Probabilistic Model Checking for Biology

Marta KWIATKOWSKA [a,1] and Chris THACHUK [a]
[a] *Department of Computer Science, University of Oxford, Oxford, UK*

Abstract. Probabilistic model checking is an automated method for verifying the correctness and performance of probabilistic models. Property specifications are expressed in probabilistic temporal logic, denoting, for example, the probability of a given event, the probability of its occurrence within a given time interval, or expected number of times it has occurred in a time period. This chapter focuses on the application of probabilistic model checking to biological systems modelled as continuous-time Markov chains, illustrating the usefulness of these techniques through relevant case studies performed with the probabilistic model checker PRISM. We begin with an introduction to discrete-time Markov chains and the corresponding model checking algorithms. Then continuous-time Markov chain models are defined, together with the logic CSL (Continuous Stochastic Logic), and an overview of model checking for CSL is given, which proceeds mainly by reduction to discrete-time Markov chains. The techniques are illustrated with examples of biochemical reaction networks, which are verified against quantitative temporal properties. Next a biological case study analysing the Fibroblast Growth Factor (FGF) molecular signalling pathway is summarised, highlighting how probabilistic model checking can assist in scientific discovery. Finally, we consider DNA computation, and specifically the DSD formalism (DNA Strand Displacement), and show how errors can be detected in DNA gate designs, analogous to model checking for digital circuits.

Keywords. Temporal logic, Model checking, Markov chains, Chemical reaction networks, Biological signalling pathways, DNA computation

1. Introduction

Probabilistic model checking is an automated verification technique for the analysis of systems that exhibit stochastic characteristics. It involves the construction and systematic analysis of a probabilistic model, typically a variant of a Markov chain, against a range of quantitative properties, for example performance or reliability. Such an exhaustive analysis can confirm that the probability of some undesirable event is indeed apprpriately small, or it can reveal anomalies or unusual trends in the quantitative behaviour under different scenarios. Probabilistic model checking, and in particular the probabilistic model checker PRISM [21], has been used to analyse and detect faults in a wide variety of protocols and systems, drawn from distributed systems, wireless protocols, power management, nanotechnology and biology.

In this chapter we describe how probabilistic model checking (also known as stochastic model checking) [19] can be used to study the behaviour of biological sys-

[1] Corresponding Author E-mail: marta.kwiatkowska@cs.ox.ac.uk

tems, where we have concentrated on modelling and analysis of molecular networks. There are two established frameworks for modelling molecular reactions, the continuous deterministic approach and the discrete stochastic approach [14,32]. In the deterministic approach, one approximates the number of molecules using a continuous function that represents the change in molecular concentrations using differential equations (ODEs) based on mass action kinetics. The ODE approach is suitable for modelling average behaviour and assumes large numbers of molecules. The discrete stochastic approach, on the other hand, models the stochastic evolution of populations of molecules, where reactions are discrete events, governed by stochastic rates typically assumed to be constant and dependent on the number of molecules, which admits their modelling in terms of continuous-time Markov chains. This approach is more accurate in cases where the number of molecules are small, since it can capture the situation when the system behaviour becomes non-continuous due to, e.g., molecules degrading [22]. Conventionally, discrete stochastic models have been analysed using stochastic simulation; here, we focus on the complementary technique of probabilistic model checking [16], which, in contrast to simulation, is exhaustive and able to discover best- and worst-case scenarios.

We begin this chapter by giving an introduction to probabilistic model checking based on the discrete-time Markov chain models. We model molecular networks as continuous-time Markov chains (CTMCs), in which transitions between states are annotated by real-valued rates, interpreted as the parameters of negative exponential distributions. CTMCs can be additionally annotated by (state or transition) rewards, which can be non-negative real numbers. Quantitative properties will be written in temporal logic CSL (Continuous Stochastic Logic), and can express, e.g., "what is the probability that phosphorylation occurs within 30 minutes?", "what is the expected time until phosphorylation?" and "what is the expected number of phosphorylation reactions before degradation?". Probabilistic model checking, as e.g. implemented in PRISM [21], can be invoked to compute the probability or expectation that the property is satisfied in the model. The computation can be exact, involving numerical algorithms based on uniformisation (essentially a discretisation of the CTMC), or approximate, based on probability estimation of the proportion of simulated trajectories that satisfy the property (known as statistical model checking [33]). We include examples of chemical reaction networks to illustrate the working of numerical model checking.

We then describe two case studies of molecular networks analysed in PRISM, with the aim to highlight the potential that these techniques offer to accelerate the scientific discovery and to become a key component of computer-aided design tools for nanotechnology. The first study of the FGF signalling pathway [16] was modelled directly in PRISM's input language; more information on how to model molecular networks in PRISM can be found [20]. We demonstrate how model checking against quantitative properties can be used to perform *in silico* genetics, and highlight the predictive power of such models with which we were able to identify trends that were later confirmed in wetlab experiments [17]. In the second case study, a DNA transducer was modelled in the DSD (DNA Strand Displacement) tool [26], from which PRISM models were automatically generated for analysis [23]. We show that, analogous to conventional circuit designs, automated verification techniques can be applied to check for correctness and identify flaws in the designs [23].

The case studies discussed in this chapter demonstrate the usefulness of probabilistic model checking techniques in supporting the design, analysis, prediction and debugging

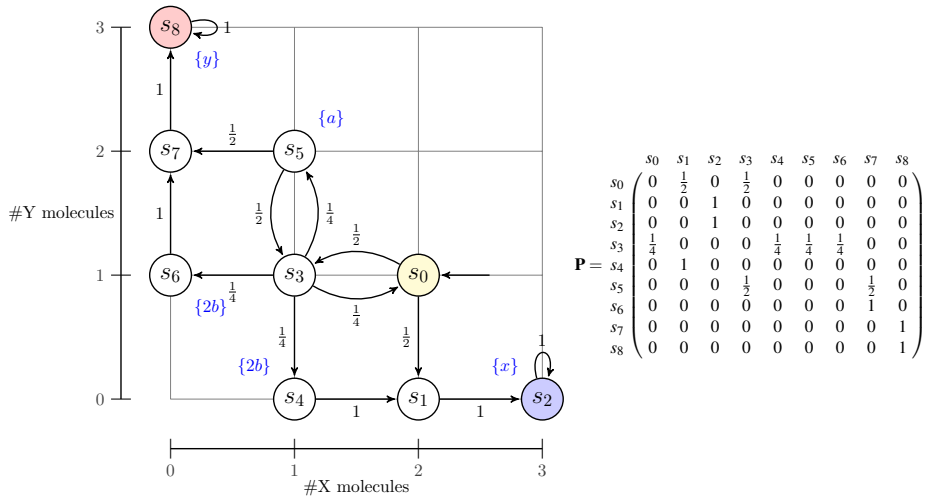


Figure 1. A DTMC and its transition probability matrix **P** for an approximate majority chemical reaction network (CRN) that initially contains 2 molecules of X and 1 molecule of Y.

for molecular-scale processes and devices. DNA computation, in particular, is an exciting new direction likely to benefit from future developments of this field.

2. Model Checking for Discrete-time Markov Chains

In this chapter, continuous-time Markov chains (CTMCs) will be used to model the behaviour of populations of molecules and their interaction via biochemical reactions. As we will see later, probabilistic model checking for CTMCs follows by discretisation, and hence we first introduce the model of discrete-time Markov chains (DTMCs) and the corresponding model checking algorithms. More information about these topics can be found in tutorial papers [19,20].

2.1. Discrete-time Markov Chains

In a DTMC model, discrete probability distributions are used to denote transitions between states, quantifying the likelihood of moving to a given target state.

Definition 1 (Discrete-time Markov chain (DTMC))**.** *A* discrete-time Markov chain (DTMC) *is a tuple* $\mathsf{D} = (S, \bar{s}, \mathbf{P}, L)$*, where is* S *a finite set of states,* $\bar{s} \in S$ *is a distinguished initial state,* $\mathbf{P} : S \times S \to [0,1]$ *is a transition probability matrix such that* $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ *for all* $s \in S$*, and* $L(s) \subseteq AP$ *is labelling with atomic propositions.*

The behaviour of a DTMC is represented by the set of its execution paths $s_0 s_1 s_2 \ldots$ such that $s_0 = \bar{s}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A probability space can be defined over paths of the DTMC [4], where events correspond to measurable sets of paths, for example those reaching an error state. *Probabilistic model checking* then involves computing the probability of a given event, and will be described later. The simpler case of probabilistic reachability refers to the probability of reaching a given set of target states.

Example 1. *A chemical reaction equation, such as $A+B \to C$, describes a process where reactant molecules can interact to create product molecules. In this instance, a molecule A and a molecule B will be consumed to produce a new molecule of C. A chemical reaction network (CRN) is a set of chemical reaction equations and a count of initial molecules. For example, consider a CRN that initially contains 2 molecules of species X, 1 molecule of species Y, and has the following four chemical reaction equations:*

(a) $X+Y \to X+B$ ***(b)*** $X+Y \to B+Y$ ***(c)*** $X+B \to X+X$ ***(d)*** $Y+B \to Y+Y$

*Equations **(a)–(d)** describe an* approximate majority *algorithm [2]. Given some initial quantity of X and Y molecules, the network will converge to one of two consensus states: only X molecules are present, or only Y molecules are present. The consensus that is reached favours whichever species is initially present in a larger quantity (the majority); however, it is possible to reach a consensus which favours the species initially present in a smaller quantity (the minority). In the approximate majority CRN the transitions are taken uniformly at random, and we can model it as a DTMC $\mathsf{D} = (S, \bar{s}, \mathbf{P}, L)$ shown in Figure 1. The states of the DTMC are pairs of molecule counts, respectively for X and Y molecules, and hence when there are initially 2 molecules of X and 1 molecule of Y (i.e., initial state $\bar{s} = s_0$) the DTMC has 9 states, $S = \{s_0, \ldots, s_8\}$. Note that we do not explicitly include the count for the auxiliary species B, and instead use labelling with atomic propositions. The matrix $\mathbf{P}$ gives the probability of transitioning from one state to another. The set of atomic propositions AP is $\{x, y, a, 2b\}$ — x and y denote a consensus of X and Y molecules, respectively — and function L labels s_2 with x and s_8 with y. The state containing 2 Y molecules and 1 X molecule is labelled a and the states containing two B molecules are labelled $2b$.*

2.2. Probabilistic Computation Tree Logic (PCTL)

To reason about DTMCs, we use the temporal logic PCTL (Computation Tree Logic) [15,5], with which one can express probabilistic path-based properties. In common with the logic CTL, PCTL distinguishes between state (Φ) and path (ψ) formulas and includes path operators $\mathrm{X}\Phi$ (next state), $\Phi_1 \mathrm{U} \Phi_2$ (until and its bounded variant $\mathrm{U}^{\leq k}$), as well as the usual derived operators $\mathrm{F}\Phi \equiv \mathit{true}\,\mathrm{U}\,\Phi$ (eventually) and $\mathrm{G}\Phi \equiv \neg\mathrm{F}\neg\Phi$ (always). Instead of the A and E path quantifiers, PCTL introduces the *probabilistic operator* $\mathrm{P}_{\sim p}[\cdot]$

Definition 2 (Probabilistic Computation Tree Logic (PCTL) syntax)**.** *The syntax of PCTL is given by:*

$$\Phi ::= \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathrm{P}_{\sim p}[\psi]$$
$$\psi ::= \mathrm{X}\Phi \mid \Phi \mathrm{U}^{\leq k} \Phi \mid \Phi \mathrm{U} \Phi$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ and $k \in \mathbb{N}$.

PCTL formulas are interpreted over the states of a DTMC. Path formulas can occur only within the scope of the probabilistic operator. The semantics for the PCTL formulas other than the probabilistic operator is the same as for CTL. We say that a state $s \in S$ satisfies the formula $\mathrm{P}_{\sim p}[\psi]$ if the probability of the set of paths from s that satisfy

ψ meets the *probability bound* $\sim p$. We can also use PCTL in *quantitative* form, e.g. $\mathrm{P}_{=?}[\,\psi\,]$, which returns the probability of satisfying ψ.

Example 2. *Given the DTMC of Figure 1, we can formulate a number of queries using PCTL formulas. For example:*

- $\mathrm{P}_{=?}[\mathrm{F}\, y]$ - *"the probability that a consensus of Y molecules is reached"*
- $\mathrm{P}_{=?}[\neg a \mathrm{U}\, x]$ - *"the probability that a consensus of X molecules is reached without passing through the state containing 2 Y molecules and 1 X molecule (state labelled a)"*
- $\mathrm{P}_{=?}[\mathrm{F}\, 2b]$ - *"the probability that a state is reached where B molecules form a strict majority (states are labelled 2b)".*

2.3. Model Checking for PCTL over DTMCs

The PCTL model checking algorithm [15] takes as inputs a labelled DTMC $\mathsf{D} = (S, \bar{s}, \mathbf{P}, L)$ and a PCTL formula Φ. The intuition is that the probability measure of the set of ψ-paths, which is measurable as shown in [30], is calculated and compared to the probability bound, yielding true or false respectively. The algorithm is based on that for CTL [10] and proceeds by bottom-up traversal of the parse tree for Φ, recursively computing the sets $Sat(\Phi') = \{s \in S \,|\, s \models \Phi'\}$ of all states satisfying each subformula Φ'. The algorithm decides if a given state s satisfies Φ by checking if $s \in Sat(\Phi)$.

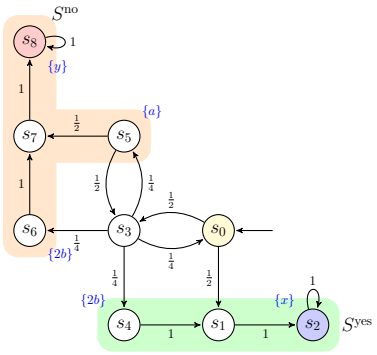
For the non-probabilistic operators, the algorithm works as for CTL and computes: $Sat(true) := S$, $Sat(a) := \{s \in S \mid a \in L(s)$, $Sat(\neg\Phi) := S \setminus Sat(\Phi)$, and $Sat(\Phi_1 \wedge \Phi_2) := Sat(\Phi_1) \cap Sat(\Phi_2)$.

For the probabilistic operator $\mathrm{P}_{\sim p}[\,\psi\,]$, first the probability measure of the set of paths satisfying ψ for all states is computed, and then compared it to the probability bound $\sim p$ before deciding which states to include in the $Sat(\mathrm{P}_{\bowtie p}[\psi])$ set. The probabilities are calculated as follows. For the next state formula $\mathrm{X}\Phi$, representing $Sat(\Phi)$ as a column vector $\underline{\Phi} : S \longrightarrow \{0,1\}$ given by $\underline{\Phi}(s) = 1$ if $s \models \Phi$ and 0 otherwise, we compute the probabilities for all states by a single matrix-by-vector multiplication, $\mathbf{P} \cdot \underline{\Phi}$. For the path formula $\Phi_1 \mathrm{U} \Phi_2$, the probabilities are obtained as the unique solution of the *linear equation system* in variables $\{x_s \mid s \in S\}$:

$$x_s = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot x_{s'} & \text{if } s \in S^{?} \end{cases}$$

where $S^{no} := Sat(\mathrm{P}_{\leq 0}[\Phi_1 \mathrm{U} \Phi_2])$ and $S^{yes} := Sat(\mathrm{P}_{\geq 1}[\Phi_1 \mathrm{U} \Phi_2])$ denote the sets of *all* states that satisfy $\Phi_1 \mathrm{U} \Phi_2$ with probability exactly 0 and 1, respectively, and $S^{?} = S \setminus (S^{no} \cup S^{yes})$. The solution of the resulting linear equation system in $|S^{?}|$ variables can be obtained by any direct method (e.g. Gaussian elimination) or iterative method (e.g. Jacobi, Gauss-Seidel). The bounded until operator $\Phi_1 \mathrm{U}^{\leq k} \Phi_2$ is similar, and computed using recursive equations.

It is worth mentioning that probability 1 and 0 states (so called precomputation) can be implemented by simply using graph traversal, which helps avoid the problem of round-off errors that are typical for numerical computation. For S^{no}, we first compute the set of states from which we can reach, with positive probability, a Φ_2-state passing only



$$
\begin{aligned}
x_1 &= & \mathbf{1} \\
x_2 &= & \mathbf{1} \\
x_4 &= & \mathbf{1} \\
x_5 &= & \mathbf{0} \\
x_6 &= & \mathbf{0} \\
x_7 &= & \mathbf{0} \\
x_8 &= & \mathbf{0} \\
x_0 &= & 0.5x_1 + 0.5x_3 = \tfrac{5}{7} \\
x_3 &= & 0.25x_0 + 0.25x_4 + 0.25x_5 + 0.25x_6 = \tfrac{3}{7}
\end{aligned}
$$

Figure 2. Determining the probabilities $Prob(S, \neg a \mathrm{U} x)$ for the DTMC from Figure 1. The DTMC has been labelled with the S^{yes} and S^{no} sets of states from the precomputation. The linear equation system on the right is used to determine the probabilities for the remaining states (i.e., s_0 and s_3).

through states satisfying Φ_1, and then subtract this set from S. For S^{yes}, we reuse S^{no} and compute the set of states from which we can reach, with positive probability, a S^{no}-state passing only through Φ_1, and then subtract this from S.

Example 3. *From the previous example, consider the query* $\mathrm{P}_{=?}[\neg a \mathrm{U} x]$. *How can this be computed? Figure 2 shows in detail how to calculate this until query for the DTMC of Figure 1. The left side of Figure 2 demonstrates the results of the precomputation. The group of states labelled* S^{no} *are those from which we cannot reach, with positive probability, a x-state passing only through states that do not satisfy a. With respect to the PCTL formula, the probability for each state in* S^{no} *is* $\mathbf{0}$. *The group of states labelled* S^{yes} *correspond to those that, with probability* $\mathbf{1}$, *can reach a state labelled x by passing only through states not labelled with a. After the precomputation, the probabilities for only two states remain unknown (*s_0 *and* s_3*). This results in a system of two linear equations with two unknowns, that could be easily solved using a number of standard methods. From the initial state* s_0, *we find that the probability of eventually reaching the state labelled x, without passing through the state labelled a, is* $\frac{5}{7}$.

2.4. Extending PCTL and DTMCs with Rewards

In order to reason about a broad range of *quantitative properties*, we augment probabilistic models with *reward* (or *cost*) information. For a DTMC D, a *reward structure* $(\underline{\rho}, \iota)$ consists of a vector of state rewards $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ incurred per time unit, together with a matrix $\iota : S \times S \to \mathbb{R}_{\geq 0}$ of transition rewards, incurred each time a transition is taken. For a given a reward structure, we can perform quantitative analysis by computing *expectations* of (instantaneous or cumulative) rewards with respect to the previously defined probability space on paths, for example expected energy usage until termination.

To capture expected rewards, we extend the logic PCTL with the *reward operator* $\mathrm{R}_{=?}[\cdot]$ [19] as follows:

$$\mathrm{R}_{\sim r}[\mathrm{C}^{\leq k}] \mid \mathrm{R}_{\sim r}[\mathrm{F}\, \Phi]$$

where $\sim \in \{<, \leq, \geq, >\}$, $r \in \mathbb{R}_{\geq 0}$, $k \in \mathbb{N}$ and Φ is a PCTL state formula.

Intuitively, a state s satisfies $\mathrm{R}_{\sim r}[\mathtt{C}^{\leq k}]$ (*cumulative* reward) if, from state s, the expected reward cumulated over k time-steps satisfies $\bowtie r$; and $\mathrm{R}_{\bowtie r}[\mathtt{F}\,\Phi]$ (*reachability* reward) is true if, from state s, the expected reward cumulated before a state satisfying Φ is reached meets $\bowtie r$. Formally, the semantics of the reward operator is defined using the expectation of random variables $X_{\mathtt{C}^{\leq k}}$ and$X_{\mathtt{F}\,\phi}$, which are defined, for any path $\omega = s_0 s_1 s_2 \ldots$, as follows:

$$X_{\mathtt{C}^{\leq k}}(\omega) := \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} \underline{\rho}(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

$$X_{\mathtt{F}\,\Phi}(\omega) := \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}.\, s_i \not\models \Phi \\ \sum_{i=0}^{\min\{j | s_j \models \Phi\}-1} \underline{\rho}(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise.} \end{cases}$$

Model checking of the reward operator is similar to computing probabilities for the probabilistic operator, and follows through the solution of recursive equations (for $\mathrm{R}_{\sim r}[\mathtt{C}^{\leq k}]$) or a system of linear equations (for $\mathrm{R}_{\sim r}[\mathtt{F}\,\Phi]$). We can also use the reward operator as a quantitative query, namely $\mathrm{R}_{=?}[\cdot]$, and return the expectation instead. The reward operator can also be used in conjunction with the name of the reward structure, e.g. $\mathrm{R}\{\text{“time”}\}_{=?}[\cdot]$ denotes the expected reward for the reward structure called “time”. For more details on this, and other aspects of probabilistic model checking for DTMCs, see e.g. [19,4].

Example 4. *The previous examples demonstrate the usefulness of PCTL to formulate queries to determine the probability of reaching certain states, or for satisfying certain path formulas. The use of rewards allows us to express the following queries:*

- $\mathrm{R}_{=?}[\mathrm{F}\ (x \vee y)]$ - *“the expected number of reactions before reaching a consensus state”, assuming state rewards of 0 and transition rewards of 1*
- $\mathrm{R}_{=?}[\mathrm{F}\ y]$ - *“the expected number of times the state labelled a is reached before a consensus of Y molecules”, where the transition rewards are 0 and state reward is 1 for each state labelled with a*
- $\mathrm{R}\{\text{“eq”}\}_{=?}[\mathtt{C}^{\leq 100}]$ - *“the expected number of times the system enters a state with equal numbers of X and Y molecules in the first 100 steps”, where “eq” is the reward structure that assigns transition rewards of 0 and state rewards of 1 to states which have equal numbers of X and Y molecules.*

3. Model Checking for Continuous-time Markov Chains

In DTMCs, the progress of time is modelled by discrete time steps, and the model is appropriate for systems which progress in lock-step synchrony. For many applications, it is preferable to use a *continuous* model of time, where the delays between transitions can be arbitrary real values. The classical model of *continuous-time Markov chains (CTMCs)* extends DTMCs with real-time by modelling transition delays with exponential distributions.

3.1. Continuous-time Markov Chains

We define continuous-time Markov chains as follows.

Definition 3 (Continuous-time Markov chain (CTMC))**.** *A* continuous-time Markov chain (CTMC) *is* $\mathsf{C} = (S, \bar{s}, \mathbf{P}, E, AP, L)$ *where:*

- $(S, \bar{s}, \mathbf{P}, AP, L)$ *is a DTMC (called the* embedded DTMC*);*
- $E : S \to \mathbb{R}_{\geq 0}$ *is the* exit rate*.*

In a CTMC C, the residence time of a state $s \in S$ is a random variable governed by an exponential distribution with rate parameter $E(s)$. The rate of an exponential distribution corresponds to the number of times a given event occurs in a unit of time. Therefore, the probability to exit state s in t time units is given by $\int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$. To take the transition from s to another state s' in t time units, the probability equals $\mathbf{P}(s,s') \cdot \int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$, where $\mathbf{P}$ is the embedded DTMC of the CTMC C.

Alternatively, a CTMC can be defined by specifying the *rates matrix* $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$, where $\mathbf{R}(s,s')$ is the rate of transitioning from state s to s'. A transition can only occur between states s and s' if $\mathbf{R}(s,s') > 0$, and the probability of this transition being triggered within t time-units is $1 - e^{-\mathbf{R}(s,s') \cdot t}$. If there is more than one state s' for which $\mathbf{R}(s,s') > 0$, the first transition to be triggered determines the next state of the CTMC. The exit rate $E(s)$ is then equal to $\sum_{s' \neq s} \mathbf{R}(s,s')$ and the embedded DTMC is obtained as follows: $\mathbf{P}(s,s') = \mathbf{R}(s,s')/E(s)$ if $E(s) \neq 0$, 1 if $E(s) = 0$ and $s = s'$, and 0 otherwise.

Intuitively, the CTMC executes as follows: in each state s, it stays in this state for time t, drawn from the exponential distribution with parameter $E(s)$, and then moves to state s' with probability $\mathbf{P}(s,s')$. A *timed path* (or simply *path*) of C is a finite or infinite sequence $s_0 t_0 s_1 t_1 s_2 \cdots t_{n-1} s_n \ldots$, where $t_i \in \mathbb{R}_{>0}$ for each $i \geq 0$. As for DTMCs, a probability space over the paths through a CTMC can be defined [3], where events correspond to certain sets of paths.

3.2. Continuous Stochastic Logic (CSL)

To specify quantitative properties of CTMCs, the logic CSL [3] has been proposed, which is syntactically similar to PCTL, except that it replaces the step-bounded path operators with the continuous time-bounded variants. For example, in PCTL we can query the probability of reaching a Φ-state in 10 steps ($\mathrm{P}_{=?}[\mathrm{F}^{\leq 10}\, \Phi]$), whereas in CSL it is possible query the probability of reaching a Φ-state in 10.5 units of time ($\mathrm{P}_{=?}[\mathrm{F}^{[0,10.5]}\, \Phi]$).

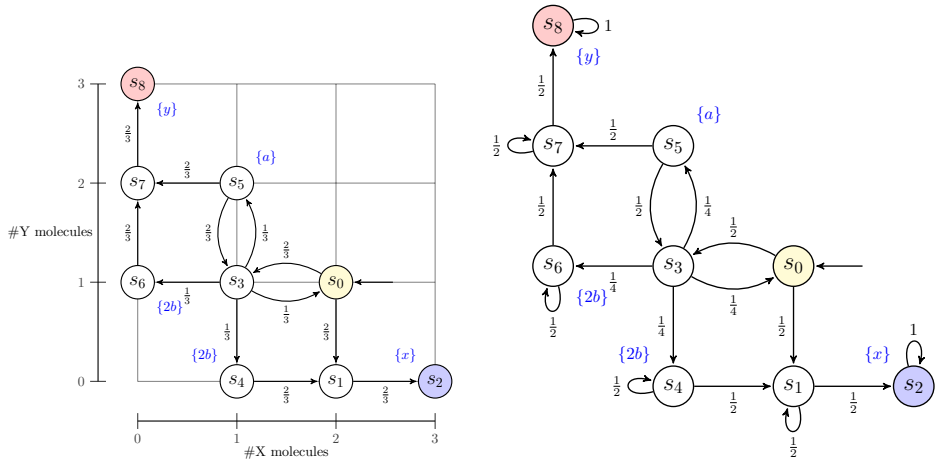
Definition 4. *The syntax of CSL is given by:*

$$\Phi ::= \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathrm{P}_{\sim p}[\psi] \mid \mathrm{S}_{\sim p}[\Phi]$$

$$\psi ::= \mathrm{X}\,\Phi \mid \Phi\,\mathrm{U}^{[t,t']}\,\Phi \mid \Phi\,\mathrm{U}\,\Phi$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ and $t, t' \in \mathbb{R}_{\geq 0}$.

The path formula $\Phi_1\,\mathrm{U}^{[t,t']}\,\Phi_2$, where $t, t' \in \mathbb{R}_{\geq 0}$ is true for a path if Φ_1 is satisfied at all time points until Φ_2 becomes true at a time point belonging to the interval $[t,t']$. The usual unbounded until $\Phi_1\,\mathrm{U}\,\Phi_2$ corresponds to the interval $[0,\infty)$. As for PCTL, we can define the derived variants, e.g. $\mathrm{F}\,\Phi$ (eventually). The probabilistic operator formula



(a) CTMC C (b) Uniformised DTMC of C

Figure 3. (Left) A CTMC C, with reaction rates, that formalises the approximate majority CRN with 2 initial molecules of X and 1 initial molecule of Y. The DTMC of Figure 1 is the embedded DTMC of C. (Right) The uniformised DTMC of C for $q = \frac{4}{3}$.

$$\mathbf{Q} = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{pmatrix} -\frac{4}{3} & \frac{2}{3} & 0 & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2}{3} & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & -\frac{4}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & -\frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{3} & 0 & -\frac{4}{3} & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{2}{3} & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{2}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{unif(\mathsf{C})} = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 4. The infinitesimal generator matrix (left) and the uniformised DTMC matrix (right) for CTMC C of Figure 3.

$\mathrm{P}_{\sim p}[\psi]$ is true in state s if the probability of paths from s that satisfy ψ meets the probability bound $\sim p$. The formula $\mathrm{S}_{\sim p}[\Phi]$ denotes steady-state, and is true in state s if the long-run probability of residing in state satisfying Φ meets the probability bound $\sim p$.

3.3. Model Checking for CSL over CTMCs

CSL model checking for the probabilistic operator reduces to transient probability calculation, defined as the probability of being in a certain state at time instant t, and typically proceeds through discretisation via uniformisation (a numerical transformation which optimises and numerically stabilises the computation of the transient probabilities). More specifically, for a CTMC $\mathsf{C} = (S, \bar{s}, \mathbf{P}, E, AP, L)$ with the rates matrix $\mathbf{R}$ we define its *infinitesimal generator matrix* $\mathbf{Q} : S \times S \to \mathbb{R}$ by $\mathbf{Q}(s,s') = \mathbf{R}(s,s')$ if $s = s'$ and $-\sum_{s' \neq s} \mathbf{R}(s,s') = E(s)$ otherwise. The *uniformised DTMC* $\mathbf{U} : S \times S \to [0,1]$ is then computed by $\mathbf{U} = \mathbf{I} - \mathbf{Q}/q$, where $\mathbf{I}$ is the identity matrix and $q \geq max\{E(s) \mid s \in S\}$. Intuitively, the execution of the CTMC is considered in terms of discrete jumps of the uniformised DTMC, each step corresponding to one 'epoch', where the value q is the

rate of the 'epoch'. The transient probability at time t can be approximated via an infinite summation of Poisson-distributed jumps corresponding to the epoch. Note that the uniformised DTMC is different from the embedded DTMC, in that it can contain additional self-loops on states where the residence time in the state is longer than $1/q$.

Example 5. *The infinitesimal generator matrix for CTMC* C *of Figure 3 is shown in Figure 4. The uniformised DTMC is computed by using the maximum waiting rate,* $q = \frac{4}{3}$*, determined from matrix* **Q**.

Model checking for CSL proceeds as for PCTL, by bottom-up traversal of the parse tree of the formula. For next state and unbounded until, computing the probability can can be done directly on the embedded DTMC **P**. Model checking for the probabilistic operator $\mathrm{P}[\Phi_l \ \mathrm{U}^{[t,t']} \ \Phi_2]$ reduces to the computation of transient probability, which is computed numerically on the uniformised DTMC **U**. Model checking for the steady-state operator involves solving the linear equation system $\pi \cdot \mathbf{Q} = 0$ subject to the constraint $\sum_{s' \in S} \pi(s') = 1$ for each bottom strongly connected component and combining the values probabilistic reachability for the bottom strongly connected components, and is usually computed on the embedded DTMC **P**.

Reward structures and operators $\mathrm{R}_{\sim r}[\mathrm{C}^{\leq t}]$ (*cumulative* reward up to time t) and $\mathrm{R}_{\bowtie r}[\mathrm{F}\Phi]$ (*reachability* reward) can also be added, similarly to the case of DTMCs. The intuition that state rewards are incurred in proportion to residence time, and this fact must be captured in the definition of the corresponding random variable. The meaning of a reward formula is defined in terms of the expectation of the corresponding random variable. The computation proceeds, as for the probabilistic operator, through uniformisation and recursive equations; for more information, see [19].

Example 6. *In a stochastic chemical reaction network, the propensity of a particular reaction follows an exponential distribution with a well defined rate [11]. In the approximate majority CRN, each reaction is* bimolecular *as there are always two reactant molecules. The propensity of a bimolecular reaction* α*, of the form* $A + B \rightarrow \ldots$*, is equal to* $k_\alpha \cdot \frac{\#A \cdot \#B}{v}$ *where* k_α *is reaction* α*'s rate constant, #A is the number of A molecules (similarly for B) and v is the size of the reaction volume. If we assume a uniform rate constant of* 1 *for all reactions and a reaction volume of size* 3*, the approximate majority CRN with 2 initial copies of molecule X and 1 initial copy of molecule Y can be formulated by CTMC* C *depicted in Figure 3 (left). The uniformised DTMC of* C *is also depicted in Figure 3 (right). The DTMC we studied in Example 1 is the* embedded *DTMC of* C*. As we have learned in this section, we must begin with a CTMC to answer queries concerning time. For example, while the embedded DTMC is sufficient to determine the number of expected* reaction steps *until a consensus is reached, we require the uniformised DTMC to determine the expected* elapsed time *until a consensus is reached.*

We complete this section by demonstrating how probabilistic model checking can be applied to analyse the behaviour of molecular networks modelled as CTMCs against temporal properties expressed in the logic CSL.

Example 7. *We now analyse the approximate majority system modelled by the CTMC shown in Figure 3 by probabilistic model checking. First, we show how to compute transient properties. Note that each curve in Figure 5 shows the probability that the number of X molecules in the system equals some particular quantity. The red curve is the*

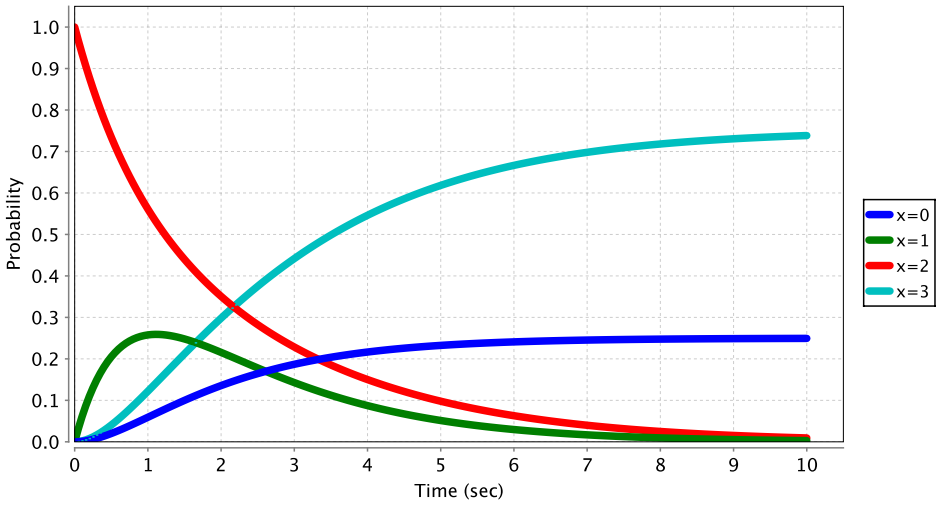


Figure 5. An example of computing transient properties of CTMC C. Shown is the probability, over time, that the X molecule is present in one of four possible quantities.

only one to have positive probability at time 0, as the system initially begins with 2 X molecules (and 1 Y molecule). Similarly, only the teal and blue curves converge towards a positive probability as the system will eventually form a consensus of 3 X molecules (teal curve) or a consensus of 3 Y molecules (blue curve). The probability for the system to have exactly i molecules of X at time T can be formulated as the query $\mathrm{P}_{=?}[\mathrm{F}^{[T,T]} a_i]$*, where atomic proposition* a_i *is defined to be true for a state if and only if the number of X molecules equals i. Solving the query for different instances of time gives insight into the evolution of the system as it reaches a consensus.*

4. Biological Case Studies

In the remainder of this chapter, we describe two cases studies where probabilistic model checking as described in previous sections has been used, with great effect, to study biological systems. We begin with a brief summary of an *in silico* study of a real-world molecular signalling pathway performed in collaboration with biologists [16]. We use this case study to introduce the probabilistic model checking tool PRISM and also demonstrate how it can be used to help answer biological hypotheses. Many details for this case study have been omitted, but the interested reader can find them, together with additional model checking techniques relevant to molecular networks, in the original publication [16]. We then describe, in greater detail, the application of probabilistic model checking in the context of DNA computing. The systems we study use strands of DNA to perform computation, including the approximate majority algorithm modelled earlier in the chapter. For this second case study, we demonstrate how PRISM can be used to verify a number of PCTL and CSL formulas (with and without rewards). For additional details of DNA strand displacement systems, or results omitted in this summary, the reader is referred to the original publication [23].

We also note that probabilistic model checking, and the PRISM tool in particular, has been used for a number of biologically motivated studies, including modelling of the

cell cycle switch [9], and verification of nanoscale DNA 'robots' that walk autonomously along a network of tracks [12].

4.1. Model Checking Molecular Networks

In this section, we show how one can use the probabilistic model checker PRISM [1,21] to model and analyse molecular networks. PRISM is a symbolic model checker which, amongst other model types, supports both discrete- and continuous-time Markov chains introduced earlier in this chapter, together with the associated logics PCTL and CSL. PRISM supports both numerical model checking for PCTL and CSL formulas [18], as well as the sampling-based, statistical model checking (confidence interval and hypothesis testing methods) [33]. It is a free, open source application which runs on all major operating systems, and comes with a GUI that supports a range of functionalities, including graph plotting and import/export to other tools, including via SBML (Systems Biology Markup Language).

Models are supplied to PRISM written in a simple textual language, which is a probabilistic variant of Reactive Modules due to Alur and Henzinger. A model is composed (by synchronising parallel composition) from modules, where each module represents an entity whose state is determined by a number of state variables which it controls, and which can transition to other states depending on the values of variables belonging to other modules or its own. A module therefore can naturally represent the behaviour of a molecule (or a population of molecules), where the transitions model biochemical reactions such as binding, releasing or degradation.

We will explain the modelling of molecular networks in PRISM using a simplified FGF (Fibroblast Growth Factor) case study [16]. FGF are a family of proteins which play an important role in cell signalling, and have been linked to e.g. skeletal development and wound healing. The mechanisms of FGF signalling are not well understood, and several hypotheses exist, particularly regarding the role of phosphorylation and degradation. Since certain molecules in the pathway occur in small numbers in the cell, the discrete stochastic modelling framework, and in particular continuous-time Markov chain models, are particularly appropriate. In [16], we studied a number of detailed hypotheses using probabilistic model checking in PRISM.

The simplified set of reactions based on the role of FGF in receptor biosynthesis is given below. An FGF molecule can bind to FGF receptor (FGFR) to form a compound molecule FGFR:FGF, via reversible reaction (1), and the compound can unbind via reaction (2). Whilst bound, FGFR can become phosphorylated, resulting in FGFRP, reaction (3), and FGFRP can dephosphorylate, via the reverse reaction (4). Finally, the phosphorylated FGFR (FGFRP) can relocate, but only when phosphorylated, via reaction (5). The reactions are annotated with kinetic rates $k_1, \cdots, k_5$ in s^{-1} and (for bimolecular reactions) also molar concentrations, i.e. the units are $M^{-1}s^{-1}$.

FGF binds/releases FGFR:

$$FGF + FGFR \rightarrow FGFR : FGF \qquad \text{with rate } k_1 = 5000\,M^{-1}s^{-1} \qquad (1)$$

$$FGF + FGFR \leftarrow FGFR : FGF \qquad \text{with rate } k_2 = 0.002\,s^{-1} \qquad (2)$$

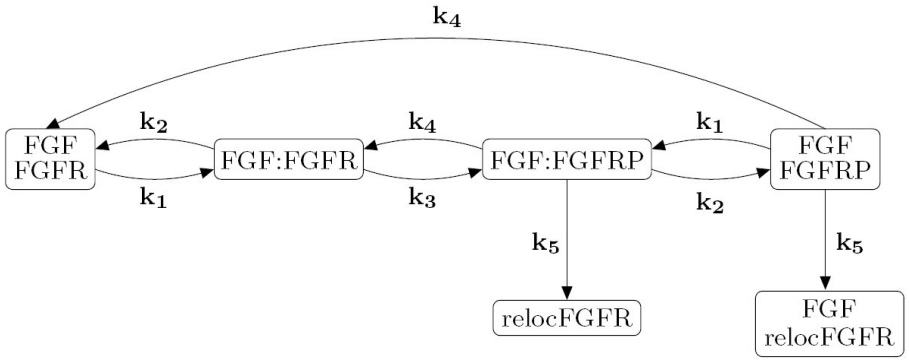


Figure 6. The resulting CTMC that models reaction equations (1)–(5).

Phosphorylation of FGFR (whilst FGFR:FGF):

$$FGFR \rightarrow FGFRP \qquad \text{with rate } k_3 = 0.1\,s^{-1} \tag{3}$$

Dephosphorylation of FGFR:

$$FGFRP \rightarrow FGFR \qquad \text{with rate } k_4 = 0.01\,s^{-1} \tag{4}$$

Relocation of FGFR (whilst FGFRP):

$$FGFR \rightarrow relocFGFR \qquad \text{with rate } k_5 = 1/60\,min^{-1} \tag{5}$$

If we assume a spatially uniform mixture in a fixed volume at constant pressure and temperature, the above biochemical reactions can be shown to induce a (time-homogeneous) continuous-time Markov chain, since the rates are only dependent on the states. The resulting CTMC, where we model one molecule of each species only, is shown in Figure 6. The stochastic rates ($\mathbf{k}_i$) are obtained from the kinetic rates (k_i) by dividing by volume multiplied by the Avogadro number [14,20].

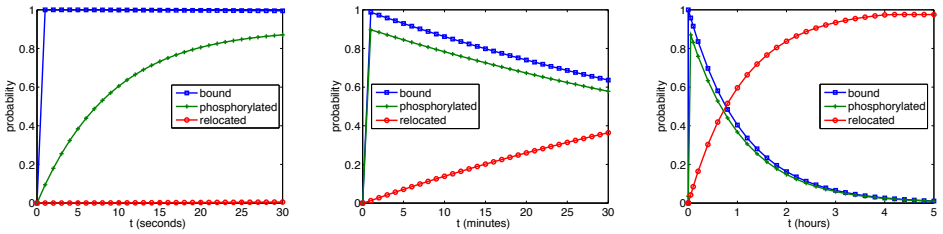
In the modelling language of PRISM, the reaction network is shown in Figure 7. We represent the network by two modules, molecule FGF and FGFR, which encode their states (e.g. free, bound, etc) using an integer variable. The reactions between the species are modelled by synchronisation via the actions `bind` and `rel`. Guarded commands, e.g. `[bind] fgf=0 -> (fgf'=1);` state the conditions (e.g. `fgf = 0`) under which an update (which assigns a new value to the variable), e.g. `fgf' = 1` can be executed, while synchronising on action `bind`. The auxiliary module RATES takes care of the handling of the rates via synchronisation. Note that this model represents a single molecule of FGF and FGFR, but it is also possible to devise models that consist of populations of molecules. More detail about modelling approaches can be found in [20].

PRISM models of molecular networks can now be subjected to detailed quantitative analysis, but how should we use the models to aid the process of scientific discovery? The key idea is to represent each hypothesis in terms of a network of reactions, model

```
ctmc

const double k1 = 5000;          // rate of binding
const double k2 = 0.002;         // rate of release
const double k3 = 0.1;           // rate of phosphorylation
const double k4 = 0.01;          // rate of dephosphorylation
const double k5 = 1/(60 * 60); // rate of relocation

module FGF

fgf: [0..2] init 0;          // 0 - free, 1 - bound, 2 - removed from system
[bind]  fgf=0 -> (fgf'=1); // FGF and FGFR bind
[rel]   fgf=1 -> (fgf'=0); // FGF and FGFR unbind
[reloc] fgf=1 -> (fgf'=2); // FGF disappears from the system since
                           // the time it was bound and FGFR relocates
endmodule

module FGFR

fgfr:  [0..1] init 0; // 0 - free, 1 - bound
phos:  [0..1] init 0; // 0 - unphosphorylated, 1 - phosphorylated
reloc: [0..1] init 0; // 0 - not relocated, 1 - relocated

[bnd] reloc=0 & fgfr=0           -> k1 :(fgfr'=1); // FGF and FGFR bind
[rel] reloc=0 & fgfr=1           -> k2 :(fgfr'=0); // FGF & FGFR release
[]    reloc=0 & fgfr=1 & phos=0 -> k3 :(phos'=1); // FGFR phosphorylates
[]    reloc=0 & phos=1           -> k4 :(phos'=0); // FGFR dephosphorylates
[]    reloc=0 & phos=1           -> k5 :(reloc'=1); // FGFR relocates

endmodule

module RATES

[bind]  true -> k1 : true; // FGF and FGFR bind
[rel]   true -> k2 : true; // FGF and FGFR unbind
[reloc] true -> k5 : true; // FGFR relocates

endmodule
```

Figure 7. PRISM model for the CTMC of Figure 6.

the reactions in PRISM or input them into PRISM via SBML, and perform a series of *in silico* experiments on the resulting model for a range of CSL properties. The outcome of probabilistic model checking can be collected in quantitative plots of probability and/or expected cost, shown over the time evolution of the model or for a range of model or formula parameters. The trends in the quantitative plots can be used to either confirm known facts, or help identify unusual behaviour that can guide wetlab experiments. This type of analysis can also include *in silico* genetic knock-out experiments, where a species is removed in order to study its influence on the network behaviour. Indeed, in the FGF case study [16] we successfully identified predictions that were later confirmed in wetlab experiments [17].



(a) Time scale seconds (b) Time scale minutes (c) Time scale hours

Figure 8. Shown are the probabilities, over three time scales, for the FGFR system to be in one of three states.

For the simplified FGF network, and assuming appropriate reward structures, the following are examples of properties that can be analysed:

- $(phos = 1) \implies \mathrm{P}_{>0.1}[\mathrm{F}^{[0,t]}\ (reloc = 1)]$ - "if FGFR is currently phosphorylated, then the probability of it being relocated within the next t seconds is greater than 0.1"
- $\mathrm{P}_{=?}[\mathrm{F}^{[t,t]}\ (fgf = 1)]$ - "the probability that FGF is bound to FGFR at time instant t"
- $\mathrm{R}\{\text{"time"}\}_{=?}[\mathrm{F}\ (reloc = 1)]$ - "the expected time taken before FGFR relocates", where "time" is a reward structure that assigns state rewards of 1 and transition rewards of 0
- $(reloc = 0) \implies \mathrm{R}\{\text{"bind"}\}_{\geq 2.4}[\mathrm{C}^{\leq t}]$ - "if FGFR is not relocated, the expected number of bindings during the next t seconds is at least 2.4", where "bind" is the reward structure which assigns the transition rewards of 1 iff the reaction is *bind* and state reward 0 for all states.

To illustrate the quantitative analysis and what can be learnt, in Figure 8 we show results obtained with PRISM, over different timescales, for the probabilities that, at time instant t, FGFR is: (i) bound to FGF; (ii) phosphorylated; and (iii) relocated. For example, Figure 8(a) demonstrates that, initially, FGF and FGFR bind very quickly and remain bound, but that, as time progresses (Figure 8(b)), their chance of binding diminishes and FGFR becomes fully relocated (Figure 8(c)).

Summarising, we have shown here how to model a hypothesis about molecular interactions using PRISM, and the type of quantitative analysis that can be obtained by exploiting CSL model checking. For simplicity, we have worked with a model which contains one molecule each of FGF and FGFR, but in the next section we will show how populations of molecules can also be modelled. We note, however, that scalability can be an issue for large population counts and methods to tackle state-space explosion are sought for. Techniques such as symmetry reduction, for example, can assist in reducing the size of the model so that analysis with PRISM becomes feasible [16].

4.2. Model Checking DNA Strand Displacement Systems

The examples up until now have focused on the approximate majority chemical reaction network. But how can these reactions be implemented in practice? Recent work in DNA computing provides an answer. Soloveichik et al. [29] have shown that any chemical reaction network can be realized using so called *DNA strand displacement systems (DSD)*

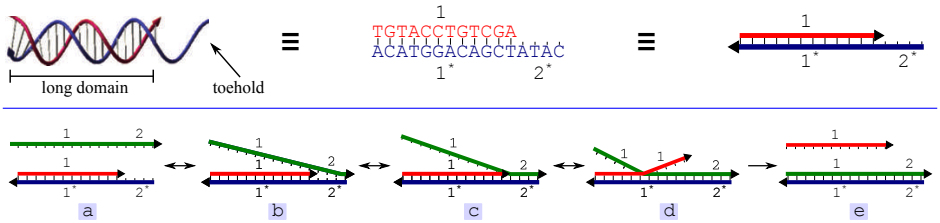


Figure 9. Top panel: A DNA duplex with an unbound *toehold* is formed by the base pairing of complementary *long domain*s of two strands of DNA. In domain level diagrams (top right), the sequences of strands are abstracted into domains indicating which are complementary to others. **Bottom panel**: Toehold-mediated DNA strand displacement (DSD) is initiated when the green invading strand forms base pairs with an unbound toehold on the blue template strand *(events a–c)*. The invading strand competes for base pairs with the red incumbent strand *(event d)*, via a random walk process, until the incumbent is *displaced* and disassociates from the template strand *(event e)*.

[34,36,35]. In a DSD, molecules of a particular species are represented by a strand of DNA. Chemical reactions can be simulated by a series of strand displacement events, where a free *signal* strand can displace another strand that shares a common domain, or sequence. In this section, we give an overview of DSD systems, how strand displacement works, and then highlight key examples from a recent study that explored the use of probabilistic model checking techniques in DSDs [23]. The study used the DNA strand displacement programming language (DSD) [27,24] developed to facilitate the design, simulation and analysis of DNA strand displacement devices. DSD has been integrated with PRISM, via SBML, to enable probabilistic verification, as an alternative to stochastic simulation.

4.2.1. DNA strand displacement systems (DSD)

Strands of DNA are an oriented sequence of four bases: `A`, `C`, `G` and `T`. We say strands are oriented because there is a 5' and 3' end. Two strands in opposite orientation can hybridize to one another by creating hydrogen bonds between Watson-Crick base pairs: an `A` base can bond with a `T` base, and a `C` base can bond with a `G` base. Consider the red and blue strands of DNA in the top of Figure 9. The sequence of the red strand is abstracted into a *domain* labeled 1. This domain is *complementary* and in opposite orientation to a domain on the blue strand labeled 1^*. Because these domains are complementary, the red and blue strand can hybridize to form a duplex region. The stability of binding between two strands is dependent on the temperature of the system. In this simplified example, 1 and 1^* are *long* domains, meaning that they form enough bonds such that they will not spontaneously disassociate. Note that the blue strand still has a short, unbound domain (2^*) called a *toehold*. Now consider what happens when the green strand is introduced to the system (see Figure 9 bottom). The green strand is fully complementary to the blue strand. Roughly speaking, in a well-mixed solution of DNA, the system becomes most stable when the number of base pairs is maximized. Because of this, the complementary toehold domain of the green strand, labeled 2, will eventually hybridize to the *free* toehold on the blue strand, labeled 2^*. A toehold domain is too short to ensure a stable binding between strands. Therefore, the green strand may spontaneously disassociate and associate with the blue (events *a–c* in Figure 9). Eventually, because it is a complement of the blue strand, the green strand will compete for base pairs with the red strand in a

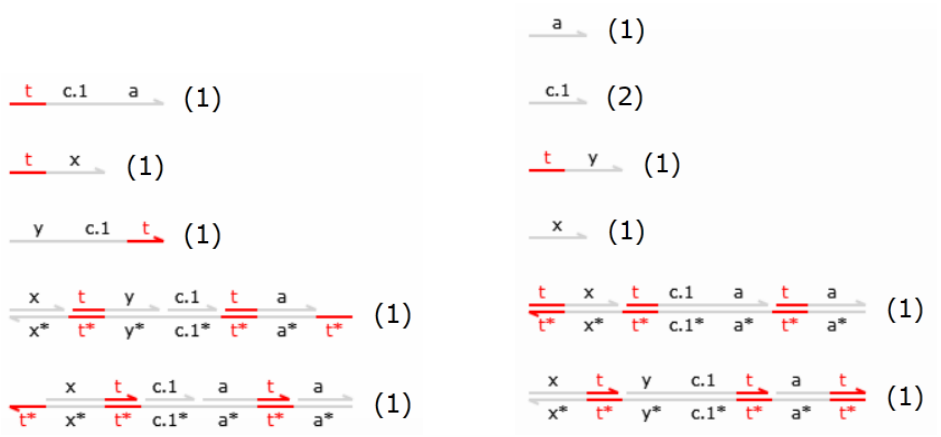


Figure 10. Two of the possible states of a DNA strand displacement system that implements the transducer reaction $X \to Y$. A universal toehold domain, t, is used throughout the implementation. The signal X is represented by a strand consisting of a toehold t followed by a long domain x (similarly for signal Y). The quantity for each strand type is shown in parentheses. **(a)** The initial state of the transducer consisting of the strand representing signal X and the remaining strands and complexes that comprise the *transducer gate*. **(b)** The final state of the transducer gate which consists of the strand representing signal Y and unreactive waste produced by the transducer gate. Note that this final state is a *deadlock* state as there are no free toeholds to permit an additional strand displacement.

random-walk like process called *branch migration* (event e in Figure 9). When the green strand has formed all possible complementary base pairs to the blue strand, the red strand will disassociate. This process is referred to as *toehold-mediated strand displacement* and the red strand is said to have been *displaced*. In a DSD, we refer to the blue strand as the *template*, the red strand as the *incumbent*, and the green strand as the *invader*. Note that this displacement is considered irreversible as there is no free toehold for the red strand to bind and initiate branch migration.

Sets of strands can be designed to perform useful computation. For example, we could consider the strands of Figure 9 to behave as a primitive logic gate: if the green strand is present (the *input*), then it can react with the red and blue complex (the *gate*) to release the red strand (the *output*). A number of gate designs for DSD systems have been proposed and demonstrated to work in experiments. In the remainder of this case study, we will focus on variations of gates proposed by Cardelli [7,8]. In the examples we study, we will assume a *universal* toehold domain, meaning that all toehold domains use the same sequence. Similarly, the complementary toehold domains of template strands share a common sequence.

4.2.2. Verifying the correctness of DSD transducer gates

How can we implement the chemical equation $X \to Y$ with a DSD system? One solution is depicted in Figure 10. In this scheme, molecules such as X are represented by a *signal* strand consisting of one toehold and one long domain that identifies the species of the molecule it represents. The quantity of X present is equal to the number of *reactive* signal strands that represent X. A strand is said to be *reactive* if it contains a toehold and is

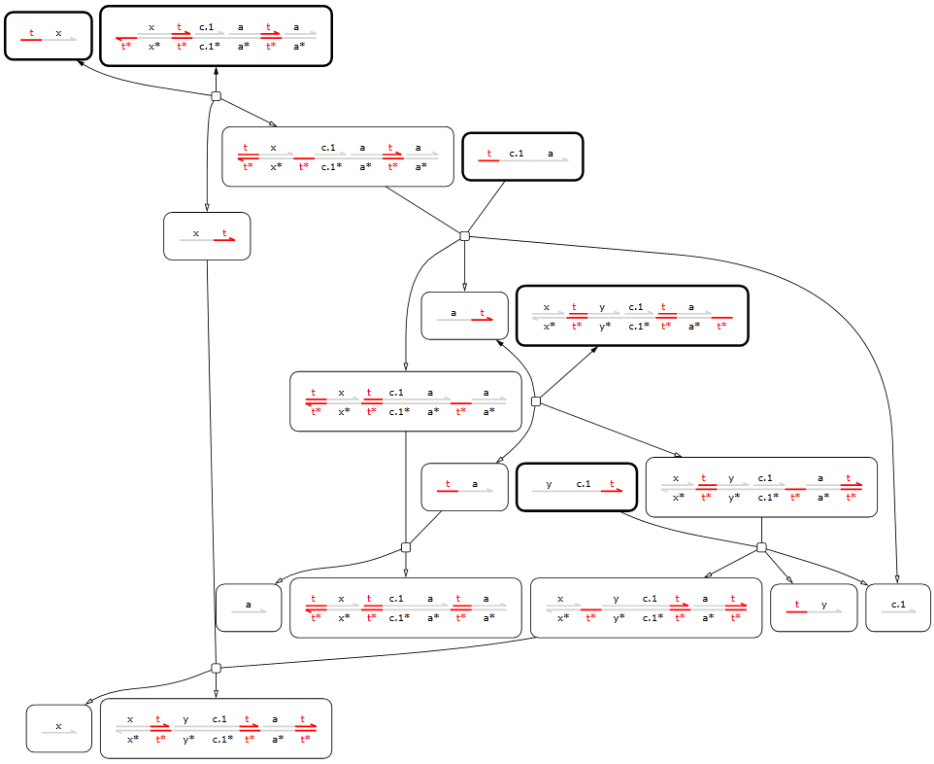


Figure 11. A graph showing the inputs and outputs (depicted in large boxes) of reactions (depicted as small boxes) between the initial and final states of the transducer gate in Figure 10. The species in the initial state are shown in bold.

not bound to any other strand. Similarly, a gate is reactive if it contains a free toehold. Figure 10a shows the initial state of a system consisting of a single reactive copy of signal X; the remaining strands and complexes form a *transducer* gate capable of consuming a signal strand X and producing a signal strand Y. The final state of this system (see Figure 10b), consisting of a single copy of Y, is reached after a series of displacement reactions. Note that, by design, there are no free toeholds of the transducer gate in the final state. The gate is said to be *unreactive* — a desirable property once it has completed its goal.

The state-space for DSD systems can be very large. For example, a compressed representation of the simple system described above —depicting input and outputs of displacement reactions— is shown in Figure 11. Manual verification of this simple system is possible, but much more complicated systems require the use of automated verification. We now illustrate the use of non-probabilistic model checking to identify a *design error* in a more complicated transducer example.

Consider a system that begins with a single copy of molecule X_0 and should end with a single copy of X_2 by using two chemical reactions: $X_0 \rightarrow X_1, \quad X_1 \rightarrow X_2$. This system could be implemented by two transducer gates in series (one for each reaction). However, as previously pointed out in the literature [8], this has the potential for *cross-talk* between the two gates. Cross-talk could lead to an undesirable state of the system.

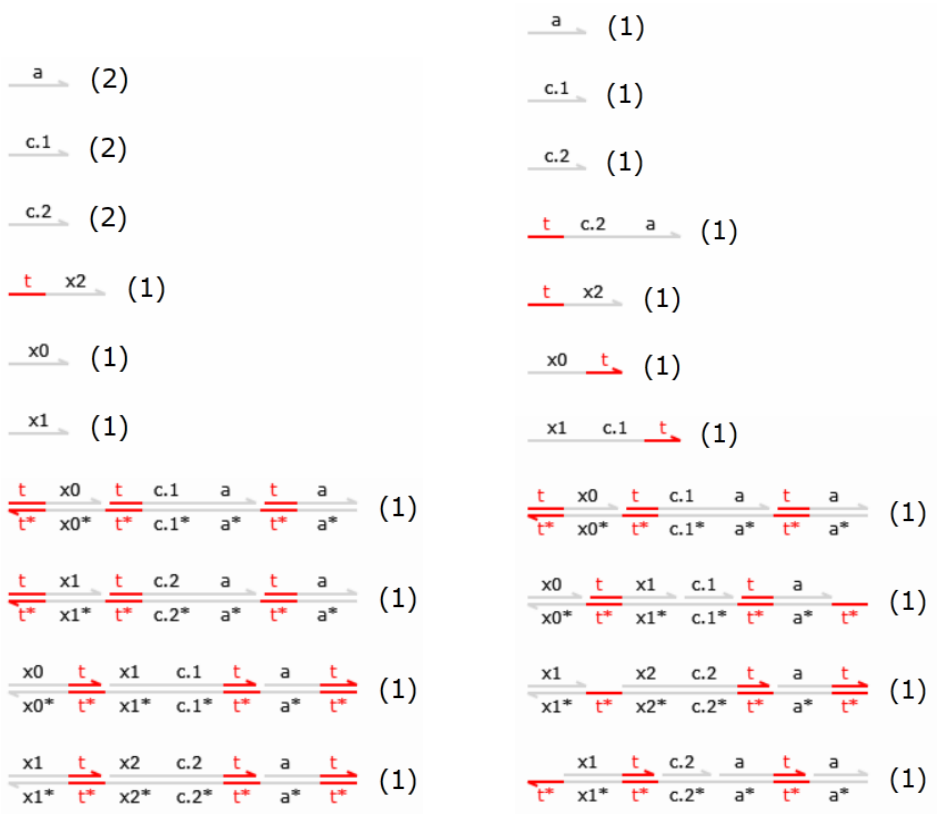


Figure 12. Two final states are shown for a system that should implement two transducer gates in series: $X_0 \rightarrow X_1$, $X_1 \rightarrow X_2$. **(a)** The *correct* final state which has produced X_2 with the intended sequence of displacement reactions and does not have free toehold domains. **(b)** An incorrect final state which has produced X_2 by an unintended sequence of displacement reactions and does have free toehold domains.

Two unique deadlock states (i.e., no additional reactions are possible) are shown for this system in Figure 12. Figure 12a shows the intended final state of the system: the X_2 signal strand has been produced and all gates are *unreactive*. In contrast, Figure 12b shows an undesirable deadlock state: the X_2 signal has been produced by an unintended sequence of reactions resulting in gates that are still reactive.

PRISM can be used to automatically identify this error and produce a trace of reactions from the initial state to the unintended deadlock state. First, we must identity the intended final state of the system where the correct output is produced and no gates are reactive. This is formalised in PRISM with the following code:

```
label "all_done" = strands_reactive=output &
                   output=N & gates_reactive=0
```

The code uses two formulas, `strands_reactive` and `gates_reactive`, that respectively count the number of strands and gates that are reactive in a state. In this example, the variable `output` is the number of reactive signal strands that represent X_2

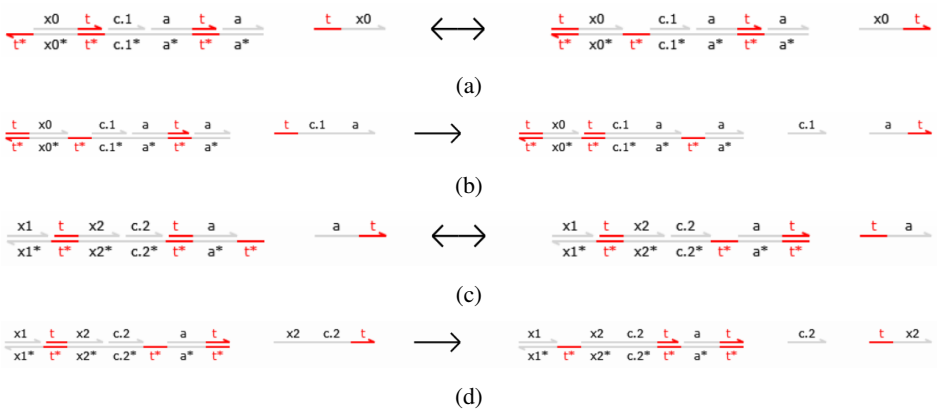


Figure 13. A series of reactions of two, in series, transducer gates ($X_0 \to X_1$, $X_1 \to X_2$) that leads to the incorrect final state depicted in Figure 12b.

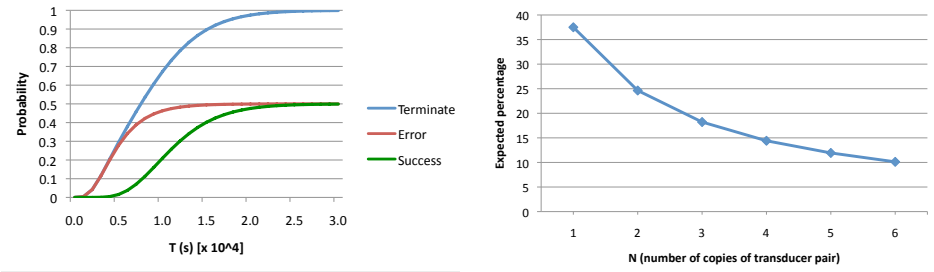
and `N` is the number of parallel copies of the system in the same solution. Therefore, the intended behaviour of the system is to produce `N` copies of the signal strand representing X_2, before reaching a deadlock state that contains no reactive gates and no reactive strands other than those representing X_2. This can be formalized using the following (non-probabilistic) CTL properties which can be verified by PRISM:

```
A [ G "deadlock" => "all_done" ]
E [ F "all_done" ]
```

The first property ensures that all deadlock states, reachable from the initial state, are correct, i.e., all output strands are produced and no other gates or strands are reactive. The second property ensures that a correct final state is reachable from the initial state. Using PRISM on this example we find the second property is verified to be true, but the first is found to be false and results in a counterexample in the form of a reaction trace leading from the initial state to the incorrect deadlock state, given in Figure 13. The first two reactions proceed, as intended, on the same gate and produce a strand with a long domain labeled a (Figures 13a and 13b). This strand should next react with the second complex of the gate that produces the X_1 signal. Instead, it is possible for this strand to react immediately with the gate that produces the X_2 signal (Figure 13c), permitting another strand to displace the X_2 signal (Figure 13d), without the X_1 signal ever having being produced. The trace produced by PRISM makes it clear why this unintended state is reached: both the $X_0 \to X_1$ and $X_1 \to X_2$ transducer use an auxiliary strand with the same long domain (labeled a). By instead using unique domains for auxiliary strands of different gates, this bug is removed and the gates act as intended.

4.2.3. Verifying the reliability and performance of DSD transducer gates

In addition to correctness properties, PRISM can be used to examine quantitative properties of DSD systems. Consider again the faulty pair of in series transducers (i.e., those implementing $X_0 \to X_1$, $X_1 \to X_2$). The following CSL formulas (in PRISM syntax) can be used to determine the probability, at a specific time T, that the system will have (a) terminated, (b) terminated correctly, and (c) terminated incorrectly:



(a) (b)

Figure 14. **(a)** Shown are the probabilities, over time, of a pair of faulty, in series, transducer gates to (i) terminate, (ii) terminate in an error state, and (iii) terminate in a correct state. **(b)** As the number of parallel copies of the faulty, in series, transducer gates increase, the expected number of unreacted gates in the final state decreases.

```
P=? [ F[T,T] "deadlock" ]
P=? [ F[T,T] "deadlock" & "all_done" ]
P=? [ F[T,T] "deadlock" & !"all_done" ]
```

The probability for these three queries is shown for different values of time in Figure 14a. As expected, the probability that the system eventually deadlocks converges towards 1. Otherwise, the plot indicates that, early on, the system is more likely to proceed towards the error state than the intended state. The reasoning for this is that reaching the intended state requires a number of additional intermediate reactions compared to reaching the erroneous state. The plot also shows that the probabilities for the system to end in a correct or incorrect state each converge towards 0.5. The following queries, which do not use a time bound, confirm that the probability of reaching each of the final states is 0.5 (i.e., they are equally likely):

```
P=? [ F "deadlock" & "all_done" ]
P=? [ F "deadlock" & !"all_done" ]
```

This is also expected as the strand with long domain *a* produced in the first complex of the first gate could either interact with the second complex of the first gate (intended reaction), or the second complex of second gate (unintended reaction). In either case, the very next reaction is irreversible and only one of the two final states of Figure 12 will be reachable.

Clearly, a 0.5 probability of failure of a logic gate, without a mitigating design, is unacceptable for reliable computation. Fortunately, computer science has a rich tradition of performing reliable computation from unreliable parts [31]. Cardelli suggests that, by increasing the number of parallel copies of these faulty transducer gates, the overall reliability will increase [8]. The hypothesis is that the increased number of additional auxiliary strands could be used to *unblock* gates in the incorrect deadlock state. This hypothesis can be tested by determining, for different numbers of initial parallel copies, the expected number of reactive gates in the incorrect deadlock state. The following query determines this probability when the number of reactive gates equals *i*:

```
P=? [ F "deadlock" & gates_reactive=i ]
```

Figure 14b shows the percentage of reactive gates for various numbers of initial copies of the system. The hypothesis is supported by this plot which shows that, indeed, the percentage of reactive gates decreases as the number of initial copies of the system increases (i.e., the more copies of the system, the more reliable the computation).

As previously stated, the faulty behaviour of the transducer gates can be corrected by ensuring auxiliary strands are unique to each gate. Using these corrected gates, we can test another hypothesis related to performance. It has been formally shown that the expected time for a DSD circuit to complete scales linearly with the depth of the circuit [28]. We have so far focused on two transducer gates in series. To test the performance hypothesis, we can instead see how expected time for a circuit to complete scales by increasing the number of transducers in the series (e.g., $X_0 \to X_1, \quad X_1 \to X_2, \quad \ldots, \quad X_{k-1} \to X_k$). This can be accomplished with a rewards structure called "time", that assigns 1 to each state of the model, and with the following query:

```
R{"time"}=? [ F "all_done" ]
```

Intuitively, the above query determines the expected time that will elapse before reaching the deadlock state. (With the corrected transducers, this will always be the correct state with no reactive gates.) The results for $1 \ldots 7$ transducers in series, which confirm the linear scaling of expected time, are plotted in Figure 15a.

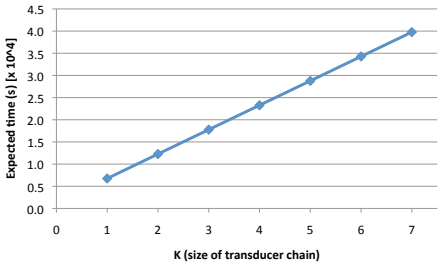
4.2.4. Verifying a DSD implementation of approximate majority

We end this case study by briefly considering a DSD implementation of the approximate majority CRN that we investigated in the examples of Section 2 and Section 3. The approximate majority CRN consists of four *bimolecular* reactions. While the ideas are similar, the details of a DSD gate that implements a bimolecular reaction are more complicated than one implementing a transducer gate. Readers interested in these details are directed to the original study [23]. Furthermore, as each reaction of the CRN is implemented by a cascade of multiple DSD reactions, the resulting CTMC of the DSD implementation is significantly larger than the CTMC shown in Figure 3, making automated verification all the more useful.

Recall that the input to the approximate majority algorithm is some quantity of X molecules and some quantity of Y molecules. The final state of the system results in a consensus of either X molecules or Y molecules. It has been formally shown that the approximate majority algorithm will form a consensus of the species initially present in the majority, with high probability, provided that the majority outnumbers the minority by a significant margin [2]. In particular, if the system consists of $\Theta(N)$ molecules in total, and X forms the initially majority and outnumbers the quantity of Y by $\Omega(\sqrt{N})$, then the algorithm will converge to an X consensus with high probability. We can determine the probability of reaching an X consensus or Y consensus, in a system where sum of initial X and Y molecules is N, using the following queries:

```
P=? [ F output_x=N ]
P=? [ F output_y=N ]
```

Here, the variables `output_x` and `output_y` count the number of X and Y molecules, respectively, in the consensus state. The probabilities for the DSD systems to converge to an X consensus are given in Figure 15b for different initial counts of X and



(a)

	$Y=1$	$Y=2$	$Y=3$	$Y=4$	$Y=5$
$X=1$	0.5000	0.2531	0.1290	0.0658	0.0334
$X=2$	0.7468	0.5000	0.3156	0.1917	0.1131
$X=3$	0.8709	0.6843	0.5000	0.3462	0.2299
$X=4$	0.9341	0.8082	0.6537	0.5000	0.3651
$X=5$	0.9665	0.8869	0.7699	0.6349	0.5000

(b)

Figure 15. **(a)** The expected time for a chain of k transducer gates to complete increases with linearly in k. **(b)** The probability of reaching an X consensus in the DSD implementation of the approximate majority CRN is shown for various initial population sizes of X and Y.

Y molecules (strands). As one would expect, when both X and Y are initially present in equal quantity, there is a 0.5 probability of reaching an X consensus. Furthermore, the probability of an X consensus becomes more or less likely depending on the initial ratio of X and Y molecules.

5. Conclusions

In this chapter we briefly overviewed probabilistic model checking for discrete- and continuous time Markov chains, and introduced the main features of the probabilistic model checker PRISM [21]. More detail on probabilistic model checking can be found in the textbook [4], the tutorial paper [19] and the PRISM website [1]. We then discussed applications of probabilistic model checking in the context of biological systems, focusing on molecular signalling and DNA computation. The former study [16] enables predictive modelling of signalling pathways, which has been shown to assist in gaining a better understanding of biological functions in the wetlab [17]. The latter application [23] is more akin to the use of model checking in hardware verification, where it serves the function of a computer-aided design and verification environment, opening up exciting opportunities for probabilistic verification to play a part in designing, debugging and optimising molecular-scale devices. In addition, PRISM has been used in several biologically motivated studies, including RKIP-inhibited ERK pathway [6], influenza virus fusion [13], bone pathologies [25], modelling of the cell cycle switch [9], and verification of nanoscale DNA 'robots' that walk autonomously along a network of tracks [12]; see the Case Studies section of the PRISM website [1].

Acknowledgements. The authors are part supported by ERC Advanced Grant VERIWARE and the Oxford Martin School.

References

[1] PRISM website. `www.prismmodelchecker.org`.
[2] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29:524–541, 2003.
[4] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
[5] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. FSTTCS'95*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
[6] M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using continuous time Markov chains. *Transactions on Computational Systems Biology VI*, 4220:44–67, 2006.
[7] L. Cardelli. Strand algebras for DNA computing. In R. J. Deaton and A. Suyama, editors, *DNA*, volume 5877 of *Lecture Notes in Computer Science*, pages 12–24. Springer, 2009.
[8] L. Cardelli. Two-domain DNA strand displacement. *Developments in Computational Models*, 26:47–61, 2010.
[9] L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2, 2012.
[10] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.
[11] M. Cook, D. Soloveichik, E. Winfree, and J. Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.
[12] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. DNA walker circuits: Computational potential, design, and verification. In *Proceedings of the 19th International Conference on DNA Computing and Molecular Programming (DNA 19)*, 2013.
[13] M. P. Dobay, A. Dobay, J. Bantang, and E. Mendoza. How many trimers? Modeling influenza virus fusion yields a minimum aggregate size of six trimers, three of which are fusogenic. *Molecular BioSystems*, 2011.
[14] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
[15] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994. 10.1007/BF01211866.
[16] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 319(3):239–257, 2008.
[17] M. Kwiatkowska and J. Heath. Biological pathways as communicating computer systems. *Journal of Cell Science*, 122(16):2793–2800, 2009.
[18] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.
[19] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.
[20] M. Kwiatkowska, G. Norman, and D. Parker. *Symbolic Systems Biology*, chapter Probabilistic Model Checking for Systems Biology, pages 31–59. Jones and Bartlett, 2010.
[21] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
[22] M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney. Simulation and verification for computational modelling of signalling pathways. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, editors, *Proc. Winter Simulation Conference*, pages 1666–1675. Omnipress, 2006.
[23] M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.
[24] M. R. Lakin, S. Youssef, L. Cardelli, and A. Phillips. Abstractions for DNA circuit design. *J R Soc Interface*, Jul 2011.
[25] P. Liò, E. Merelli, and N. Paoletti. Multiple verification in computational modeling of bone pathologies. In *Proc. 3rd International Workshop on Computational Models for Cell Processes (COMPMOD'11)*, volume 68 of *EPTCS*, pages 82–96, 2011.

[26] A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 2009.

[27] A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *J R Soc Interface*, 6 Suppl 4:S419–S436, Aug 2009.

[28] G. Seelig and D. Soloveichik. Time-complexity of multilayered dna strand displacement circuits. In *DNA computing and molecular programming*, pages 144–153. Springer, 2009.

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

[30] M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:327–338, 1985.

[31] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[32] O. Wolkenhauer, M. Ullah, W. Kolch, and K.-H. Cho. Modeling and simulation of intracellular dynamics: choosing an appropriate framework. *NanoBioscience, IEEE Transactions on*, 3(3):200–207, 2004.

[33] H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):216–228, 2006.

[34] B. Yurke and A. P. M. Jr. Using DNA to Power Nanostructures. *Genetic Programming and Evolvable Machines*, 4(2):111–122, 2003.

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

[36] D. Y. Zhang, A. J. Turberfield, B. Yurke, and E. Winfree. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science*, 318(5853):1121, 2007.