Chapter 12
Approximation Techniques for Stochastic Analysis of Biological Systems

Thakur Neupane, Zhen Zhang, Curtis Madsen, Hao Zheng and Chris J. Myers

Abstract There has been an increasing demand for formal methods in the design process of safety-critical synthetic genetic circuits. Probabilistic model checking techniques have demonstrated significant potential in analyzing the intrinsic probabilistic behaviors of complex genetic circuit designs. However, its inability to scale limits its applicability in practice. This chapter addresses the scalability problem by presenting a state-space approximation method to remove unlikely states resulting in a reduced, finite state representation of the infinite-state continuous-time Markov chain that is amenable to probabilistic model checking. The proposed method is evaluated on a design of a genetic toggle switch. Comparisons with another state-of-the-art tool demonstrate both accuracy and efficiency of the presented method.

12.1 Introduction

Computational biologists typically construct models to better understand and explore the possible behaviors of biological systems [35]. By using formal methods, such as model checking, to analyze these models, researchers are able to ensure that certain

T. Neupane · Z. Zhang (✉)
Utah State University, Logan, UT, USA
e-mail: zhen.zhang@usu.edu

T. Neupane
e-mail: thakur.neupane@aggiemail.usu.edu

C. Madsen
Boston University, Boston, MA, USA
e-mail: ckmadsen@bu.edu

H. Zheng
University of South Florida, Tampa, FL, USA
e-mail: haozheng@usf.edu

C. J. Myers
University of Utah, Salt Lake City, UT, USA
e-mail: myers@ece.utah.edu

P. Liò and P. Zuliani (eds.), *Automated Reasoning for Systems Biology and Medicine*, Computational Biology 30,
https://doi.org/10.1007/978-3-030-17297-8_12

properties hold in biological system designs [19]. In order to numerically model check a system, the system's state space must be enumerated. For systems that are highly concurrent and have infinite states, such as *genetic circuits* (i.e., the collections of genes within DNA that interact to control the behavior of cells, see Sect. 12.4 for more details), enumerating the state space can be computationally intractable due to the state-space explosion problem. Techniques such as partial order reduction that reduce the number of reachable states in a system have shown some promise in tackling this problem [3, 4, 9], but these methods often rely on transition dependencies based on the disablings (and/or enablings) and commutativity of independent transitions. Most models of genetic circuits do not contain transitions that disable/enable other transitions leading researchers to seek other solutions to this problem.

Another way to reduce the state space of a system is to introduce threshold abstractions to collapse multiple states of the system together [31]. This type of abstraction works very well in systems where there exist groups of states in equivalence classes. This is often the case in genetic circuits where firing a single transition does not have a great effect on the likelihood of firing other transitions in the system. Although this type of abstraction has previously been successfully applied to genetic circuits, selecting the threshold values is currently done in a manual ad hoc manner.

This chapter presents an alternative method for deriving a reduced, finite-state representation of a genetic circuit's behavior. This method works by computing the approximate probability of reaching each state on the fly and stops exploring different paths when the cumulative path probability drops below a predetermined value, and these paths are routed to an abstract absorbing state, which accumulates probability leakage during the Markovian analysis. The resulting *continuous-time Markov chain* (CTMC) can be analyzed using probabilistic model checking approaches to determine the probability that the original genetic circuit satisfies a desired temporal logic property given in *continuous stochastic logic* (CSL) [2, 25]. This chapter illustrates the utility of this method by applying it to a model of the genetic toggle switch and by comparing the results to a previous approach where the thresholds were determined by hand to produce the finite-state representation [29, 31]. Additionally, this method is compared with a state-of-the-art stochastic hybrid analysis tool on several benchmarks, and comparisons of results demonstrate both accuracy and efficiency of our proposed method.

12.2 Related Work

To improve the scalability of probabilistic model checking, *bi-simulation minimization* (e.g., [11–13]) has been extended to the probabilistic setting [22] to achieve up to a logarithmic state-space reduction. *Probabilistic abstraction* (e.g., [10, 21, 23]) applies coarser state merging to achieve better reduction, while ensuring a simulation relation between the abstract and concrete Markov models. A transition on the abstract Markov model has a range of probabilities, represented by an interval with the maximal and minimal probabilities for taking the transition. In particular, [23]

presents a theoretical framework for reducing *discrete-time Markov chains* (DTMCs) and CTMCs using a three-valued abstraction and for model checking these abstractions. However, how to partition the state space in this framework is not discussed, nor is the refinement of the abstractions in the case that inconclusive results are produced. Although these reduction techniques can be powerful, they may not be effective in alleviating the exponential state growth caused by concurrency as they are not designed to tackle concurrency in the first place. Unfortunately, concurrency is inherent in most synthetic biological systems.

To address the state explosion problem, some approaches attempt to truncate the state space. For instance, [32] presents a method for selectively exploring states involving rare events; however, this technique requires the modification of parameters in the system to help guide this exploration. Other approaches attempt to dynamically explore the state space and continually add states until the resulting state space satisfies a desired level of precision [5, 33, 34].

A probabilistic counterexample-guided abstraction refinement approach is developed in [21, 38]. Predicate abstraction is applied to programs in a probabilistic guarded command language, and counterexamples are represented as finite Markov chains, where additional predicates are extracted by using an SMT solver in the case that such counterexamples are spurious. [26] presents a compositional verification approach to probabilistic systems using assume-guarantee reasoning. Both component assumptions and guarantees are represented as probabilistic safety properties. Component verification can be expensive in this approach as it is reduced to a linear programming problem. Furthermore, assumptions are derived manually. Additionally, [21, 26, 38] are all based on probabilistic automata, which support nondeterminism but with discrete-time semantics.

In [31], genetic circuit models are converted into CTMCs using operator site reduction abstractions relying on quasi-steady-state approximations. To avoid the state explosion problem, the authors employ a state aggregation method to collapse states together based on user-provided thresholds. While the application of probabilistic model checking to the reduced CTMC can produce results in a fraction of the time of simulation-based approaches, this method is incapable of quantifying the error introduced by this aggregation and relies on user input for good choices of thresholds. While there has been work to address the former [1], our method attempts to alleviate the latter by automatically determining a finite-state representation by removing states that are found to be extremely unlikely during the generation of the CTMC from the genetic circuit model.

A similar approach to the one presented in this chapter is the sliding window abstraction [20]. This method approximates a solution to the *chemical master equation* (CME) by dividing the time period of interest into small time steps, iteratively constructing a window of an abstract state space that preserves the probability mass at the current time step, and then "sliding" the window in each subsequent time step to include newly generated states with significant probability while abstracting away those with negligible probability contribution until the last time step has elapsed. This method effectively performs transient CTMC analysis on a manageable approximated state space and successively updates a state-space approximation

by following the direction in which probability mass moves as time evolves. The abstract state-space construction is based on a worst-case estimation of lower and upper bounds on the populations of the chemical species.

A more recent improvement of the sliding window implementation is the *STochastic Analysis of biochemical Reaction networks* (STAR) [28]. It computes approximate solutions to population Markov processes using a stochastic hybrid model that combines moment-based and state-based representations of probability distributions, and has been optimized to drop unlikely states and add likely states on the fly.

Our approach differs from the sliding window method in that it does not require many manual factors (e.g., several different initial states to compute a state update, a limited window size, etc.) to compute its state space. Additionally, the method presented in this chapter has the potential to optimize the choice of the termination indicator factor to preserve accuracy while requiring a manageable state space. Finally, our approach is based on a reaction-based abstraction model, and as a result, is readily applied to genetic circuit models while the method in [20] focuses on Markov chains that are specified by a finite set of transition classes.

12.3 Preliminaries

The high-level modeling formalism used in this chapter is the *stochastic chemical kinetic* (SCK) model [35].

Definition 1 A SCK model is a tuple $\mathcal{M} = \langle \mathbf{S}, \mathbf{R}, \mathbf{x}_0 \rangle$ which is composed of n chemical species $\mathbf{S} = \{S_1, \ldots, S_n\}$, m reaction channels $\mathbf{R} = \{R_1, \ldots, R_m\}$, and an initial molecule count of each chemical species at the beginning of analysis (i.e., $\mathbf{x}_0 : \mathbf{S}^n \rightarrow \mathbb{N}$). A reaction $R_i = \langle \alpha_i, v_i \rangle$ includes a *propensity function* $\alpha_i : \mathbb{N}^n \rightarrow \mathbb{R}^+$ that corresponds to the probability of a reaction, and the *state-change vector* $v_i \in \mathbb{Z}^n$ that corresponds to the change in molecule count for each species due to reaction R_i.

A reaction R_i can occur in state $\mathbf{x} \in \mathbf{X}$, if its propensity is greater than zero (i.e., $\alpha_i(\mathbf{x}) > 0$). The propensity function α_i essentially determines the likelihood that R_i occurs in the current state. After a reaction R_i occurs, the state is updated as follows: $\mathbf{x}' = \mathbf{x} + v_i$.

The execution of reactions in an SCK model creates a *state graph* as defined below:

Definition 2 A state graph is a tuple $\mathcal{G} = \langle \mathbf{X}, \delta, \mathbf{x}_0 \rangle$ where

- $\mathbf{X}$ is a non-empty set of states.
- $\delta \subseteq \mathbf{X} \times \mathbf{R} \times \mathbf{X}$ is the set of state transitions.
- $\mathbf{x}_0 : \mathbf{S}^n \rightarrow \mathbb{N}$ is the initial state.

Note that $|\mathcal{G}|$ represents the state count of $\mathcal{G}$.

For most SCK models of real biological networks, they incur an infinite number of states. Therefore, the goal of this chapter is to find a finite subset of the states that sufficiently represents states that are actually likely to occur. Once a finite-state

graph is obtained, properties can be verified on this state graph using probabilistic model checking.

Probabilistic model checking is a formal verification method for checking quantitative properties of probabilistic systems. The models of interest include DTMCs and CTMCs, both of which belong to a class of stochastic processes that are used to reason about random phenomena in application domains such as synthetic biology. Both Markov models are essentially a transition system with each transition labeled by a discrete probability for DTMCs or a transition rate for CTMCs. A DTMC is a transition system with a discrete probability labeled on each transition [25], which describes the likelihood of a single step moving from one state to another. A CTMC, on the other hand, is a transition system with a transition rate $r(s, s')$ labeled on the transition emanating from state s to s'. This rate determines the probability of executing this transition within t time units, which is $1 - e^{-r(s,s')t}$. The rate $r(s, s')$ uniquely characterizes an exponential distribution to govern the average state residence time of state s, which is $\frac{1}{r(s,s')}$. CTMCs allow for modeling of real-time systems, as the delay of a transition can be any arbitrary real value.

Properties to verify using probabilistic model checking are specified using *Probabilistic Computation Tree Logic* (PCTL) [18] for DTMCs and CSL for CTMCs. PCTL extends *Computation Tree Logic* (CTL) [6] by replacing existential and universal path quantifiers with a probability operator, and hence expresses probabilistic properties for a DTMC. In addition to path probabilities, two traditional properties of CTMCs are the *transient* and *steady-state* behaviors. Transient analysis reports the probability of being in each state of the Markov chain at a particular time instant, and steady-state analysis gives the corresponding probability in the long run. Model checking algorithms for PCTL (e.g., [7, 8, 18]) have identical structure to the model checking algorithm for CTL. Model checking CTMC first discretizes the CTMC into an *embedded* DTMC, from which many properties of the corresponding CTMC can be deduced, for example, checking state reachability properties regardless of how long it takes, and the expected time objectives. For checking state reachability within some time bound, the CTMC is discretized into a *uniformized* DTMC with the iterative numerical method *uniformization* [16, 17]. The uniformized DTMC preserves the state resident time so that its transient behavior is equal (up to some accuracy) to the corresponding CTMC.

In order to perform probabilistic model checking on CTMCs, CSL can be used. CSL properties consist of state formulae (formulae that are either true or false in a specific state) and path formulae (formulae that are either true or false along a specific path). CSL properties are specified using the following grammar:

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

where v_i is a variable, c_i is a constant, and t stands for time in the system. In CSL, Ψ is a state formula that can be either comparisons between numerical expressions, ϕ, or other state formula combined using logical connectives. A CSL property, *Prop*, is a path property over state formula. For example, the *Until property* is of the form $\mathtt{U}(T, \Psi_1, \Psi_2)$, and it returns the probability that along paths originating in the current state, Ψ_1 remains true until Ψ_2 becomes true during the time specified by time expression, T. The eventually operator, `F`, is simply a shorthand for an until property where Ψ_1 is `true`. The globally true operator, `G`, is another shorthand that specifies that Ψ remains true during the time in which T evaluates to true. The steady-state operator, `St`, returns the probability that when the SCK model reaches a steady state that it has reached a state where Ψ is true. Finally, CSL formulae, *Prop*, can be nested within other formula, creating recursive properties.

For example, the CSL property $\mathtt{St}(x > 5 \ \wedge \ y \geqslant 10)$ would return the probability that in the steady state, the system reaches a state where the variable x is greater than 5 and the variable y is greater than or equal to 10. Alternatively, the CSL property $\mathtt{F}(t > 100 \wedge \neg(t \geqslant 200), x > 5 \ \wedge \ y \geqslant 10)$ would return the probability that the system follows an execution path originating in the initial state where the variable x becomes greater than 5 and the variable y becomes greater than or equal to 10 sometime between 100 and 200 time units non-inclusive. For a path to satisfy this property, the system does not need these conditions to hold true for the entire 100 time unit interval; they just both need to become true simultaneously at some point within this time frame.

12.4 Motivating Example

A genetic circuit is constructed using DNA, and it typically includes, at a minimum, regions that act as *promoters*, *ribosome binding sites* (RBS), *coding sequences* (CDS), and *terminators*. The promoters are regions where *transcription* is initiated when an *RNA polymerase* (RNAP) molecule binds, and then begins to walk along the DNA copying the sequence to form a *messenger RNA* (mRNA) molecule until it reaches the location of the terminator. The terminator causes the RNAP to be released and thus ends transcription. The RBS region when copied to an mRNA results in a region that binds to a *ribosome* to initiate the *translation* process. During translation, the CDS region on the mRNA is used as instructions following the *genetic code* to select the *amino acids* to use to construct a *protein*. Proteins are a fundamental component for almost all molecular functions within a cell. Proteins can also bind to promoters to *activate* or *repress* transcription, i.e., increasing or decreasing the associated promoters binding affinity to RNAP.

The motivating example used in this chapter is a genetic circuit for a toggle switch [14] shown in Fig . 12.1. This genetic circuit is constructed from two *transcriptional units*. The one on the left begins with the promoter P_{tet} (shown as a bent arrow), followed by its RBS (shown as a half circle), a CDS that codes for the protein LacI, and finally a terminator (shown as a $\top$). The one on the right begins with the

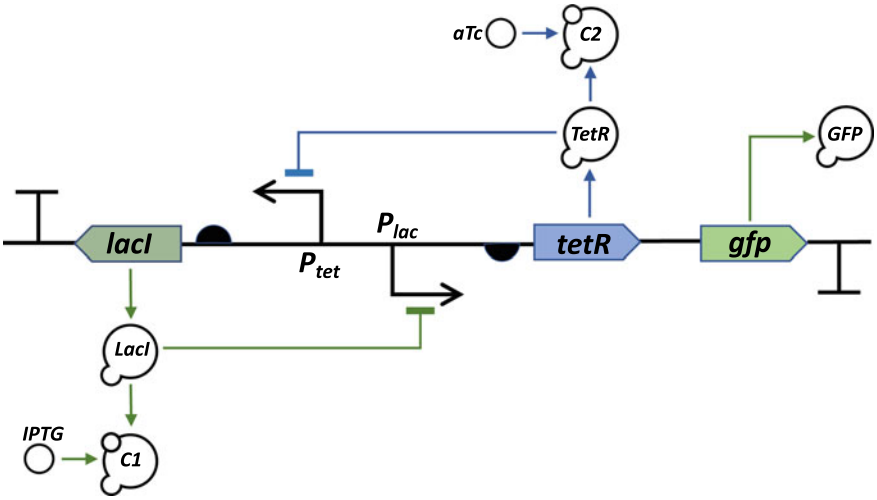


Fig. 12.1 The genetic toggle switch. This switch is created using two repressors, LacI and TetR, which repress each other's production, denoted by the $\perp$ and $\top$ arrows on promoters P_{tet} and P_{lac}. The small-molecule IPTG can bind to LacI, effectively reducing LacI's ability to repress TetR and GFP production. Similarly, the small-molecule aTc can bind to TetR to reduce TetR's ability to repress LacI's production. To indicate the ON and OFF states of this switch, this circuit includes the reporter protein GFP to cause the cell to glow green when it is present

P_{lac} promoter, which initiates transcription of the CDSs for the TetR protein and the *green fluorescent protein* (GFP). GFP is a reporter, since the cells glow green when it is present. The switch-like behavior is created by mutual repression. Namely, the TetR protein binds to P_{tet} to repress LacI production, while the LacI protein binds to P_{lac} to repress TetR production. The state of the switch is changed by adding small-molecule *chemical inducers*. Namely, when the switch is OFF (i.e., LacI is present but no TetR or GFP is present), IPTG can be added, which binds to LacI forming the complex C1, which is unable to repress P_{lac}. This situation leads to TetR and GFP being produced, which represses LacI production and thus changes the switch to the ON state. To change back to the OFF state, aTc can be added, which binds to TetR to form the complex C2, which is unable to repress P_{tet}. This situation leads to LacI being produced, which represses further production of TetR and GFP and thus the changes the genetic toggle switch to the OFF state.

One possible reaction-based model of the genetic toggle switch is shown in Fig. 12.2. This model is derived from a more detailed model, using quasi-steady-state approximations and reaction-based abstractions as described in [24, 35]. This model is composed of a species for each protein (i.e., LacI, TetR, and GFP) and each small molecule (i.e., IPTG and aTc). This model also includes a production reaction for each promoter, P_{tet} and P_{lac}, and a degradation reaction for each protein. The reactions are shown as boxes in the diagram, with their propensity functions shown inside the boxes. The parameters for these propensity functions are given in

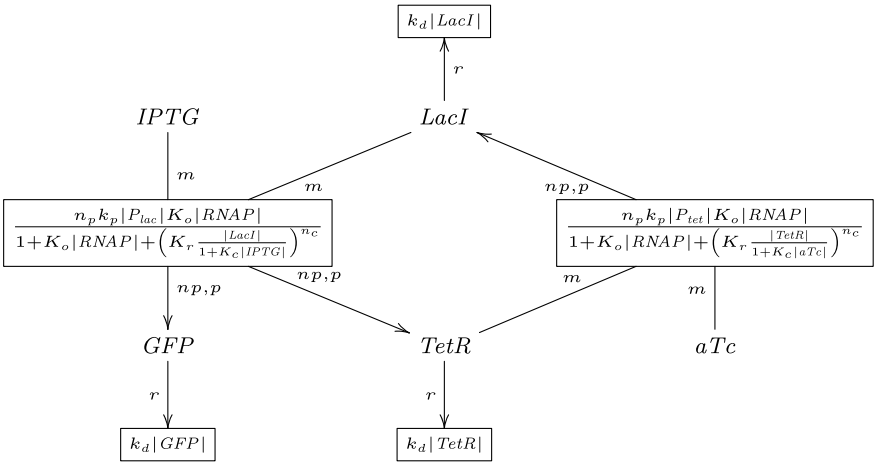


Fig. 12.2 Reaction graph adapted from [29] for the genetic toggle switch after applying reaction-based abstractions to the chemical reaction network

Table 12.1 List of parameters for the genetic toggle switch model

Parameter	Symbol	Value	Units
Degradation rate	k_d	0.0075	sec^{-1}
Complex formation equilibrium	K_c	0.05	molecule^{-1}
Stoichiometry of binding	n_c	2	molecules
Repression binding equilibrium	K_r	0.5	molecule^{-1}
RNAP binding equilibrium	K_o	0.033	molecule^{-1}
Open complex production rate	k_p	0.05	sec^{-1}
Stoichiometry of production	np	10	dimensionless
Number of RNAP molecules	$\|RNAP\|$	30	molecules
Number of P_{tet} promoters	$\|P_{tet}\|$	2	molecules
Number of P_{lac} promoters	$\|P_{lac}\|$	2	molecules

Table 12.1. Note that these are simply reasonable default parameters and not measured experimentally, and they can be easily updated if better information becomes available. The edges are labeled to indicate *reactants* (r), species consumed by the reactions, *products* (p), species produced by the reactions, and *modifiers* (m), species neither produced or consumed. The *stoichiometry*, the number of molecules produced or consumed, for each reaction is assumed to be 1, unless indicated otherwise (e.g., production reactions produce np molecules). The degradation reactions have a propensity that is just the degradation rate, k_d, times the current number of molecules of the species that is degrading. The production reactions have a propensity that is the number of molecules produced, np, times the rate of production, k_p, times the proportion of promoters bound to RNAP in steady state. This proportion is a func-

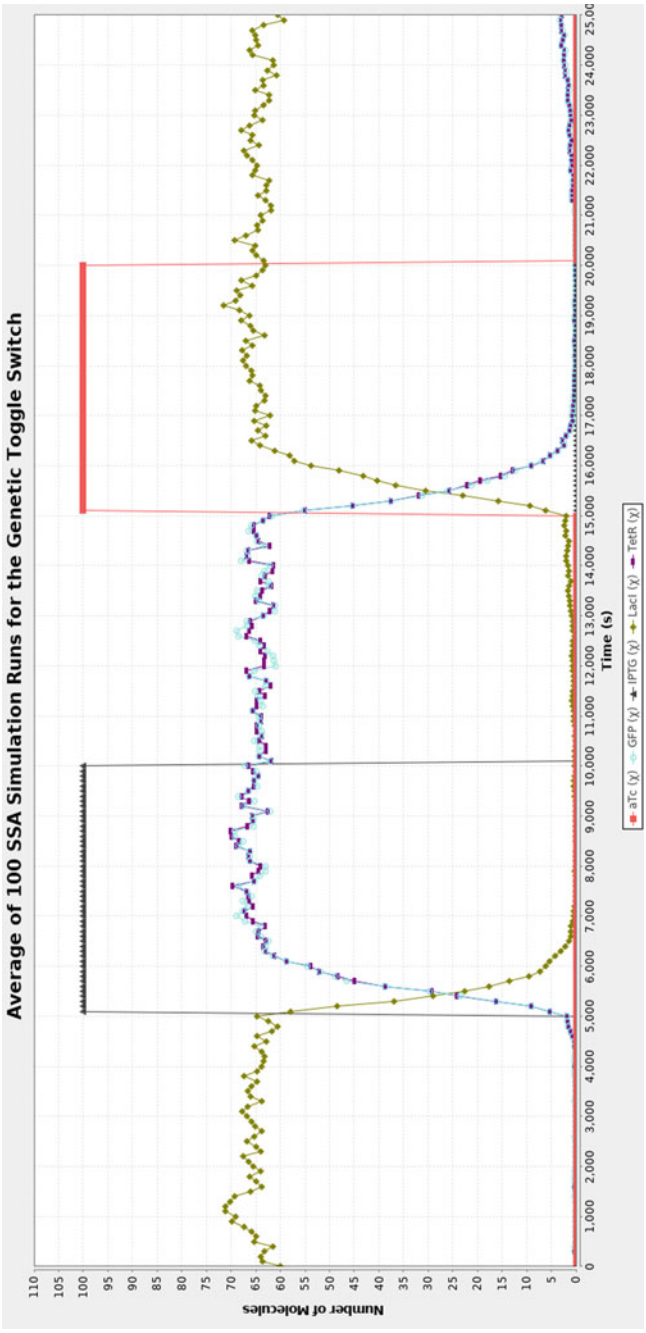


Fig. 12.3 The average of 100 stochastic simulation runs of the genetic toggle switch circuit. The LacI molecule count drops to low soon after the introduction of IPTG at time 5,000 s, which is then removed at time 10,000 s. Its molecule count sharply rises to high when aTc is added at time 15,000 s, which is then removed at time 20,000 s, leaving LacI to stay at a high molecule count. TetR has the opposite behavior, which is closely followed by GFP

tion of the amount of repressor molecules present in free form (i.e., not bound to the corresponding small-molecule inducer). Further details are outside the scope of this chapter, but they can be found in [24, 35].

Unlike an electronic circuit, the behavior of a genetic toggle switch circuit is extremely noisy due to the small-molecule counts involved. It is, therefore, necessary to evaluate a genetic circuit's behaviors using stochastic analyses. Figure 12.3 shows the average output response of 100 stochastic simulation runs using Gillespie's *stochastic simulation algorithm* (SSA) [15]. These simulations start with the same initial state with 60 LacI molecules, and 0 for other species. At time 5,000, 100 molecules of IPTG are applied, which activates the production of TetR and GFP to bring them to the high state, and represses LacI to slow down its production to allow its degradation to reduce its molecule count. When the input IPTG is removed at time 10,000 making both inputs absent, the outputs retain their current states. At time 15,000, applying inducer aTc causes the circuit to switch output states again. Removing aTc at time 20,000, once again leaves the outputs to hold their states. It should be noted that this figure shows the average output responses of 100 simulation runs, as an individual run may fail to exhibit meaningful logical behavior due to the noise in the circuit. This chapter aims to efficiently determine the probability of erroneous behavior induced by the inherent noisy nature of genetic circuits.

12.5 State-Space Approximation and Analysis

Algorithms 1–3 describe the state-space approximation procedures for a given SCK model $\mathcal{M} = \langle \mathbf{S}, \mathbf{R}, \mathbf{x}_0 \rangle$ with reaction-based abstractions. Note that models with reaction-based abstractions utilize quasi-steady-state approximations [37], where extremely fast reactions are approximated as parameters on propensity functions to prevent starvation of other slower reactions during stochastic analysis. The presented state-space approximation method assumes that probability mass is distributed on a finite and relatively small number of states, and the probability mass does not distribute uniformly as time progresses.

With a given SCK model $\mathcal{M} = \langle \mathbf{S}, \mathbf{R}, \mathbf{x}_0 \rangle$, state-space generation starts by assigning the sole initial state $\mathbf{x}_0$ a 1 to the termination indicator $\hat{\kappa}$, as shown in Algorithm 1. The termination indicator is a function $\hat{\kappa} : \mathbf{X} \to \mathbb{R}^+$, which indicates whether state search should terminate from a state onwards. The initial state graph $\mathcal{G}^0$ includes the initial state $\mathbf{x}_0$ as its set of states. The subsequent state graphs are then constructed and refined by Algorithm 2. In general, both state graphs $\mathcal{G}^{k-1}$ and $\mathcal{G}^k$ are constructed based on the same SCK model $\mathcal{M}$ and refined from the same initial state $\mathbf{x}_0$. The difference is that $\mathcal{G}^k$ refines $\hat{\kappa}$ values for some explored states in $\mathcal{G}^{k-1}$, which may expand $\mathcal{G}^{k-1}$ to include new states in $\mathcal{G}^k$. This process of expansion and refinement is repeated until the size of the approximate state graph stabilizes, at which point an absorbing state is added to this state graph by Algorithm 3. Algorithm 1 terminates by returning the approximated state graph $\mathcal{G}^k$.

Algorithm 1: Construction of approximate state space.

Input: An SCK model $\mathcal{M} = \langle \mathbf{S}, \mathbf{R}, \mathbf{x}_0 \rangle$.
Output: Approximated state graph $\mathcal{G}^k = \langle \mathbf{X}^k, \delta^k, \mathbf{x}_0 \rangle$.

1 $\mathcal{G}^0 = \langle \mathbf{X}^0, \delta^0, \mathbf{x}_0 \rangle$, where $\mathbf{X}^0 = \{\mathbf{x}_0\}$, $\delta^0 = \emptyset$;
2 $\hat{\kappa}(\mathbf{x}_0) := 1$, $\hat{\gamma}(\mathbf{x}_0) := 0$;
3 $k := 0$;
4 **repeat**
5 $\quad k := k + 1$;
6 $\quad$ Construct finite state graph $\mathcal{G}^k = \langle \mathbf{X}^k, \delta^k, \mathbf{x}_0 \rangle$ for $\mathcal{M}$ using Algorithm 2.
7 **until** $|\mathcal{G}^k| = |\mathcal{G}^{k-1}|$;
8 Update $\mathcal{G}^k$ by adding an an extra absorbing state $\mathbf{x}_{abs}$ using Algorithm 3.

Algorithm 2 constructs the approximate finite-state space based on a user-defined termination indicator $\varkappa$. Starting with the initial state $\mathbf{x}_0$, all possible reactions are scheduled to be explored (line 6). For each such reaction R_i, its updated state $\mathbf{x}'$ is obtained by adding the state-change vector v_i specified by R_i to the current state $\mathbf{x}$ (line 7). It should be noted that since the state search in each iteration k begins at the same initial state $\mathbf{x}_0$, $\mathbf{x}'$ may not necessarily be a new state after this step. The termination indicator value at the current state $\mathbf{x}$ is then compared against $\varkappa$ to determine if state exploration should continue (line 11). If the former is lower, then $\mathbf{x}$ becomes a (partially) terminal state, whose state expansion only includes its outgoing transitions leading to existing states in the current state set $\mathbf{X}^k$, but omits transitions leading to states not in $\mathbf{X}^k$. Therefore, if $\mathbf{x}'$ already exists in $\mathbf{X}^k$ (line 12), the algorithm includes the new state-transition relation $(\mathbf{x}, R_i, \mathbf{x}')$ and updates its termination indicator (lines 13–15). For every state $\mathbf{x}'$ to be updated, its predecessor set is constructed (line 14). Each element of this set is a pair of the predecessor state $\mathbf{x}$ and the reaction index i, in which a unique existing state transition $(\mathbf{x}, R_i, \mathbf{x}')$ defines reachability of $\mathbf{x}'$ from $\mathbf{x}$ through reaction R_i. Then the updated termination indicator $\hat{\gamma}(\mathbf{x}')$ is determined by line 15. It should be noted that the updated termination indicator $\hat{\gamma}$ is not used to update termination indicator values for other states explored in the current iteration k, and only becomes available at the end of the current iteration, at which point it is assigned to the current termination indicator $\hat{\kappa}$ (line 26). For each predecessor state $\mathbf{x}$ of $\mathbf{x}'$, its contribution to $\hat{\gamma}(\mathbf{x}')$ is the product of its current state termination value $\hat{\kappa}(\mathbf{x})$ and the probability of transitioning from $\mathbf{x}$ to $\mathbf{x}'$, defined as the ratio of propensity α_i, evaluated at $\mathbf{x}$, to the sum of all propensities at this state. Intuitively, $\hat{\gamma}(\mathbf{x}')$ accumulates path probabilities from all of its predecessor states that have been explored in iteration k. On line 16, the function `explored`$(\mathbf{x}', k)$ checks whether state $\mathbf{x}'$ has been either expanded or updated at the current iteration k. This state can be a state discovered at the current iteration or any of the previous iterations. This state is only scheduled to be explored, if it has not been explored yet in the current iteration. For the case where $\hat{\kappa}(\mathbf{x}) \geqslant \varkappa$, in addition to updating the state-transition relation and termination indicator for $\mathbf{x}'$, the algorithm includes it in the state set $\mathbf{X}^k$ (line 22). This is because $\mathbf{x}$ cannot be the terminal state due

to its large termination indicator value, and therefore its successor $\mathbf{x}'$ becomes the potential candidate for a terminal state. This state is then scheduled for exploration, if the current iteration has not explored it.

The termination indicator update is performed every time a new incoming path is added to a state. It is crucial to have frequent updates since a new incoming path can add its probability contribution to the state, potentially bringing the termination indicator value above $\varkappa$, which in turn changes a terminal state to be nonterminal. This update, therefore, guarantees to explore a state with many incoming paths whose accumulative probabilities are significant, although each individual one might be low compared to $\varkappa$.

Algorithm 2: State space construction and approximation using breadth-first search.

Input: An approximated global state graph $\mathcal{G}^{k-1} = \langle \mathbf{X}^{k-1}, \delta^{k-1}, \mathbf{x}_0 \rangle$.
Output: Updated state graph $\mathcal{G}^{k} = \langle \mathbf{X}^{k}, \delta^{k}, \mathbf{x}_0 \rangle$.

$\mathbf{X}^k := \mathbf{X}^{k-1}$;
$\delta^k := \delta^{k-1}$;
Enqueue$(queue, \mathbf{x}_0)$;
while $queue \neq \emptyset$ **do**
  $\mathbf{x} :=$ *Dequeue*$(queue)$;
  forall the $i \in \{j \mid \alpha_j(\mathbf{x}) > 0\}$ **do**
    Determine the state after reaction R_i: $\mathbf{x}' := \mathbf{x} + v_i$;
    if $\mathbf{x}' \notin \mathbf{X}^k$ **then**
      $\hat{\kappa}(\mathbf{x}') := 0$;
      $\hat{\gamma}(\mathbf{x}') := 0$;
    if $\hat{\kappa}(\mathbf{x}) < \varkappa$ **then**
      if $\mathbf{x}' \in \mathbf{X}^k$ **then**
        $\delta^k := \delta^k \cup \{(\mathbf{x}, R_i, \mathbf{x}')\}$;
        $\texttt{Pre}(\mathbf{x}') := \{(\mathbf{x}, i) \mid (\mathbf{x}, R_i, \mathbf{x}') \in \delta^k, \forall i \in (1, \ldots, m)\}$;
        $\hat{\gamma}(\mathbf{x}') := \sum_{(\mathbf{x},i)\in \texttt{Pre}(\mathbf{x}')} \left(\hat{\kappa}(\mathbf{x}) \cdot \frac{\alpha_i(\mathbf{x})}{\sum_{j=1}^{m} \alpha_j(\mathbf{x})} \right)$;
        if $\neg$*explored*$(\mathbf{x}', k)$ **then**
          Enqueue$(queue, \mathbf{x}')$;
      else
        $\delta^k := \delta^k \cup \{(\mathbf{x}, R_i, \mathbf{x}')\}$;
        $\texttt{Pre}(\mathbf{x}') := \{(\mathbf{x}, i) \mid (\mathbf{x}, R_i, \mathbf{x}') \in \delta^k, \forall i \in (1, \ldots, m)\}$;
        $\hat{\gamma}(\mathbf{x}') := \sum_{(\mathbf{x},i)\in \texttt{Pre}(\mathbf{x}')} \left(\hat{\kappa}(\mathbf{x}) \cdot \frac{\alpha_i(\mathbf{x})}{\sum_{j=1}^{m} \alpha_j(\mathbf{x})} \right)$;
        $\mathbf{X}^k := \mathbf{X}^k \cup \{\mathbf{x}'\}$;
        if $\neg$*explored*$(\mathbf{x}', k)$ **then**
          Enqueue$(queue, \mathbf{x}')$;
forall the $\mathbf{x} \in \mathbf{X}^k$ **do**
  $\hat{\kappa}(\mathbf{x}) := \hat{\gamma}(\mathbf{x})$;

The theoretical state space for the genetic toggle switch described in Sect. 12.4 is infinite. To analyze the model, the state space is truncated based on the value of $\varkappa$.

This truncation, however, leads to probability leakage (i.e., cumulative probabilities of reaching states not included in the explored state space) during the CTMC analysis. To account for probability loss, an absorbing state $\mathbf{x}_{abs}$ is created as the sole successor state for all terminal states on each truncated path, and is added by Algorithm 3 to the state space generated by Algorithm 1. For all states in the global state set, each possible reaction for state $\mathbf{x}$ is checked for exploration. For each reaction R_i, if it has not been explored, its updated state $\mathbf{x}'$ is set to $\mathbf{x}_{abs}$ (line 5). It is obvious that all unexplored transitions from such a terminal state $\mathbf{x}$ lead to the absorbing state.

Algorithm 3: Absorbing state update from approximated global state graph.

Input: An approximated global state graph $\mathcal{G}$.
Output: Updated state graph $\mathcal{G}$ with an absorbing state $\mathbf{x}_{abs}$.

```
1 X := X ∪ {x_abs};
2 forall the x ∈ X do
3 |   forall the i ∈ {j | α_j(x) > 0} do
4 |   |   Determine the state after reaction R_i: x' := x + v_i ;
5 |   |   if (x, R_i, x') ∉ δ then
6 |   |   |   δ := δ ∪ {(x, R_i, x_abs)};
```

The state graph returned by Algorithm 1 is essentially a (sparse) representation of the transition rate matrix. A standard CTMC analysis can be applied directly to it to compute the approximate probability distribution. It should be noted that the termination indicator value for each state is only used to determine terminal states and is omitted for the CTMC analysis.

With the addition of the absorbing state, the CTMC analysis provides a probability bound $[l, u]$, where $0 \leqslant l < u \leqslant 1$, and $(u - l)$ is the probability accumulated in $\mathbf{x}_{abs}$. Assuming the actual probability to satisfy a CSL property ϕ is p, then it holds that $l \leqslant p \leqslant u$. Because the lower bound l does not account for probabilities from paths that, if were not truncated, would feed probabilities back to the explored states, as is the case for calculating p. For the upper bound u, it is always greater or equal to p. Because u includes probabilities accumulated by the absorbing state, of which probabilities from truncated paths that would lead to falsification of ϕ are counted, in addition to probabilities of those leading to the satisfaction of ϕ.

Complexity: The size of generated state-space models depends on the distribution of probability over states and the termination threshold. Therefore, detailed characterization of state-space complexity is challenging. Intuitively, the state-space complexity increases as the termination threshold decreases. This is because a lower termination threshold would allow exploration of states with lower accumulated path probabilities that would otherwise be ignored with a higher termination threshold. Exploration of these states would likely lead to other new states. Moreover, if the majority of probability is distributed over a small number of states, a smaller number of states may be explored compared with a more even probability distribution.

The complexity is highly dependent on the user-provided termination indicator $\varkappa$. Determining a reasonable value of $\varkappa$ can be an iterative process. Initially, $\varkappa$ can be set

to any value that satisfies $0 < \varkappa << 1$, and a state graph and probability bound $[l, u]$ can be generated. The user can then decrease the value of $\varkappa$, if necessary, to tighten the probability bound window. The user can repeat the process until the probability bound returned is guaranteed to prove or disprove the given CSL property.

12.6 Proof of the Termination Condition

The presented algorithms in Sect. 12.5 are guaranteed to terminate under certain conditions. This section provides a description of the termination conditions for each algorithm and presents a proof for termination.

To facilitate the following proof, we first define finite paths of a state graph and depth for breadth-first search. A finite path ρ of a state graph is a sequence $\mathbf{x}_0 \xrightarrow{R_0} \mathbf{x}_1 \xrightarrow{R_1} \ldots \mathbf{x}_{n-1} \xrightarrow{R_{n-1}} \mathbf{x}_n$ such that for every $0 \leqslant i < n$, $(\mathbf{x}_i, R_i, \mathbf{x}_{i+1}) \in \delta$ holds for some R_i. State $\mathbf{x}_n$ is reachable in $\mathcal{G}$ if $\mathbf{x}_n$ is reachable from the initial state through a finite path included in $\mathcal{G}$. Denote the set of all states with depth $\imath$ as ${}^{\imath}\mathbf{X}$. At depth 0, ${}^{0}\mathbf{X} = \{\mathbf{x}_0\}$. At depth $\imath > 0$, ${}^{\imath}\mathbf{X}$ is obtained by collecting all newly created states resulted from the one-step BFS search on all states in ${}^{\imath-1}\mathbf{X}$. Therefore, the depth for a state is determined when it is explored for the first time. Note that ${}^{0}\mathbf{X} \cap {}^{1}\mathbf{X} \cdots {}^{\imath-1}\mathbf{X} \cap {}^{\imath}\mathbf{X} = \emptyset$.

Termination condition for Algorithm 1 requires that, as both the depth $\imath$ and iteration k increase, the sum of termination indicator values for all states of ${}^{\imath}\mathbf{X}^k$ decreases, with possibly finitely many iterations where this sum remains constant. This is formulated by Theorem 1 below.

Theorem 1 (Termination of Algorithm 1) *Algorithm 1 terminates after a finite number of iterations with a given $\varkappa$, where $0 < \varkappa << 1$, if the state graph $\mathcal{G}^{\jmath+1}$ satisfies the following condition: for each depth $\jmath > 0$, there must exist depth $0 \leqslant \imath \leqslant \jmath$ such that $\mathbf{x}_d \xrightarrow{R_h} \mathbf{x}_{d+1} \xrightarrow{R_i} \ldots \mathbf{x}_{d+(m-1)} \xrightarrow{R_l} \mathbf{x}_{d+m}$ is a finite path in $\mathcal{G}^{\jmath+1}$, and $\mathbf{x}_d \in {}^{\imath}\mathbf{X}^{\jmath+1}$, $\mathbf{x}_{d+(m-1)} \in {}^{\jmath}\mathbf{X}^{\jmath+1}$, $\mathbf{x}_{d+m} \in {}^{0}\mathbf{X}^{\jmath+1} \cup {}^{1}\mathbf{X}^{\jmath+1} \cup \cdots {}^{\jmath-1}\mathbf{X}^{\jmath+1} \cup {}^{\jmath}\mathbf{X}^{\jmath+1}$, and $m \in \mathbb{Z}_{\geqslant 0}$.*

Proof Initially, $\mathbf{X}^0 = \{\mathbf{x}_0\}$ and $\hat{\kappa}(\mathbf{x}_0) = 1$. At iteration $k = 1$, during the construction of $\mathcal{G}^1$ (line 6 of Algorithm 1), each state at depth 1, ${}^{1}\mathbf{x} \in {}^{1}\mathbf{X}^1$, is discovered for the first time when $\mathbf{x}_0$ is explored (line 6–24 of Algorithm 2). Therefore, the current termination indicator $\hat{\kappa}({}^{1}\mathbf{x})$ is assigned a 0 (line 9 of Algorithm 2), but its next termination indicator $\hat{\gamma}({}^{1}\mathbf{x})$ gets updated by $\hat{\kappa}(\mathbf{x}_0)$, so that $0 < \hat{\gamma}({}^{1}\mathbf{x}) \leqslant 1$. Each new state ${}^{2}\mathbf{x} \in {}^{2}\mathbf{X}^1$ generated from ${}^{1}\mathbf{X}^1$, is ignored, since $\hat{\kappa}({}^{1}\mathbf{x}) = 0$, which is less than $\varkappa$, and ${}^{2}\mathbf{x} \notin \mathbf{X}^1$ (line 11–16 of Algorithm 2). Then at iteration $k = 2$, the sum of termination indicators is ${}^{1}\zeta^2 = \sum_{\mathbf{x} \in {}^{1}\mathbf{X}^2} \hat{\kappa}(\mathbf{x})$, where each $\hat{\kappa}(\mathbf{x})$ is a fraction of ${}^{0}\zeta^1$, and ${}^{0}\zeta^1 = \hat{\kappa}(\mathbf{x}_0) = 1$. Therefore, ${}^{1}\zeta^2$ is solely contributed from ${}^{0}\zeta^1$. If a self-loop transition $\{\mathbf{x}_0, R_0, \mathbf{x}_0\}$ exists, then ${}^{0}\zeta^1 > {}^{1}\zeta^2$; otherwise ${}^{0}\zeta^1 = {}^{1}\zeta^2$. Therefore, ${}^{0}\zeta^1 \geqslant {}^{1}\zeta^2$. Similar to the previous iteration, the updated $\hat{\gamma}({}^{2}\mathbf{x})$ will be used in the next iteration.

In general, state set ${}^{\iota}\mathbf{X}$ at depth ι is first obtained in iteration ι by collecting all the new states, i.e., states whose depth has not been determined, which are expanded from states in ${}^{\iota-1}\mathbf{X}$. The sum of all termination indicator values for states in ${}^{\iota}\mathbf{X}$ is calculated at iteration $\iota+1$ by either line 15 or 21 of Algorithm 2. To differentiate the termination indicator function $\hat{\kappa}$ in different iterations, we denote $\hat{\kappa}^{\iota}(\mathbf{x})$ as the termination indicator value for state $\mathbf{x}$ at iteration ι. The sum of all termination indicators at iteration $\iota+1$ is computed as follows:

$$
\begin{aligned}
{}^{\iota}\zeta^{\iota+1} &= \sum_{\mathbf{x}' \in {}^{\iota}\mathbf{X}^{\iota+1}} \hat{\kappa}^{\iota+1}(\mathbf{x}') \\
&= \sum_{\mathbf{x}' \in {}^{\iota}\mathbf{X}^{\iota+1}} \sum_{(\mathbf{x},i) \in \mathtt{Pre}(\mathbf{x}')} \left(\hat{\kappa}^{\iota}(\mathbf{x}) \cdot \frac{\alpha_i(\mathbf{x})}{\sum_{j=1}^{m} \alpha_j(\mathbf{x})} \right)
\end{aligned}
$$

If $\bigcup_{\mathbf{x}' \in {}^{\iota}\mathbf{X}^{\iota+1}} \mathtt{Pre}(\mathbf{x}')$ is equal to all transition firings of every state in ${}^{\iota-1}\mathbf{X}^{\iota}$, then termination indicator values for all the states at depth $\iota-1$ are passed to depth ι, and hence ${}^{\iota}\zeta^{\iota+1} = {}^{\iota-1}\zeta^{\iota}$. On the other hand, if there exists one or more transition firings from ${}^{\iota-1}\mathbf{X}^{\iota}$ to depth other than ι, then ${}^{\iota}\zeta^{\iota+1} < {}^{\iota-1}\zeta^{\iota}$. Therefore, ${}^{\iota-1}\zeta^{\iota} \geqslant {}^{\iota}\zeta^{\iota+1}$.

We can, therefore, establish the following conclusion:

$$
1 = {}^{0}\zeta^{1} \geqslant {}^{1}\zeta^{2} \geqslant \cdots {}^{\iota-1}\zeta^{\iota} \geqslant {}^{\iota}\zeta^{\iota+1} \cdots {}^{\jmath-1}\zeta^{\jmath} \geqslant {}^{\jmath}\zeta^{\jmath+1}
$$

From the termination condition stated in Theorem 1, the slowest termination scenario, i.e., the maximal number of iterations required to terminate Algorithm 1, is the following:

$$
1 = {}^{0}\zeta^{1} = {}^{1}\zeta^{2} = \cdots = {}^{\iota}\zeta^{\iota+1} = \cdots = {}^{\jmath-1}\zeta^{\jmath} > {}^{\jmath}\zeta^{\jmath+1}
$$

The inequality ${}^{\jmath-1}\zeta^{\jmath} > {}^{\jmath}\zeta^{\jmath+1}$ holds only if at least one state in ${}^{\jmath-1}\mathbf{X}^{\jmath}$ executes a transition leading to a state in ${}^{0}\mathbf{X}^{\jmath+1} \cup {}^{1}\mathbf{X}^{\jmath+1} \cup \cdots {}^{\jmath-1}\mathbf{X}^{\jmath+1}$, but not in ${}^{\jmath}\mathbf{X}^{\jmath+1}$. State $\mathbf{x}_{d+m}$ in Theorem 1 is such a state. Additionally, the termination condition requires that at least ${}^{\iota}\zeta^{\iota+1} = \cdots = {}^{\jmath-1}\zeta^{\jmath} > {}^{\jmath}\zeta^{\jmath+1}$ holds for *every* depth $\jmath$. This requirement guarantees that the sum of termination indicator values keeps decreasing, with possibly many (or zero) iterations where this sum remains unchanged. Therefore, after a finite number ξ of iterations, ${}^{\xi-1}\zeta^{\xi} < \varkappa$. Since ${}^{\xi-1}\zeta^{\xi}$ is the sum of all individual termination indicator values, in the next iteration $(\xi+1)$, termination indicator $\hat{\kappa}({}^{\xi}\mathbf{x})$ is less than $\varkappa$ for all states in ${}^{\xi}\mathbf{X}^{\xi+1}$, and they become terminal states. Hence, $|\mathcal{G}^{\xi}| = |\mathcal{G}^{\xi+1}|$.

Finally, Algorithm 3 terminates, provided its input state graph generated by Algorithm 2 is finite. This has been proven true, and hence Algorithm 3 always terminates. Therefore, Theorem 1 is true. □

12.7 Results

Algorithms 1, 2, and 3 were implemented in Java as a prototype tool within the `iBioSim` *genetic design automation* (GDA) tool [30, 36, 39]. This tool constructs an approximate CTMC transition rate matrix, which is then analyzed with the help of the PRISM probabilistic model checking tool [27]. Experiments were performed on a 3.2 GHz AMD Debian Linux PC with six cores and 64 GB of RAM. The presented CTMC approximation method was evaluated on several CSL properties for the genetic toggle switch described in Sect. 12.4. This method is then applied to several benchmark examples, and the results are compared with those generated by the STAR tool.

12.7.1 Toggle Switch

An important metric for a toggle switch circuit is the response time. In the first set of experiments, the goal is to determine the genetic toggle switch's response time (i.e., the time it takes to switch from the OFF state to the ON state). The initial OFF state for the toggle switch has 60 LacI, 0 TetR, and 100 IPTG molecules, representing the circuit has just received the set input to switch to the ON state. It should be noted that the input value of 100 molecules is chosen to ensure that the circuit should switch to the ON state, but any moderately large value of input could be used as IPTG is represented as a boundary condition species which means that its molecule count is treated as non-depleting and that it is not consumed by any reactions that it occurs in. The CSL property, $\mathtt{F}(t \leqslant 2100, \text{LacI} < 20 \wedge \text{TetR} > 40)$, describes the probability of the circuit eventually switching to the ON state within a cell cycle of 2, 100 s (an approximation of the cell cycle in *E. coli* [40]). The ON state is characterized by LacI dropping below 20 and TetR rising above 40 molecules.

The termination indicator values are set to 10^{-5}, 10^{-6}, 10^{-7}, and 10^{-9}. Approximate state-space generation and CTMC analysis are performed for each such value. In addition, intermediate verification results are generated on a time course from 0 to 2,100 s with the increment of 100 s. To measure the accuracy of the presented state-space approximations with different termination indicator values, a reference finite-state SCK model is created allowing both LacI and TetR to reach the upper bound of 300 molecules each, which is significantly higher than the upper bounds of TetR and LacI for all experiments performed. The reference model, therefore, incurs significantly larger state space with 90,601 states, but provides accurate verification results for comparison.

Both the accuracy and performance results for the response rate verification are presented in Table 12.2. The column "$|\mathcal{G}|$" lists results for the approximate state graph size used for the CTMC analysis, respectively. The column "ϵ" reports the difference between minimum ($\mathbf{P_{min}}$) and maximum ($\mathbf{P_{max}}$) final response rate probability, which can be taken as the uncertainty window. The columns labeled $T_{\mathbf{P_{min}}}$ and $T_{\mathbf{P_{max}}}$ provide

Table 12.2 Genetic toggle switch response rate results

$\varkappa$	$\lvert\mathcal{G}\rvert$	$\mathbf{P_{min}}$	$\mathbf{P_{max}}$	ϵ	$T_{\mathbf{P_{min}}}$	$T_{\mathbf{P_{max}}}$
ref	90601	0.991789007	–	–	23.49	–
10^{-5}	6171	0.990640972	0.991838990	1.20×10^{-3}	0.492	0.499
10^{-6}	7394	0.991705919	0.991794344	8.84×10^{-5}	0.714	0.629
10^{-7}	8623	0.991781737	0.991789578	7.84×10^{-6}	0.811	0.809
10^{-9}	11394	0.991788952	0.991789012	5.98×10^{-8}	1.161	1.152

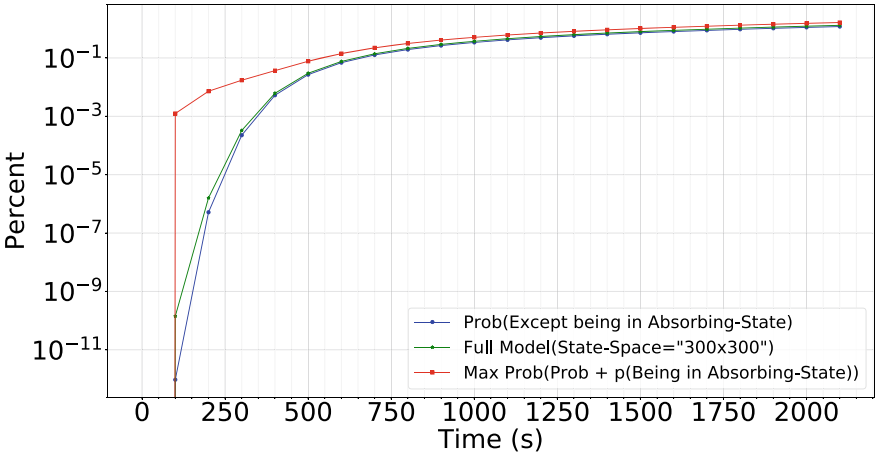
the CTMC analysis time taken by the PRISM tool to calculate the minimum and maximum probability value, respectively.

As the table shows, reducing the termination indicator value improves the accuracy of the final probability, at the price of increased performance cost. Furthermore, the final probability for $t \leqslant 2100$ of the reference model switching its state lies between the window of minimum and maximum probability for all approximate models obtained for different values of $\varkappa$. As we decrease the value for $\varkappa$, from 10^{-5} to 10^{-9}, the error window becomes narrower from 1.20×10^{-3} to 5.98×10^{-8}. The reference model has the final probability of 0.991789007. With $\varkappa = 10^{-5}$, its final probability already produces very accurate final probability with significantly smaller performance cost. The approximate model only explores 6333 states, compared to 90,601 states from the reference model, but it produces the minimum result 0.990640972 and maintains the error bound 1.20×10^{-3}. The runtime for the CTMC analysis on the reference model is 23.49 s, much longer than the runtime for analyzing the approximate CTMC. As an additional comparison, the same toggle switch model built with predetermined thresholds of molecule counts for LacI and TetR in [31] produces a state graph of 70 states, and CTMC analysis with the same initial condition and CSL property reports a final probability of 98.7 percent. The significantly smaller state space is a direct result of predetermined thresholds, which requires prior knowledge of the circuit behavior to determine. The presented state approximation method does not require threshold determination from the user, and it achieves more accurate final probability at a slightly increased performance cost, compared to [31].

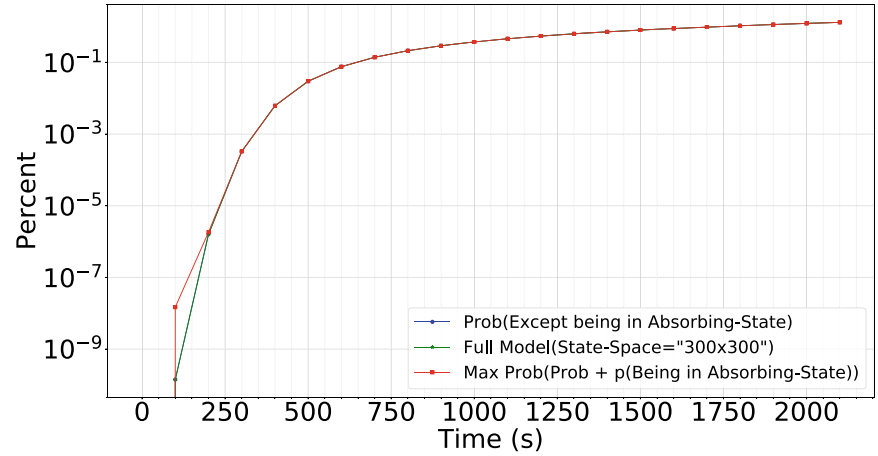
The second set of experiments involves computing the probability that the circuit changes state erroneously within a cell cycle of 2,100 s. This behavior occurs if production of LacI erroneously and significantly inhibits TetR's production to let TetR degrade away and consequently switch state. The toggle switch is initialized to a starting state with 60 LacI molecules, and 0 molecules for all other species. The same CSL properties are verified, and the results are summarized in Table 12.3. Similar to the above experiment, the final probability for $t \leqslant 2100$ of the reference model erroneously changing its state lies between the window of minimum and maximum probability for all approximate models obtained for different values of $\varkappa$. Decreasing the value for $\varkappa$ from 10^{-5} to 10^{-9} decreases the error window from 4.46×10^{-3} to 1.73×10^{-7}. Figure 12.4b shows the time-series plot for the genetic toggle switch failure rates with $\varkappa = 10^{-5}$ and $\varkappa = 10^{-9}$.

Table 12.3 Genetic toggle switch failure rate results

$\varkappa$	$\lvert\mathcal{G}\rvert$	$\mathbf{P_{min}}$	$\mathbf{P_{max}}$	ϵ	$T_{\mathbf{P_{min}}}$	$T_{\mathbf{P_{max}}}$
ref	90601	0.013098589	–	–	25.041	–
10^{-5}	2703	0.011892475	0.016356430	4.46×10^{-3}	0.239	0.241
10^{-6}	3489	0.013076325	0.013578975	5.03×10^{-4}	0.287	0.285
10^{-7}	4306	0.013097869	0.013166728	6.89×10^{-5}	0.361	0.358
10^{-9}	6697	0.013098588	0.013098761	1.73×10^{-7}	0.560	0.566



(a) Genetic toggle switch failure rate with $\varkappa = 10^{-5}$



(b) Genetic toggle switch failure rate with $\varkappa = 10^{-9}$

Fig. 12.4 Error window comparison for different values of $\varkappa$

Table 12.4 State probability comparison for birth–death model with $\varkappa = 10^{-9}$

t	ϵ_{max}	$T_{\texttt{iBioSim}}$	$T_{\texttt{PRISM}}$	$T_{\texttt{STAR}}$
10	1.34×10^{-8}	0.06	0.304	0.22
20	8.75×10^{-8}	0.06	0.311	0.34
30	5.84×10^{-7}	0.06	0.303	0.46
40	1.40×10^{-6}	0.06	0.307	0.59
50	2.84×10^{-6}	0.06	0.309	0.72

Table 12.5 State probability comparison for switching rate experiment of toggle switch model with $\varkappa = 10^{-9}$

t	ϵ_{max}	$T_{\texttt{iBioSim}}$	$T_{\texttt{PRISM}}$	$T_{\texttt{STAR}}$
400	1.84×10^{-9}	18.86	4.44	7.24
800	2.18×10^{-9}	18.86	4.51	17.90
1200	1.91×10^{-8}	18.86	4.59	29.70
1600	5.50×10^{-8}	18.86	4.69	40.65
2000	1.02×10^{-7}	18.86	4.79	50.35

12.7.2 Comparisons with the STAR Tool

To illustrate the accuracy and efficiency of the presented method, we compared the probability distribution results with the STAR tool for the birth–death model and the presented toggle switch model. Table 12.4 summarizes the comparison for a simple birth–death model, whose birth rate is 1 and death rate is 0.1. Column "t" shows the time point at which the state probability is computed, and column labeled ϵ_{max} shows the maximum absolute probability difference for the same individual state obtained from the two tools, among all explored states. Columns labeled $T_{\texttt{iBioSim}}$ and $T_{\texttt{PRISM}}$ list runtimes in seconds to generate the state space in `iBioSim` and to analyze the model in PRISM for each given time point, respectively. Column $T_{\texttt{STAR}}$ lists the runtime reported by STAR. The maximum probability difference reaches its peak value of 2.84×10^{-6} at time point $t = 50$. All other time points show significantly smaller errors. The runtime to analyze the model in `iBioSim` and PRISM is less than a second as the generated state space is only 28 states. The STAR tool also reports a similar runtime.

Table 12.5 shows a comparison of results for the aforementioned toggle switch. Our proposed method produces accurate results compared to those from the STAR tool, as is indicated by the maximal probability difference (ϵ_{max}). Columns $T_{\texttt{iBioSim}}$ and $T_{\texttt{PRISM}}$ list runtimes in seconds to generate the state space in `iBioSim` and to analyze the model in PRISM for each given time point, respectively. Column labeled $T_{\texttt{STAR}}$ lists the runtime reported by STAR. The combined runtime to generate the state space and analyze the model for our method is less than 24 s for different time points and remains almost constant as the time point t increases. The STAR tool

reports shorter runtime for smaller t but linear increase in runtime as the time point value gets larger.

12.8 Conclusion

This chapter presents a method that builds an approximate state space of genetic circuit models to analyze infinite-state continuous-time Markov chains. This approximation method iteratively expands from the initial state using a breadth-first search, computes and updates the termination indicator value for each state on the fly, based on the cumulative path probabilities for all incoming transitions to a state. The probability of each path segment is the ratio of the propensity of a reaction to the sum of all propensities for any given state. Our state-space approximation is determined by comparing the state termination indicator to a user-provided termination threshold and only exploring states with a significant termination indicator value. This method is capable of computing the approximate state space with no prior knowledge and is completely automated.

For future work, we plan to improve and optimize probability approximation for re-convergent paths that close cycles during the state exploration in order to achieve potentially faster termination of the state search. We will consider different approaches to determining the termination indicator value automatically from the CSL property being analyzed. Additionally, we plan to explore augmenting our technique with bi-simulation minimization and abstraction to further minimize the generated state space and better allow for scalability. To improve performance of tool implementation, we plan to investigate tighter integration with the PRISM tool.

Acknowledgements The authors thank Verena Wolf for providing benchmarks and the STAR tool. The authors would also like to thank Dave Parker and Joachim Klein for providing assistance in interfacing with the PRISM tool. We also thank the reviewers for their feedback on an earlier version of this paper. Chris Myers is supported by the National Science Foundation under Grant No., CCF-1218095 and No., CCF-1748200. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

1. Abate A, Brim L, Češka M, Kwiatkowska M (2015) Adaptive aggregation of Markov chains: quantitative analysis of chemical reaction networks. In: Kroening D, Păsăreanu CS (eds) Computer aided verification. Springer International Publishing, Cham, pp 195–213
2. Aziz A, Sanwal K, Singhal V, Brayton R (2000) Model-checking continuous-time Markov chains. ACM Trans Comput Logic 1:162–170
3. Baier C, Größer M, Ciesinski F (2004) Partial order reduction for probabilistic systems. In: 1st international conference on quantitative evaluation of systems (QEST 2004), Enschede, The Netherlands, 27–30 September 2004. IEEE Computer Society, pp 230–239. https://doi.org/10.1109/QEST.2004.1348037

4. Baier C, D'Argenio P, Groesser M (2006) Partial order reduction for probabilistic branching time. Electron Notes Theor Comput Sci 153(2):97–116. https://doi.org/10.1016/j.entcs.2005.10.034
5. Burrage K, Hegland M, Macnamara S, 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: Langville AN, Stewart WJ (eds) MAM 2006: Markov anniversary meeting: an international conference to celebrate the 150th anniversary of the birth of A.A. Markov. Boston Books, Charleston, South Carolina, pp 21–38. http://eprints.qut.edu.au/46148/
6. Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst 8(2):244–263. https://doi.org/10.1145/5397.5399
7. Courcoubetis C, Yannakakis M (1988) Verifying temporal properties of finite state probabilistic programs. In: Proceedings of the 29th annual symposium on foundations of computer science (FOCS'88). IEEE Computer Society Press, pp 338–345
8. Courcoubetis C, Yannakakis M (1995) The complexity of probabilistic verification. J ACM 42(4):857–907
9. Díaz ÁF, Baier C, Earle CB, Fredlund L (2012) Static partial order reduction for probabilistic concurrent systems. In: Ninth international conference on quantitative evaluation of systems, QEST 2012, London, United Kingdom, 17–20 September 2012. IEEE Computer Society, pp 104–113. https://doi.org/10.1109/QEST.2012.22
10. Fecher H, Leucker M, Wolf V (2006) Don't know in probabilistic systems. In: Proceedings of the 13th international conference on model checking software. Springer-Verlag, Berlin, SPIN'06, pp 71–88. https://doi.org/10.1007/11691617_5
11. Fisler K, Vardi MY (1998) Bisimulation minimization in an automata-theoretic verification framework. In: FMCAD. Lecture notes in computer science, vol 1522. Springer, Berlin, pp 115–132
12. Fisler K, Vardi MY (1999) Bisimulation and model checking. In: Pierre L, Kropf T (eds) Correct hardware design and verification methods. Springer, Berlin, pp 338–342
13. Fisler K, Vardi MY (2002) Bisimulation minimization and symbolic model checking. Form Methods Syst Des 21(1):39–78. https://doi.org/10.1023/A:1016091902809
14. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in Escherichia coli. Nature 403:339–342
15. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Chem Phys 81(25):2340–2361
16. Grassmann W (1977) Transient solutions in Markovian queueing systems. Comput Oper Res 4(1):47–53. https://doi.org/10.1016/0305-0548(77)90007-7, http://www.sciencedirect.com/science/article/pii/0305054877900077
17. Gross D, Miller DR (1984) The randomization technique as a modeling tool and solution procedure for transient Markov processes. Oper Res 32(2):343–361. https://doi.org/10.1287/opre.32.2.343
18. Hansson H, Jonsson B (1994) A logic for reasoning about time and reliability. Form Asp Comput 6(5):512–535
19. Heath J, Kwiatkowska M, Norman G, Parker D, Tymchyshyn O (2006) Probabilistic model checking of complex biological pathways. In: Priami C (ed) Computational methods in systems biology. Springer, Berlin, pp 32–47
20. Henzinger TA, Mateescu M, Wolf V (2009) Sliding window abstraction for infinite Markov chains. In: Bouajjani A, Maler O (eds) Computer aided verification. Springer, Berlin, pp 337–352
21. Hermanns H, Wachter B, Zhang L (2008) Probabilistic CEGAR. In: Proceedings of the 20th international conference on computer aided verification. Springer-Verlag, Berlin, CAV '08, pp 162–175. https://doi.org/10.1007/978-3-540-70545-1_16
22. Katoen JP, Kemna T, Zapreev I, Jansen DN (2007a) Bisimulation minimisation mostly speeds up probabilistic model checking. In: Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems. Springer-Verlag, Berlin, TACAS'07, pp 87–101. http://dl.acm.org/citation.cfm?id=1763507.1763519

23. Katoen JP, Klink D, Leucker M, Wolf V (2007b) Three-valued abstraction for continuous-time Markov chains. In: Proceedings of the 19th international conference on computer aided verification. Springer-Verlag, Berlin, CAV'07, pp 311–324. http://dl.acm.org/citation.cfm?id=1770351.1770401
24. Kuwahara H, Myers C, Barker N, Samoilov M, Arkin A (2006) Automated abstraction methodology for genetic regulatory networks. Trans Comp Syst Biol VI:150–175
25. Kwiatkowska M, Norman G, Parker D (2007) Stochastic model checking. In: Bernardo M, Hillston J (eds) Formal methods for the design of computer, communication and software systems: performance evaluation (SFM'07). LNCS (tutorial volume), vol 4486. Springer, Berlin, pp 220–270
26. Kwiatkowska M, Norman G, Parker D, Qu H (2010) Assume-guarantee verification for probabilistic systems. In: Esparza J, Majumdar R (eds) Tools and algorithms for the construction and analysis of systems, pp 23–37
27. Kwiatkowska M, Norman G, Parker D (2011) PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan G, Qadeer S (eds) Proceedings of the 23rd international conference on computer aided verification (CAV'11). LNCS, vol 6806. Springer, Berlin, pp 585–591
28. Lapin M, Mikeev L, Wolf V (2011) Shave: stochastic hybrid analysis of Markov population models. In: Proceedings of the 14th ACM international conference on hybrid systems: computation and control, HSCC 2011, Chicago, IL, USA, 12–14 April 2011. ACM, pp 311–312. https://publications.cispa.saarland/891/, pub_id: 705. Bibtex: LaMiWo_11:Shave. URL date: None
29. Madsen C (2013) Stochastic analysis of synthetic genetic circuits. PhD thesis, University of Utah
30. Madsen C, Myers CJ, Patterson T, Roehner N, Stevens JT, Winstead C (2012) Design and test of genetic circuits using iBioSim. IEEE Des Test Comput 29(3):32–39
31. Madsen C, Zhang Z, Roehner N, Winstead C, Myers C (2014) Stochastic model checking of genetic circuits. J Emerg Technol Comput Syst 11(3):23:1–23:21. https://doi.org/10.1145/2644817
32. Mikeev L, Sandmann W, Wolf V (2013) Numerical approximation of rare event probabilities in biochemically reacting systems. In: Gupta A, Henzinger TA (eds) Computational methods in systems biology. Springer, Berlin, pp 5–18
33. Munsky B, Khammash M (2006) The finite state projection algorithm for the solution of the chemical master equation. J Chem Phys 124(4):044104. https://doi.org/10.1063/1.2145882
34. Munsky B, Khammash M (2008) The finite state projection approach for the analysis of stochastic noise in gene networks. IEEE Trans Autom Control 53(Special Issue):201–214. https://doi.org/10.1109/TAC.2007.911361
35. Myers CJ (2009) Engineering genetic circuits. Chapman & Hall/CRC mathematical and computational biology, 1st edn. Chapman & Hall/CRC, London
36. Myers CJ, Barker N, Jones K, Kuwahara H, Madsen C, Nguyen NPD (2009) iBioSim: a tool for the analysis and design of genetic circuits. Bioinformatics 25(21):2848–2849. https://doi.org/10.1093/bioinformatics/btp457
37. Rao CV, Arkin AP (2003) Stochastic chemical kinetics and the quasi-steady-state assumption: application to the Gillespie algorithm. J Chem Phys 118(11):4999–5010. https://doi.org/10.1063/1.1545446
38. Wachter B, Zhang L, Hermanns H (2007) Probabilistic model checking modulo theories. In: fourth international conference on the quantitative evaluation of systems (QEST 2007), pp 129–140
39. Watanabe L, Nguyen T, Zhang M, Zundel Z, Zhang Z, Madsen C, Roehner N, Myers C (0) ibiosim 3: A tool for model-based genetic circuit design. ACS Synth Biol 0(0):null. https://doi.org/10.1021/acssynbio.8b00078, pMID: 29944839
40. Zheng H, Ho PY, Jiang M, Tang B, Liu W, Li D, Yu X, Kleckner NE, Amir A, Liu C (2016) Interrogating the Escherichia coli cell cycle by cell dimension perturbations. Proc Natl Acad Sci 113(52):15000–15005. https://doi.org/10.1073/pnas.1617932114, http://www.pnas.org/content/113/52/15000, http://www.pnas.org/content/113/52/15000.full.pdf