). Once all of the nested properties are dealt with, the algorithm determines the amount of

time necessary for the analysis, t, which is essentially the maximum value that time can take while still allowing for the time bound expression to evaluate to true in the transient property or infinity, ∞, in the case of a steady-state property (line 7). Finally, the algorithm checks whether transient or steady-state analysis should be performed and calls the appropriate analysis method (lines 8-11). When checking transient properties, the model checker determines if the encoding either does not satisfy the left hand side of an until formula or satisfies the right hand side of the formula since all transient properties can be written as until formulae. If either of these checks hold, the state is marked as absorbing and all transitions out of the state are pruned from the CTMC before analysis. The transient analysis method utilizes the well known method of *uniformization* while the steady-state analysis method uses the *power iteration method* [Stewart 1994].

ALGORITHM 1: SMC(Model M; Levels L; CSL property Φ; Error-bound ϵ)

```
1  Set C = computeCTMC(M, L, Φ)
2  foreach Nested property Φ' ∈ Φ do
3      foreach State s ∈ C do
4          Set M' = M.setInitialState(s)
5          Set p = SMC(M', L, Φ', ε)
6          s.addVariable(Φ', p)
7  Set t = determineTimeLimit(Φ)
8  if t ≠ ∞ then
9      return transientAnalysis(C, t, Φ, ε)
10 else
11     return steadyStateAnalysis(C, Φ, ε)
```

3.2. Logical Abstraction

The critical step in preparing a genetic circuit for stochastic model checking is the conversion of the circuit into a CTMC. This conversion begins by finding the state space of the genetic circuit; however, since this state space is likely infinite, it must be logically abstracted into a finite state space. As stated earlier, there are many ways of abstracting the infinite state space into a finite one. Each method first constructs a sparse matrix where each entry, $p_{i,j}$, represents the rate of moving from state i to state j. Next, a state is created with an encoding of the initial values of the species in the model. The conversion then performs a depth first search by changing one species amount encoding at a time to a higher or lower encoding. For previously developed methods, this process increments or decrements the amount of each species by one. The method presented in this paper, on the other hand, abstracts the logical representation further by utilizing a predetermined level set L, an ordered list of threshold levels, L_s, for each species $s \in S$ in the model to collapse states together. Each level, $l_{s,i}$ represents a critical threshold in the amount of the species s, and it is assumed that $l_{s,0}$ is always 0, and $l_{s,i-1} < l_{s,i}$ for all $i > 0$. Each valid change found by changing one species encoding is pushed onto a stack. The algorithm then pops an encoding off the stack and checks to see if a transition rate for moving from the current state to the new state exists in the matrix. If it does, the algorithm stops exploring this path and pops the next change off the stack to explore further. Otherwise, the transition rate is calculated and is added to the matrix. Figure 3 shows a graphical representation of the state space for the genetic toggle switch with thresholds selected at 0, 30, and 60 for both LacI and TetR yielding nine states labeled S0 through S8. State S0, the state where LacI is at its highest level of 60 and TetR is at its lowest level of 0, is the initial state.

3.3. Transition Rate Computation

The next step is to compute the transition rates between states. Typically, the reaction rate equations in a model of a genetic circuit cannot be directly used as the transition rates because it usually

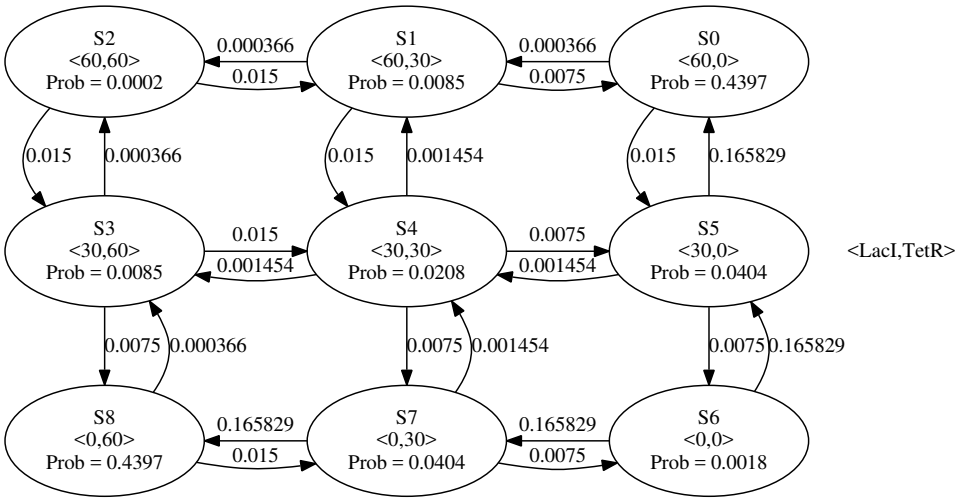


Fig. 3: The CTMC annotated with probabilities after applying steady-state Markov chain analysis.

takes a collection of reactions to represent a change in the level of one of the interesting species in the system. To solve this problem, reaction-based abstractions [Kuwahara et al. 2006; Kuwahara 2007] can be used to construct the transition rates. Although these abstractions introduce some unquantifiable error to numerical stochastic model checking methods, the savings in computational efficiency greatly outweigh the small loss in accuracy. Equations 1 and 2 present the abstracted rate for moving the value of a species to a higher or lower value, respectively:

$$\text{production}(s, L) = \frac{\sum_{p \in Pro(s)} n_p \cdot \text{rate}(p)}{(l_{s,i+1} - l_{s,i})} \quad (1)$$

$$\text{degradation}(s, L) = \frac{k_d l_{s,i}}{(l_{s,i} - l_{s,i-1})} \quad (2)$$

In these equations, s is the species value that is changed by the transition rate equation, L is the set of level sets for each species, and Pro(s) returns the set of promoters that initiate transcription of genes that lead to the production of species s. When the state transition increases the level of species s from $l_{s,i}$ to $l_{s,i+1}$, then Equation 1 is used, and when the state transition decreases the level of s from $l_{s,i}$ to $l_{s,i-1}$, then Equation 2 is used. Equation 1 is computed by determining the rate of production for each promoter p which produces species s using the rate(p) function defined in Equation 3. To obtain a numerical value from this equation, the species variables in the rate(p) equation are replaced by indexing into L to get their current values. This rate is then multiplied by n_p, the number of proteins produced per transcript, to convert this rate into the rate for a single protein production. Equation 2 is computed utilizing an amplified degradation rate approximation where k_d, the degradation rate parameter, is multiplied by $l_{s,i}$, the starting level for species s before degradation. In both cases, these rates must be normalized by the difference in the level before and after the state change. For methods like the FSP, this normalization is not necessary as the rates produced are for the production or degradation of a single molecule of s. However, for the presented method, the state change produces $l_{s,i+1} - l_{s,i}$ molecules or degrades $l_{s,i} - l_{s,i-1}$ molecules which may be more than one. Indeed, this method is actually a generalization of previously published methods like the FSP and will reduce to the FSP if thresholds are selected at every integer value in between the lower and

upper bounds for each species. This method, however, can be used to collapse states together to gain some computational efficiency at the cost of introducing some additional unquantifiable error.

The function rate(p), shown in Equation 3, utilizes an operator site reduction approximation [Tyson and Othmer 1978; Kuwahara et al. 2006; Kuwahara 2007] to return the rate of production initiated from promoter p:

$$\text{rate}(p) = \begin{cases} \dfrac{n_p k_o p K_o RNAP}{1 + K_o RNAP + \sum_{s_r \in \text{Rep}(p)} (K_r v_{s_r})^{n_c}} & \text{if } |\text{Act}(p)| = 0 \\ \dfrac{n_p k_b p K_o RNAP + \sum_{s_a \in \text{Act}(p)} n_p k_a p K_{oa} RNAP (K_a v_{s_a})^{n_c}}{1 + K_o RNAP + \sum_{s_r \in \text{Rep}(p)} (K_r v_{s_r})^{n_c} + \sum_{s_a \in \text{Act}(p)} K_{oa} RNAP (K_a v_{s_a})^{n_c}} & \text{otherwise} \end{cases} \quad (3)$$

In this equation, n_p is the number of proteins produced per transcript, k_o is the open-complex production rate, k_b is the *basal* production rate, k_a is the activated production rate, K_o is the RNAP binding equilibrium constant, K_{oa} is the activated RNAP binding equilibrium constant, K_r is the repressor binding equilibrium constant, K_a is the activator binding equilibrium constant, and n_c is the degree of cooperativity. Note that Act(p) returns the set of activating species, s_a, for promoter p, and Rep(p) returns the set of repressing species, s_r, for promoter p. The derivation of this function is a bit involved, so this paper just presents an informal overview of the process. The variables, v_{s_r} and v_{s_a}, represent the value of the repressing species and activating species for this promoter, respectively. This function breaks this down into two cases. The first case is for a promoter that does not have any species which are activating it. In this case, it is assumed that the promoter is *constitutive* which simply means that it can initiate transcription at a significant rate without the aid of another activating species. Assuming that there are no repressor molecules present, this rate is approximately $n_p k_o p$. However, this rate is reduced as the number of repressor molecules, v_{s_r}, increases. The second case is for a promoter that must be activated for significant transcription. Assuming that there are no activator or repressor molecules present, the rate of production of this promoter is $n_p k_b p$ where k_b is typically much smaller than k_o. In this case, as the number of activator molecules, v_{s_a}, increases so does the rate of production from this promoter. Like the first case, this production can also potentially be inhibited, if there exists species which can repress this promoter. Lastly, if there are complex formation reactions between repressing species and chemical inducers such as that between LacI and IPTG in Figure 1(c), our method applies a complex formation abstraction that uses both the quasi-steady-state approximation and the law of mass conservation. This abstraction replaces the variable v_{s_r} in the rate function with the expression $\frac{v_{s_{r_{total}}}}{1+K_c v_i}$, where $v_{s_{r_{total}}}$ is the variable for the total repressing species (free and in complex), K_c is the complex formation equilibrium constant, and v_i is the variable for the chemical inducer. Consequently, as the amount of chemical inducer increases, the effective amount of repressing species decreases and production from the promoter increases. The conversion process coupled with the corresponding rate functions have been carefully constructed such that the CTMC generated gives a reasonable approximation of the behavior of the genetic circuit. This translation procedure allows the user to efficiently trade-off between accuracy and analysis time by adjusting the number of thresholds. For more details about derivation of application of the quasi-steady-state approximation, please see, for example, [Madsen 2013] and [Myers 2009]. The use of the quasi-steady assumptions with stochastic simulation is justified in [Rao and Arkin 2003].

3.4. Threshold Selection

The presented method relies on a good state space partition when logically abstracting a genetic circuit in order to yield a good approximation of the entire state space. If more levels are used in the level set for each species, the state space is larger leading to longer analysis times. Although, not as

many levels would conversely be able to be analyzed in less time, the results obtained may not be accurate enough to predict the true behavior of the system.

One method of accomplishing threshold selection is to perform a few simulations of the circuit and then to analyze the resulting traces to figure out the range of values for each species. A user could then determine a number of thresholds to be selected uniformly from these ranges. By selecting thresholds uniformly from a range, this method may end up not using enough thresholds around values of a species that cause some of the calculated transition rates to change dramatically and may use too many thresholds for values of a species that do not affect these rates significantly.

Instead of selecting thresholds uniformly, another method is to attempt to select thresholds for each species at points where the rates of the reactions in the system change drastically. This method would likely involve graphing the rates of each reaction and searching for the inflection points of the resulting curves. This approach could then automatically select more thresholds for the species that affect these rates at the values around the inflection points and could select fewer thresholds for these species at values that are further away from the inflection points.

An example of applying steady-state analysis to the genetic toggle switch is presented in Figure 3. In this example, the user is interested in the probability of the system being in a state where LacI is 0 in the long run which is represented by the CSL property, $\mathtt{St}(\mathrm{LacI} = 0)$. Summing over the states that satisfy this property, states $S6$, $S7$, and $S8$, results in a probability of about 48.2 percent. Figure 4 shows the CTMC in Figure 3 annotated with probabilities after applying transient Markov chain analysis to determine the probability of the CSL property $\mathtt{F}(t \leq 100, \mathrm{LacI} = 0)$. Note that states S6, S7, and S8 are marked as absorbing states, since they satisfy the property. The sum of the probability of reaching these states represents the probability of satisfying the property, which is about 5.9 percent.

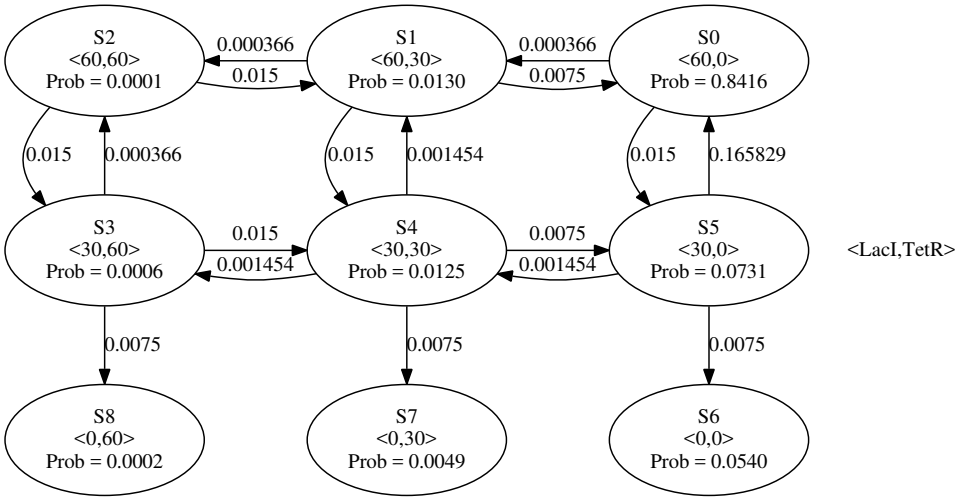


Fig. 4: The CTMC annotated with probabilities after applying transient Markov chain analysis with the CSL property, $\mathtt{F}(t \leq 100, \mathrm{LacI} = 0)$.

4. RESULTS

This section presents the application of our stochastic model checking methodology to the analysis of four genetic circuits, two oscillators and two state holding gates.

4.1. Repressilator

The repressilator model shown in Figure 5 is comprised of the species CI, LacI, and TetR that are connected in a loop [Elowitz and Leibler 2000]. This circuit is a ring oscillator where each species represses the next one forming the loop. When one of the species (e.g., LacI) is produced, it represses the next species in the chain (e.g., TetR). This repression allows the species downstream of that species to start being produced (e.g., CI) which in turn leads to the repression of the first species. This cycle continues causing this circuit to oscillate.

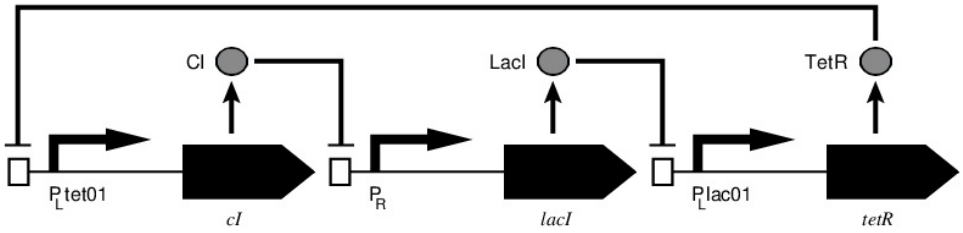


Fig. 5: Model for repressilator. In this circuit, CI represses the production of LacI, LacI represses the production of TetR, and TetR represses the production of CI. This model is expected to produce oscillatory behavior of the CI, LacI, and TetR species.

Figure 6 presents the results of applying steady-state analysis to the repressilator using 9 levels evenly spaced between 0 and 80 for CI, LacI, and TetR and the CSL property, $\mathtt{St}((\mathrm{CI} \geq 30 \wedge \mathtt{F}(t \leq \mathrm{limit}, \mathrm{CI} < 30) \geq 0.95) \vee (\mathrm{CI} < 30 \wedge \mathtt{F}(t \leq \mathrm{limit}, \mathrm{CI} \geq 30) \geq 0.95))$, which determines the likelihood that the value of the CI species is low and goes high or is high and goes low within a predetermined amount of time. These results show that as the time limit is extended, the likelihood of the circuit oscillating increases until for 800 seconds, it is almost certain to oscillate. In addition to the probability increasing as the time limit increases, the run-time also increases because it takes longer to perform analysis of nested properties with larger time bounds. The run-time for a time limit of 200 seconds is 3 minutes, for 400 seconds is 5 minutes and 40 seconds, for 600 seconds is 8 minutes and 25 seconds, and for 800 seconds is 11 minutes. This type of analysis can give a designer an idea of how reliable a circuit like the repressilator is as compared to other oscillator circuits so that the best circuit can be selected for a given task.

4.2. Dual-Feedback Genetic Oscillator

The dual-feedback genetic oscillator model shown in Figure 7 is composed of two identical promoters, P_1 and P_2, which are activated by AraC and repressed by LacI [Stricker et al. 2008]. LacI is produced when transcription is initiated at promoter P_1 and the *lacI* gene is transcribed. Similarly, AraC is produced when transcription is initiated at promoter P_2 and the *araC* gene is transcribed. AraC builds up in the system causing LacI to build up. LacI is then able to repress promoters P_1 and P_2 which causes both AraC and LacI concentrations to drop leading to AraC building up again causing the circuit to oscillate.

Similar to the repressilator, the dual-feedback genetic oscillator can be analyzed using stochastic model checking to determine the probability that it oscillates within a certain time bound. Figure 8 presents the results of applying steady-state analysis to this model using 8 levels evenly spaced between 0 and 120 for AraC and LacI and the CSL property, $\mathtt{St}((\mathrm{AraC} \geq 60 \wedge \mathtt{F}(t \leq \mathrm{limit}, \mathrm{AraC} < 60) \geq 0.95) \vee (\mathrm{AraC} < 60 \wedge \mathtt{F}(t \leq \mathrm{limit}, \mathrm{AraC} \geq 60) \geq 0.95))$, which captures the probability that AraC is low and goes high or is high and goes low within a predetermined amount of time. Like the repressilator, these results show that as the time limit is extended, the likelihood of the circuit oscillating increases. The run-time for a time limit of 1000 seconds is 40 seconds, for 2000 seconds is 1 minute and 25 seconds, for 3000 seconds is 1 minute and 55 seconds, and for 4000 seconds is

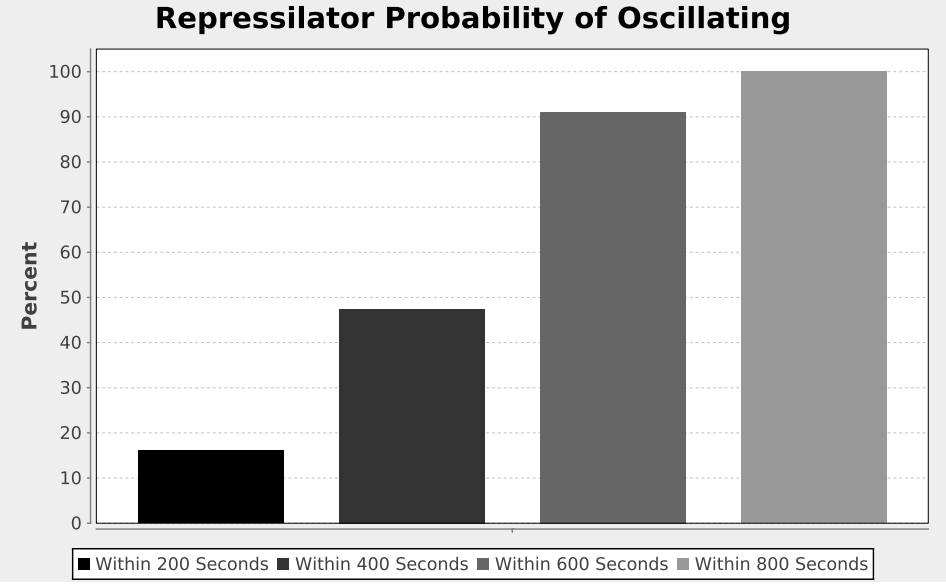


Fig. 6: Stochastic model checking results for the repressilator. This plot presents the results of checking a property on the repressilator that represents states of the circuit switching from low to high or high to low within a specified amount of time. The probabilities represent the likelihood that the circuit oscillates.

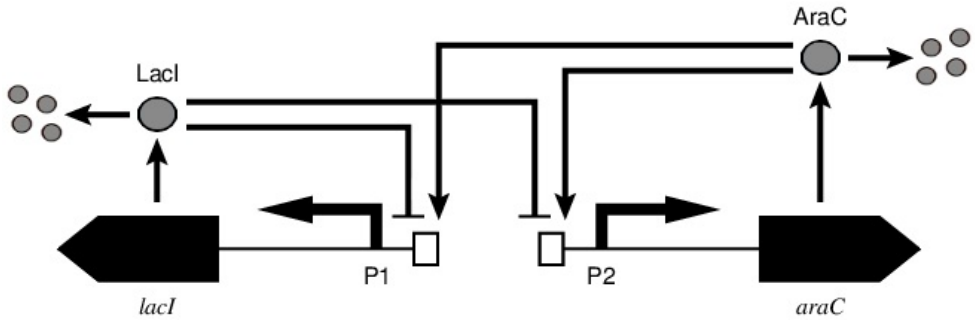


Fig. 7: Model for the dual-feedback genetic oscillator. In this model, LacI represses itself and AraC through promoters P_1 and P_2. Conversely, AraC activates itself and LacI through the same promoters. This model is expected to produce oscillatory behavior of the LacI and AraC species.

2 minutes and 40 seconds. These results show that the dual-feedback genetic oscillator has a much longer period than the repressilator.

4.3. Genetic Toggle Switch

A useful experiment for the genetic toggle switch is to determine the probability that it changes state erroneously within a cell cycle (2,100 seconds) which occurs if some spurious production of the low signal inhibits the high signal enough to allow it to degrade away and switch state. For this experiment, the toggle switch is initialized to a starting state where LacI is set to a high state of 60 molecules and TetR is set to a low state of 0 molecules. In order to test whether or not it changes state, the CSL property, $\mathrm{F}(t \leq 2100, \mathrm{LacI} < 20 \wedge \mathrm{TetR} > 40)$, is checked. This property makes states

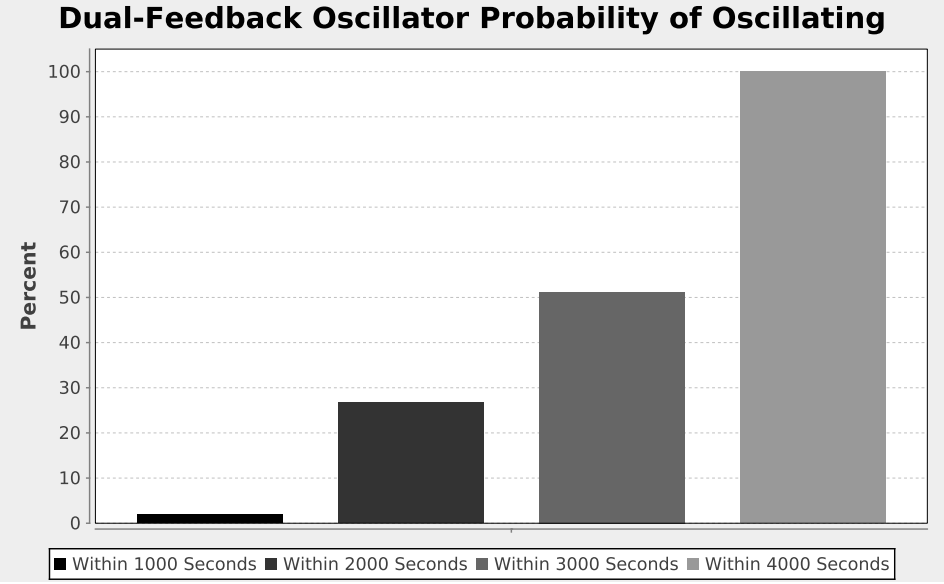


Fig. 8: Stochastic model checking results for the dual-feedback genetic oscillator. This plot presents the results of checking a property on the dual-feedback genetic oscillator that represents states of the circuit switching from low to high or high to low within a specified amount of time. The probabilities represent the likelihood that the circuit oscillates.

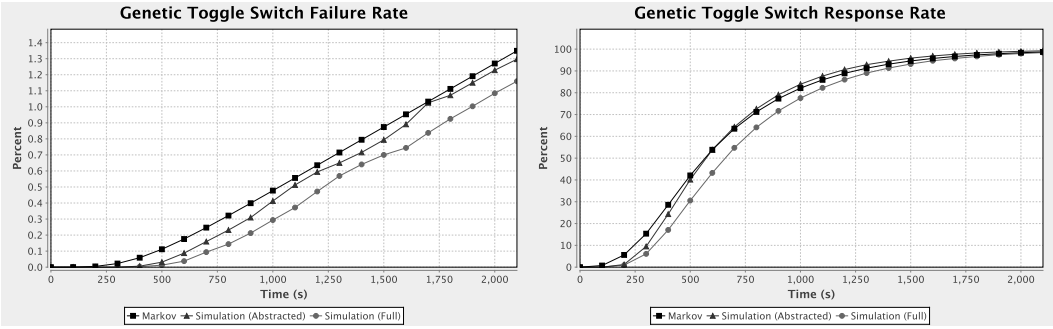
absorbing in which LacI has dropped below 20 (the low state) and TetR has risen above 40 (the high state). For this analysis, 9 levels are selected for LacI uniformly distributed between 0 and 80, and 11 levels are selected for TetR uniformly distributed between 0 and 50, which produces a CTMC with 99 states. Levels are selected to ensure that one of the levels captures the initial amount for each species and that the levels span over the possible values for each species going slightly above and below the property bounds. For comparison, the FSP would generate a CTMC of 4,131 states for the same upper and lower bounds on LacI and TetR.

Figure 9(a) shows a comparison of results found using the SSA both with and without reaction-based abstractions [Kuwahara et al. 2006; Kuwahara 2007] and applying transient Markov chain analysis. It should be noted that the reaction-based abstractions used in the simulated models utilize the same quasi-steady-state approximation [Rao and Arkin 2003] as the logical abstractions presented in Section 3. This figure shows that the transient Markov chain analysis tracks the simulation results fairly closely and ends up with a final probability of 1.35 percent which is quite close to the 1.18 percent found by simulation of the full model. However, the transient Markov chain analysis method greatly outperforms the simulation based approaches as it takes under one second to obtain results while the simulation with abstraction takes 3 minutes and 15 seconds to perform 32,000 runs and the simulation without abstraction takes 43 minutes to perform the same number of runs. It is clear from Figure 9(a) that the stochastic simulations have not yet converged to the mean. When doing stochastic simulation, to have a 95 percent confidence interval, the relative error bound, d, is given by the equation:

$$d = 1.96 \times \sqrt{\frac{1-p}{p \times n}} \tag{4}$$

where p is the predicted probability and n is the number of simulation runs [Kuwahara and Mura 2008; Gillespie et al. 2009]. In order to fairly compare the transient Markov chain analysis that

has about a 10 percent error bound to the simulation methods, 32,000 simulation runs are required to be 95 percent confident that the results are within 10 percent of the actual results assuming a probability of failure of 1.2 percent.



(a) (b)

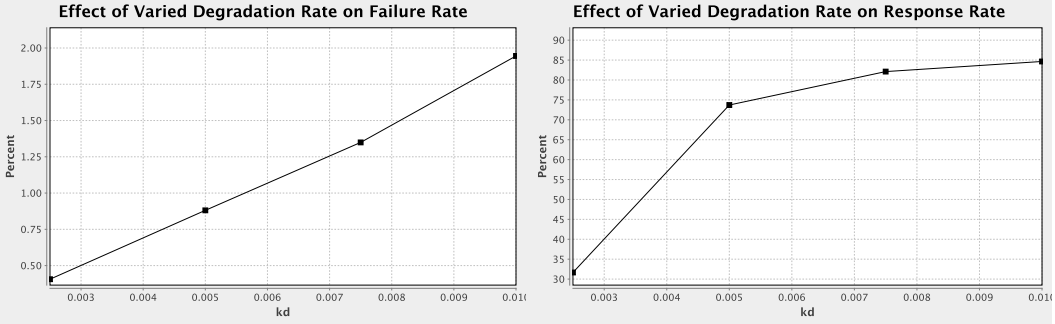
Fig. 9: (a) Time course plot comparing a relative error bound of 10 percent for simulation both with and without reaction-based abstraction and the results of Markov chain analysis. Each simulation requires 32,000 simulation runs to obtain this error bound. (b) Time course plot showing the probability of the genetic toggle switch changing state correctly in response to an input change. This plot also compares the results of 32,000 simulation runs both with and without reaction-based abstraction and the results of Markov chain analysis using the same CSL property but with a different initial value of IPTG.

The next experiment is to determine the response time of the circuit when switching from the off state to the on state, and these results are presented in Figure 9(b). This analysis uses the same CSL property but a slightly different initial condition. As before, LacI is set to 60 and TetR is set to 0, but IPTG is set to 100 representing that it has just been added to set the toggle switch to the high state. For this experiment, 14 levels for LacI are selected uniformly distributed between 0 and 130, since individual simulation results show it reaching a much higher value than in the last experiment. For TetR, only 5 levels are used uniformly selected between 0 to 60 because less resolution is required to catch its change from a low to high state. This level selection results in a CTMC of 70 states (two orders of magnitude fewer than the 7,991 states that the FSP would generate). Again, transient Markov chain analysis tracks the simulation results fairly closely ending up with a final probability of 98.7 percent while the simulation of the full model results in 98.9 percent. Also like the previous example, the transient Markov chain analysis method outperforms the simulation-based approaches as it takes a half a second to obtain results while 32,000 simulation runs of the reaction-based abstracted model takes 1 minute and the full model takes about 3 hours and 12 minutes due to explicit simulation of the binding and unbinding reactions.

With this analysis method, the design space can be efficiently explored. For example, a genetic designer may consider the effect of parameter variation on robustness and performance. One important parameter for the genetic toggle switch is the degradation rate, k_d, and the results of varying this parameter are shown in Figure 10. These results indicate that tuning the degradation rate has a significant effect. If it is too high, the circuit is less robust, but if it is too low, it responds too slowly.

4.4. Genetic Muller C-Element

The next circuit analyzed is a genetic Muller C-element [Muller and Bartky 1959; Nguyen et al. 2010]. C-elements are critical state-holding gates commonly used by asynchronous designers to coordinate parallel processes. In a two-input C-element, its output goes high when both inputs are



(a) (b)

Fig. 10: (a) Plot depicting probability of the genetic toggle switch changing state erroneously within 2,100 seconds for different values of k_d. (b) Plot depicting probability of the genetic toggle switch changing state correctly within 1,000 seconds in response to input change for different values of k_d.

high, goes low when both inputs are low, and it retains its previous state when the inputs are mixed. To construct asynchronous genetic circuits, a C-element could be very useful. For example, consider an application in which it is desired that a population of bacteria respond to a stimulus. In nature, it is common for a bacterial population to use a mechanism known as *quorum sensing* to communicate by producing a small molecule that is released into its environment and taken up by neighboring cells. Using this quorum sensing mechanism, a synthetic genetic circuit can be designed to make a more reliable response. Namely, the bacteria can be programmed to only respond when it detects both the stimulus and the quorum signal indicating agreement by the population that the stimulus is present. This circuit using a C-element would then continue to respond until both the stimulus and quorum signal disappear.

Our analysis considers three implementations of the C-element circuit with their logic diagrams shown in Figure 11. For each of these circuits, the inputs are IPTG and aTc and the output is GFP similar to the genetic toggle switch example. The first implementation shown in Figure 11(a) has a majority gate design where the two inputs to the circuit and a feedback signal from the circuit's output are fed through three NAND gates in varying combinations. The outputs from these NAND gates are then compared with each other and the signal that has the majority of the votes is selected as the new output. The next implementation shown in Figure 11(b) has a speed independent design. The idea behind this circuit is that no matter how fast or slow the gates change, the circuit behaves correctly. The final implementation shown in Figure 11(c) uses the genetic toggle switch described earlier with some additional logic. This circuit takes an inverted NAND gate signal of the two inputs as the set part of the circuit and an inverted NAND gate signal of the inverted inputs as the reset part of the circuit.

Once again, these circuits are analyzed to find their failure and response rates. Instead of analyzing these circuits against a property that only checks the amount of GFP, dual-rail properties are analyzed because they provide a better measure of the circuit changing state. Figure 12(a) shows a comparison of the failure rate analysis when each circuit is set in its high mixed state meaning that one input is high, one input is low, the output is high, and the internal species are set appropriately. For this analysis, the CSL property, $\mathtt{F}(t \leq 2100, \mathrm{GFP} < 20 \wedge \mathrm{E} > 40)$, is used to analyze the majority gate implementation with 16 evenly spaced levels for GFP between 0 and 150, 16 evenly spaced levels for E between 0 and 45, 6 evenly spaced levels for D between 0 and 250, and 5 evenly spaced levels for X, Y, and Z between 0 and 120. The CSL property for the speed-independent implementation is $\mathtt{F}(t \leq 2100, \mathrm{S3} < 20 \wedge \mathrm{S2} > 80)$, and the levels used are 11 evenly spaced levels for S2 between 0 and 100, 11 evenly spaced levels for S3 between 0 and 150, 6 evenly spaced levels for S1 between

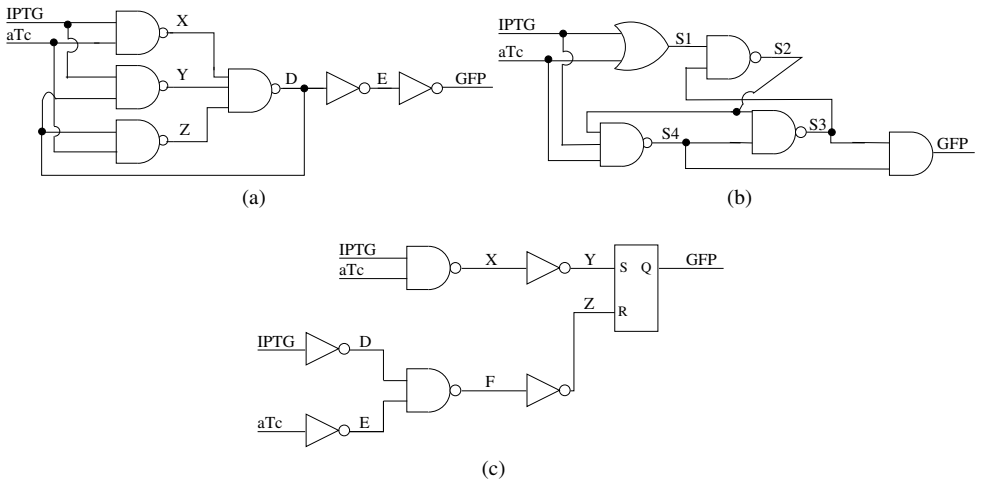


Fig. 11: Logic diagrams for the genetic Muller C-element. (a) Majority gate design. (b) Speed independent design. (c) Toggle switch design.

0 and 250, 4 evenly spaced levels for S4 and GFP between 0 and 120, and 4 evenly spaced levels for X, Y, and Z between 0 and 90. Finally, the CSL property for the toggle switch implementation is $\mathbf{F}(t \leq 2100, \mathrm{Y} < 40 \wedge \mathrm{Z} > 80)$, and the levels used are 16 evenly spaced levels for Y between 0 and 225, 16 evenly spaced levels for Z between 0 and 90, 6 evenly spaced levels for F between 0 and 250, 5 evenly spaced levels for GFP between 0 and 120, and 5 evenly spaced levels for D, E, and X between 0 and 80.

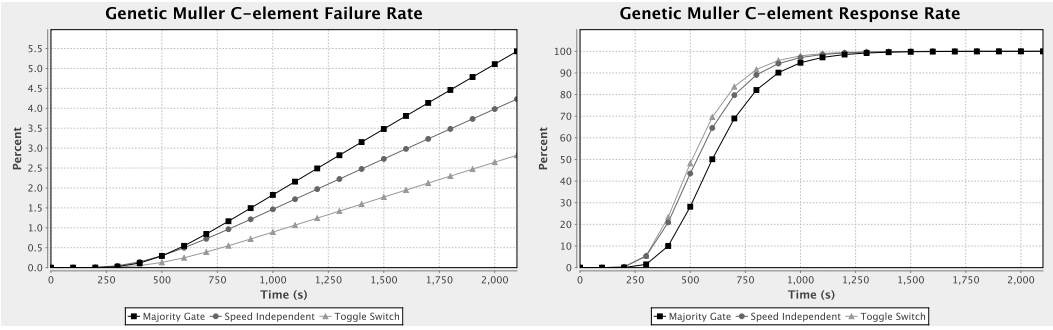


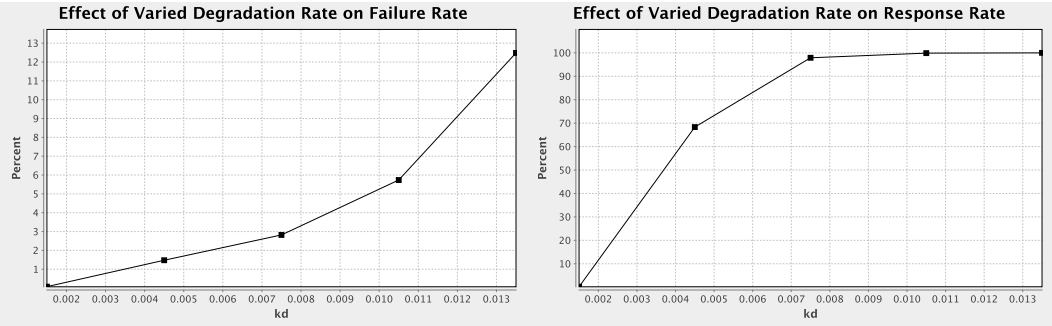
Fig. 12: (a) Time course plot showing the probability of the C-element implementations losing state with mixed inputs. (b) Time course plot showing the probability of the C-element implementations changing state correctly in response to both inputs changing state. These plots compare the results of using Markov chain analysis on the majority gate implementation, the speed independent implementation, and the toggle switch implementation.

Run-times for these analyses are 13 minutes and 15 seconds for the majority gate, 20 minutes and 15 seconds for the speed independent, and 21 minutes and 45 seconds for the toggle switch. This increase in run-time over the genetic toggle switch is due to the fact that each of these models

contains substantially more species. In addition, most of the species in the C-element models have more levels defined for them resulting in larger CTMCs. For instance, compared to the genetic toggle switch's 70 to 99 states, the majority gate implementation has nearly 200,000 states, the speed independent implementation has about 750,000 states, and the toggle switch implementation has nearly 1,000,000 states. For comparison, the FSP would generate a computationally intractable CTMC over eight orders of magnitude larger with about 3.32×10^{14} states for the toggle switch implementation alone. From the plot in Figure 12(a), it is clear that the toggle switch Muller C-element is the most likely to maintain its state with mixed inputs, followed by the speed independent implementation, and finally the majority gate implementation.

In order to determine the response time of the C-element circuits, the same initial condition, levels, and properties for each circuit are used with the exception that both inputs are set low indicating that the circuit's output should change from high to low. The results of this analysis are shown in Figure 12(b) where is can be seen that the toggle switch implementation again outperforms the speed independent and majority gate circuits. These analyses have similar run-times to the failure rate experiment with the majority gate taking 11 minutes and 15 seconds, the speed independent taking 20 minutes, and the toggle switch taking 22 minutes.

After determining that the genetic toggle switch Muller C-element is the most robust (perhaps a surprising conclusion to some), we can now perform various parameter variation experiments to determine the best parameter choices for the application. Figure 13 gives an example of varying the degradation rate of this circuit similar to the experiments performed on the genetic toggle switch in Figure 10. These results show comparable behavior to that of the genetic toggle switch indicating that there is a trade-off between robustness and responsiveness when varying this parameter.



(a) (b)

Fig. 13: (a) Plot depicting the probability of the toggle switch implementation of the genetic Muller C-element changing state erroneously within 2,100 seconds for different values of k_d. (b) Plot depicting the probability of the toggle switch implementation of the genetic Muller C-element changing state correctly within 1,000 seconds in response to an input change for different values of k_d.

5. DISCUSSION

Utilizing stochastic model checking, synthetic biologists can explore the effect of varying parameters in their genetic circuits more rapidly than using traditional methods allowing them to rapidly explore design trade-offs in an effort to make their circuits more responsive to inputs and more robust to failures. Even in cases where circuits become complex, a user can define fewer levels for each species so that the state space of the model does not grow too large. Although defining fewer levels may produce less accurate results, it is still very useful for design space exploration. Indeed, unquantifiable error is often introduced in electronic circuit design where switch-level abstractions

are used to get a better understanding of how a circuit behaves. Once a designer settles on a design using these abstractions, he or she likely uses a more exact analysis approach such as a SPICE simulation for further refinements.

The methods discussed here have been implemented in the tool, `iBioSim` [Myers et al. 2009], which is freely available at `http://www.async.ece.utah.edu/iBioSim/`. This tool includes a schematic capture tool for constructing genetic circuit representations of designs, automated model construction and abstraction, and a variety of simulation and visualization methods. While we believe that this tool represents an excellent first step towards a fully functional GDA tool, there is still significant work that needs to be done.

One important challenge is that it can be difficult to choose the levels on the species amounts. If these levels are chosen poorly, the state space exploration and Markov chain analysis are slow when there are too many states or inaccurate when there are too few states. Currently, a user can run a few simulations to get a general feel for what levels should be selected. One improvement to this method would be to analyze the rate equations of the original genetic circuit model and determine which levels would be best for the species' amounts.

Currently, the Markov chain analyzer only allows a single rate value to be used on a transition between two states, but it could be extended to deal with ranges of rates. For example, one approach could be to analyze the state graph using the lower bounds and then re-analyze using the upper bounds and then display a range of probabilities for each state. An issue when dealing with ranges is that the probabilities of all of the states need to sum to 1. Therefore, picking a probability out of the range from each state may end up leading to a distribution that is not consistent. One possible approach to solving this problem is to utilize a three-valued abstraction [Katoen et al. 2012].

In addition to this work, we are also developing a variant of the SSA called the *incremental stochastic simulation algorithm* (iSSA) [Kuwahara et al. 2010; Madsen 2013; Winstead et al. 2010]. The idea behind this algorithm is to perform stochastic simulations in small time increments, compute statistics at the end of each increment, and determine a new starting state for each run at the beginning of the next increment with these statistics. This allows users to perform many simulation runs to observe the "typical" behavior of their systems instead of simply averaging several SSA runs together which can often hide interesting behaviors due to a washing out effect. Ideally, iSSA and stochastic model checking could be used in concert to improve the genetic circuit design process. For example, the typical behavior of the system could be determined using iSSA, then the probability of this typical behavior being observed could be checked using a CSL property and stochastic model checking.

To date, only small genetic circuits can be readily constructed in the laboratory composed of just a few genes, so designing genetic circuits by hand at the schematic level makes sense. However, the number of elements in the registry of standard biological parts (see `http://partsregistry.org`) is growing almost daily. Furthermore, the cost and efficiency of DNA synthesis is also steadily improving. Therefore, automated synthesis and technology mapping methods must be developed in the future. We are working at adapting new methods for this purpose. There are, however, challenges when dealing with the inherently stochastic nature of genetic circuits. Cost functions must not only consider complexity, but they must also consider a design's robustness. Hence, it is crucial that stochastic analysis techniques like the one described in this paper be utilized throughout the design exploration process.

An interesting fact though is that future silicon and proposed nano devices will likely also be highly noisy and ultimately unreliable. Therefore, designers of electronic circuits must face the prospect of needing to find a way to produce reliable systems using unreliable components. Since GDA tools will have already had to address this problem, they can potentially be utilized to produce better electronic circuits.

REFERENCES

Anderson, J. C., Clarke, E. J., and Arkin, A. P. 2006. Environmentally controlled invasion of cancer cells by engineering bacteria. *J. Mol. Biol. 355*, 619–627.

ARKIN, A. 2008. Setting the standard in synthetic biology. *Nature Biotech. 26*, 771–774.

ATSUMI, S. AND LIAO, J. C. 2008. Metabolic engineering for advanced biofuels production from *Escherichia coli*. *Current Opinion in Biotechnology 19,* 5, 414 – 419. Tissue, cell and pathway engineering.

AZIZ, A., SANWAL, K., SINGHAL, V., AND BRAYTON, R. 2000. Model-checking continuous-time Markov chains. *ACM Trans. Comput. Logic 1*, 162–170.

BEAL, J. AND BACHRACH, J. 2008. Cells are plausible targets for high-level spatial languages. *Proceedings of the 2008 Second IEEE International*, 284–291.

BHATIA, S. AND DENSMORE, D. 2013. Pigeon: a design visualizer for synthetic biology. *ACS Synth. Biol. 2,* 6, 348–350.

BILITCHENKO, L., LIU, A., CHEUNG, S., WEEDING, E., XIA, B., LEGUIA, M., ANDERSON, J. C., AND DENSMORE, D. 2011. Eugene - a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS ONE 6,* 4.

BURRAGE, K., HEGLAND, M., MACNAMARA, S., AND SIDJE, R. 2006. A krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In *Mam 2006 : Markov Anniversary Meeting: an international conference to celebrate the 150th anniversary of the birth of A.A. Markov*, A. N. Langville and W. J. Stewart, Eds. Boston Books, Charleston, South Carolina, 21–38.

CAI, Y., WILSON, M. L., AND PECCOUD, J. 2010. GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs. *Nucleic Acids Res. 38,* 8, 2637–44.

CAO, Y., GILLESPIE, D. T., AND PETZOLD, L. R. 2006. Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys. 124*.

CASES, I. AND DE LORENZO, V. 2005. Genetically modified organisms for the environment: stories of success and failure and what we have learned from them. *International Microbiology 8*, 213–222.

CHANDRAN, D., BERGMANN, F. T., AND SAURO, H. M. 2009. TinkerCell: modular CAD tool for synthetic biology. *J. Biol. Eng. 3,* 19.

CIOCCHETTA, F., DEGASPERI, A., HILLSTON, J., AND CALDER, M. 2009. Some investigations concerning the CTMC and the ODE model derived from Bio-PEPA. *Electronic Notes in Theoretical Computer Science 229,* 1, 145 – 163. Proceedings of the Second Workshop From Biology to Concurrency and Back (FBTC 2008).

ELOWITZ, M. AND LEIBLER, S. 2000. A synthetic oscillatory network of transcriptional regulators. *Nature 403,* 6767, 335–338.

ENDY, D. 2005. Foundations for engineering biology. *Nature 438*, 449–453.

GARDNER, T. S., CANTOR, C. R., AND COLLINS, J. J. 2000. Construction of a genetic toggle switch in *Escherichia coli*. *Nature 403*, 339–342.

GIBSON, M. AND BRUCK, J. 2000. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A 104*, 1876–1889.

GILLESPIE, D. T. 1977. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem. 81,* 25, 2340–2361.

GILLESPIE, D. T. AND PETZOLD, L. R. 2003. Tau leaping. *J. Chem. Phys. 119*, 8229–8234.

GILLESPIE, D. T., ROH, M., AND PETZOLD, L. R. 2009. Refining the weighted stochastic simulation algorithm. *J Chem Phys 130,* 17.

HAHN, E. M., HERMANNS, H., WACHTER, B., AND ZHANG, L. 2009. Time-bounded model checking of infinite-state continuous-time Markov chains. *Fundam. Inf. 95*, 129–155.

HENZINGER, T., MATEESCU, M., AND WOLF, V. 2009. Sliding window abstraction for infinite Markov chains. In *Computer Aided Verification*, A. Bouajjani and O. Maler, Eds. Lecture Notes in Computer Science Series, vol. 5643. Springer Berlin / Heidelberg, 337–352.

HINTON, A., KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. 2006. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, H. Hermanns and J. Palsberg, Eds. LNCS Series, vol. 3920. Springer, 441–444.

KATOEN, J.-P., KLINK, D., LEUCKER, M., AND WOLF, V. 2012. Three-valued abstraction for probabilistic systems. *J. Log. Algebr. Program. 81,* 4, 356–389.

KUWAHARA, H. 2007. Model abstraction and temporal behavior analysis of genetic regulatory networks. Ph.D. thesis, U. of Utah.

KUWAHARA, H., MADSEN, C., MURA, I., MYERS, C., TEJADA, A., AND WINSTEAD, C. 2010. Efficient stochastic simulation to analyze targeted properties of biological systems. In *Stochastic Control*, C. Myers, Ed. Sciyo, 505–532.

KUWAHARA, H. AND MURA, I. 2008. An efficient and exact stochastic simulation method to analyze rare events in biochemical systems. *The Journal of Chemical Physics 129,* 16.

KUWAHARA, H., MYERS, C., BARKER, N., SAMOILOV, M., AND ARKIN, A. 2006. Automated abstraction methodology for genetic regulatory networks. *Trans. Comp. Syst. Biol. VI*, 150–175.

KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. 2007. Stochastic model checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, M. Bernardo and J. Hillston, Eds. LNCS (Tutorial Volume) Series, vol. 4486. Springer, 220–270.

Madsen, C. 2013. Stochastic analysis of synthetic genetic circuits. Ph.D. thesis, University of Utah.

Marchisio, M. A. and Stelling, J. 2008. Computational design of synthetic gene circuits with composable parts. *Bioinform. 24,* 17, 1903–10.

Marchisio, M. A. and Stelling, J. 2011. Automatic design of digital synthetic gene circuits. *PLoS Comput Biol 7,* 2.

McAdams, H. H. and Arkin, A. 1999. Genetic regulation at the nanomolar scale: it's a noisy business. *Trends Genet. 15,* 2, 65–69.

Muller, D. E. and Bartky, W. S. 1959. A theory of asynchronous circuits. In *Proc. International Symposium on the Theory of Switching*. Harvard University Press, Cambridge, MA, 204–243.

Munsky, B. and Khammash, M. 2006. The finite state projection algorithm for the solution of the chemical master equation. *The Journal of Chemical Physics 124,* 4.

Myers, C. J. 2009. *Engineering Genetic Circuits*. Chapman and Hall/CRC.

Myers, C. J., Barker, N., Jones, K., Kuwahara, H., Madsen, C., and Nguyen, N.-P. D. 2009. iBioSim: a tool for the analysis and design of genetic circuits. *Bioinform. 25,* 21, 2848–2849.

Nguyen, N., Myers, C., Kuwahara, H., Winstead, C., and Keener, J. 2010. Design and analysis of a robust genetic Muller C-element. *Journal of Theoretical Biology 264,* 2, 174 – 187.

Pedersen, M. and Phillips, A. 2009. Towards programming languages for genetic engineering of living cells. *J. R. Soc. Interface 6,* (Suppl. 4), S437–S450.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. 1992. *Numerical Recipes in C: The art of ScientiiňĄc Computing, 2nd ed.* Cambridge University Press.

Quinn, J., Beal, J., Bhatia, S., Cai, P., Chen, J., Clancy, K., Hillson, N. J., Galdzicki, M., Maheshwari, A., Umesh, P., Pocock, M., Rodriguez, C., Stan, G.-B., and Endy, D. 2013. Synthetic Biology Open Language Visual (SBOL Visual), Version 1.0.0. BBF RFC 93.

Rao, C. V. and Arkin, A. P. 2003. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. *J. Phys. Chem. 118,* 11.

Ro, D.-K., Paradise, E. M., Ouellet, M., Fisher, K. J., Newman, K. L., Ndungu, J. M., Ho, K. A., Eachus, R. A., Ham, T. S., Kirby, J., Chang, M. C. Y., Withers, S. T., Shiba, Y., Sarpong, R., and Keasling, J. D. 2006. Production of the antimalarial drug percursor artemisinic acid in engineered yeast. *Nature 440*.

Slepoy, A., Thompson, A. P., and Plimpton, S. J. 2008. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics 128,* 20, 205101.

Stewart, W. J. 1994. *Introduction to the Numerical Solution of Markov Chains*. Priceton University Press, 41 William Street, Princeton, NJ, 08540.

Stricker, J., Cookson, S., Bennett, M., Mather, W., Tsimring, L., and Hasty, J. 2008. A fast, robust and tunable synthetic gene oscillator. *Nature 456*, 516–519.

Strogatz, S. H. 1994. *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry And Engineering*. Westview Press.

Thieffry, D. and Thomas, R. 1995. Dynamical behaviour of biological networks: Ii. immunity control in bacteriophage lamabda. *Bull. Math. Biol. 57,* 2, 277–297.

Thomas, R. 1991. Regulatory metworks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology 153*, 1–23.

Tyson, J. and Othmer, H. 1978. The dynamics of feedback control circuits in biochemical pathways. *Progress in Theoretical Biology 5*, 1–62.

Waage, P. and Guldberg, C. M. 1864. Studies concerning affinity. *Forhandlinger: Videnskabs - Selskabet i Christinia 35*.

Winstead, C., Madsen, C., and Myers, C. J. 2010. iSSA: An incremental stochastic simulation algorithm for genetic circuits. In *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 553–556.

Yordanov, B., Tumova, J., Belta, C., Cerna, I., and Barnat, J. 2010. Formal analysis of piecewise affine systems through formula-guided refinement. In *49th IEEE Conference on Decision and Control (CDC)*. 5899–5904.

Younes, H., Kwiatkowska, M., Norman, G., and Parker, D. 2006. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT) 8*, 216–228.