On Quantitative Modelling and Verification of DNA Walker Circuits Using Stochastic Petri Nets

Benoît Barbot and Marta Kwiatkowska

University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Abstract. Molecular programming is an emerging field concerned with building synthetic biomolecular computing devices at the nanoscale, for example from DNA or RNA molecules. Many promising applications have been proposed, ranging from diagnostic biosensors and nanorobots to synthetic biology, but prohibitive complexity and imprecision of experimental observations makes reliability of molecular programs difficult to achieve. This paper advocates the development of design automation methodologies for molecular programming, highlighting the role of quantitative verification in this context. We focus on DNA 'walker' circuits, in which molecules can be programmed to traverse tracks placed on a DNA origami tile, taking appropriate decisions at junctions and reporting the outcome when reaching the end of the track. The behaviour of molecular walkers is inherently probabilistic and thus probabilistic model checking methods are needed for their analysis. We demonstrate how DNA walkers can be modelled using stochastic Petri nets, and apply statistical model checking using the tool Cosmos to analyse the reliability and performance characteristics of the designs. The results are compared and contrasted with those obtained for the PRISM model checker. The paper ends by summarising future research challenges in the field.

1 Introduction

Molecular programming is an emerging field concerned with building synthetic biomolecular computing devices at the nanoscale, for example from DNA or RNA molecules. Several nanotechnologies have been developed, of which *DNA strand displacement (DSD)* [56,55] is particularly popular, since it uses only DNA molecules, is enzyme-free, and easy to synthesize chemically. DSD has ben used to implement logic circuits [44,41], diagnostic biosensors [29] and controllers programmed in DNA [11]. Further, DNA self-assembly technologies such as origami folding [43] have enabled novel designs, including *DNA walker systems* that can traverse tracks 'printed' on origami tiles and deliver cargo [54,51].

Molecular computing devices built from DNA are *autonomous* – they can interact with the biochemical environment, process information, make decisions and act on them – and *programmable*, that is, they can be systematically configured to perform specific computational or mechanical tasks. The computational

power of such systems has been shown to be equivalent to Turing computability [45]. The future potential of these developments is tremendous, particularly for smart therapeutics, point-of-care diagnostics and synthetic biology. For example, *biosensing* involves a decision process that aims to detect various input biomarkers in an environment, such as strands of messenger RNA within a cell, and take action based on the detected input.

Since such systems can perform information processing within living cells, their use is envisaged in healthcare applications, where safety is paramount. As argued in [32], this paper advocates the development of design automation methodologies for molecular programming. There are similarities to existing work in design automation for silicon circuits and hardware verification, but we must consider inherent *stochasticity* of the underlying molecular interactions, the need to state requirements in *quantitative* form, and the importance to consider *control* of molecular systems. Therefore, *probabilistic modelling* and *automated, quantitative verification* and *synthesis* techniques are needed [31,33,34].

In this paper, we focus on DNA walker circuits introduced in [8,50,51]. The behaviour of molecular walkers is inherently probabilistic and we have studied their performance and reliability in [15,14,16]. In [38], we have developed techniques to automatically synthesise rates so that a given quantitative requirement is guaranteed to be satisfied. The models of DNA walkers were developed in PRISM's modelling language, a notation based on reactive modules [34]. DNA walkers, however, perform spatially localised computation by following programmable tracks. Therefore, graphical notations such as Petri nets are particularly well suited to their modelling and analysis. We demonstrate how DNA walker circuits can be modelled using stochastic Petri nets, matching the layout of the original circuit designs, and analyse the reliability and performance characteristics of the designs. In view of state-space explosion observed in the original study [15,16], we focus on evaluating the potential of statistical model checking using the tool Cosmos. We develop a family of models, some designed by biochemist and some artificial schemes aimed to exhibit design challenges, and a range of quantitative requirements. The results are compared and contrasted with those obtained for the PRISM model checker using state-of-the-art numerical techniques (uniformisation and fast adaptive uniformisation [14,21]) and PRISM's statistical model checking implementation known as approximate model checking [34]. We conclude that statistical model checking significantly benefits from parallelisation and enables efficient analysis of much larger models at no great loss of accuracy compared to numerical methods. The paper ends by summarising future research challenges in the field.

2 Background on Molecular Walkers

DNA computing has so far mainly focused on designing logic circuits that perform computation *in vitro*, by transforming DNA strands using strand displacement systems as e.g. demonstrated experimentally in [44,41]. However, this approach has limitations, in that the strands are cascaded through a series of logic

gates in solution, which may lead to unintended interference [36] and consequently incorrect outcomes. An alternative approach is to design localised computation by 'printing' circuits on origami tiles as proposed in [41,9]. In this paper we focus on DNA walker systems [8,50,51], and particularly the programmable walkers of [48].

A DNA walker system consists of a *track* of strands, called *anchorages*, that are tethered to a DNA origami tile and traversed by a *walker* strand [48]. Origami tiles [43] are long circular single-stranded DNA scaffolds that can be folded into the tile shape with complementary short DNA strands that hybridize with the scaffold. The tracks can fork at junctions, and the walkers can be programmed to take a left or the right branch by selectively unblocking the anchorages that the walker can follow. Fig. 1 shows an example of a double-junction circuit that we will study later, where the two directions at the first junction are respectively labelled X and $\neg X$.

The stepping process is shown in Fig. 2. The walker, which carries a quencher (Q), is initially bound (hybridized) to the initial anchorage. The target anchorages at the end of the tracks have fluorophores (F) attached to them. After some initial preparation, the walker is able to autonomously step from one anchorage to another, eventually reaching the target anchorages and quenching the fluorophores. The programming of the tracks is achieved as follows. Initially, the anchorages are hybridized to the tile, and are either *unblocked*, meaning that the walker can bind to them, or *blocking*, that is, initially bound to a blocking strand that will prevent the walker from binding to them. Sections of the track can be reprogrammed by selectively unblocking them through the addition of strands that are complementary to the blocking strand; see Fig. 2 (Pane 2), where $\neg X$ is unblocked but X remains blocked.

After the walker is placed at the initial anchorage, a nicking enzyme is added to the solution. It binds to the walker-anchorage complex, melting the top of the anchorage away, which frees the top of the walker. This enables the walker strand to bind to the next anchorage through a displacement reaction (Panes 3 and 4 of Fig. 2). The process repeats, and the walker thus continues along the unblocked section of the track, through junctions, towards the final anchorage, where it reports the outcome by quenching the fluorophore.

Formally, the system can be viewed as a planar graph, composed from undirected tracks (consecutive anchorages) and gates (track junction points) that connect at most three tracks. [48] experimentally demonstrated that a walker can be directed to any leaf in a complete two-level binary tree by selectively unblocking the anchorages. In [16,15], we have studied the expressive power of DNA walker circuits implemented by this technology and showed that the circuits can compute any Boolean function through reduction to 3-CNF. Compared to DNA strand displacement systems in solution, an advantage of this technology is its spatial locality, but we have found that the undirected nature of the tracks imposes limitations on the use of parallelism. Other walker technologies

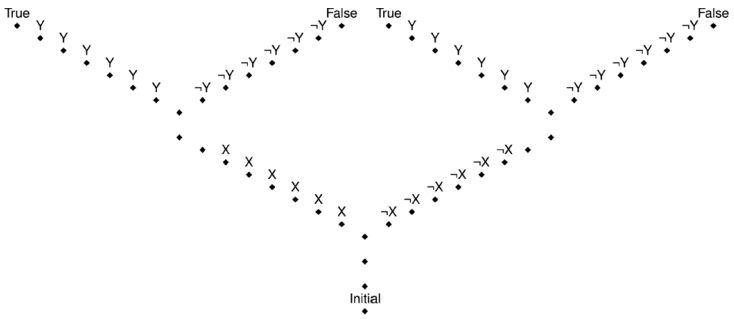

Fig. 1. A double-junction DNA walker circuit ([16]).

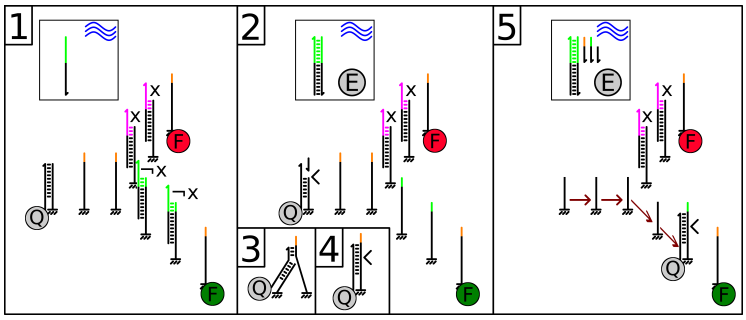

Fig. 2. The stepping action of the walker ([16])

that work with directed tracks have been demonstrated, so this limitation does not apply to all walker systems.

Computing Boolean functions using nanotechnologies such as the DNA walker systems has application in biosensing, for example to detect the presence of certain molecules. However, experiments have shown that the computation is quite unreliable, in the sense that walkers may release from a track, jump over two anchorages, or a blockade can fail to block an anchorage, delaying or diverting the walker to a different anchorage, thus returning wrong result. To study the reliability and performance of such systems, in [16,15] we also developed a stochastic model of DNA walker systems based on [8,48,50]. Note that, since we are considering localised computation, standard mass action kinetics which applies to well-mixed solution cannot be used, and we instead derive a model from experimental observations.

The model can be configured to a specific circuit layout, where we can vary the topology of the circuit, the number of anchorages in each section, and their physical spacing. We also model the different modes of anchorages, such as *blocked*, *unblocked*, *empty* or *bound* to the walker. The stepping process of Fig. 2 has been abstracted into a single walker step transition, taking the walker from one anchorage to the next. The rate of the stepping transition is dependent on

the distance between anchorages, and was derived using rate constants estimated in [50]. Maximum interaction distance was observed to be $d_M = 24$ nm. Taking into account the average distance between anchorages in the experiment of 6.2 nm, we have defined the walker stepping rate k to be a function of the distance d_a and the base rate k_s given by:

$$k = \begin{cases} k_s = 0.009\text{s}^{-1} & \text{when } d \leq 1.5d_a \\ k_s/50 & \text{when } 1.5d_a < d \leq 2.5d_a \\ k_s/100 & \text{when } 2.5d_a < d \leq d_M \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

This determines a sphere of reach of up to d_M around the walker-anchorage complex, within which the walker may step onto an uncut anchorage. We note that this abstraction of the stepping rate makes certain simplifying assumptions, such as we do not consider walker moving between intact anchorages or stepping backwards; these aspects have been experimentally observed and the model can be refined further in future.

One aspect that we do consider, however, and which also has been observed experimentally, is the failure of the blocking mechanism. We can allow the anchorages to spontaneously unblock and assume that the unblocking is uniform. If this happens, the walker may step onto such an unblocked anchorage and follow an incorrect track. The failure rate of 30% was estimated based on [48]. We have not modelled the failure of other mechanisms, such as missing anchorages or the failure of the reporting mechanisms, but these could again be added to the model.

In [16,15], we have constructed a family of models for a variety of circuits, fitting the rates from the single-junction circuit experiments [48], and then evaluating the quality of the model on the double-junction circuits. We found good alignment of model predictions with the experimental data, in particular also observing the effect of *leakage* transitions, that is, when the walker unintentionally transfers to the neighbouring track because of its proximity. In addition, we have considered a range of circuit designs and analysed their performance and reliability using the probabilistic model checker PRISM [34].

3 Stochastic Models and Analysis Techniques

As discussed in the previous section, stochasticity is an important aspect that we need to consider when designing molecular circuits, particularly localised computation such as DNA walkers placed on origami tiles. In this section we introduce the background notation and briefly overview existing stochastic models and analysis methods applicable to molecular systems.

3.1 Continuous-time Markov Chains (CTMC)

The evolution of molecular systems is naturally modelled as a stochastic process tracking the probability of molecular populations over time. This process,

under the assumption of constant volume and temperature, is a (homogeneous) *continuous-time Markov chain (CTMC)*.

Formally, a CTMC $\mathcal{C}$ is a tuple $(S, s_0, \mathbf{R})$, where S is a set of states, $s_0 \in [0,1]^S$ is the initial distribution over states, and $\mathbf{R}$ is the rate transition matrix with $\forall s, s' \in S,\ \mathbf{R}(s,s') \geq 0,\ \mathbf{R}(s,s) = 0$.

Each CTMC can be unfolded into execution paths from the start state as follows. The residence time in state s is exponentially distributed with exit rate $\lambda_s = \sum_{s' \in S} \mathbf{R}(s,s')$. Once the residence time expires, the probability to move to state s' from s is $\frac{\mathbf{R}(s,s')}{\lambda_s}$. We refer to the discrete-time Markov chain encoding the discrete transition probabilities for each state as the *embedded DTMC*.

Alternatively, for a CTMC the probability over time (*transient probability distribution*) is given by the Chemical Master Equation (CME) [23] $\frac{d}{dt}\pi_t = \pi_t \cdot \mathbf{Q}$, where $\mathbf{Q}$ is the infinitesimal generator matrix, defined as $\mathbf{Q}(s,s') = \mathbf{R}(s,s')$ if $s \neq s'$, and $1 - \sum_{s'' \neq s} \mathbf{R}(s,s'')$ otherwise, and $\pi_0 = s_0$.

A CTMC can be extended with a reward structure (ρ, ι), where ρ and ι are respectively a vector and matrix of non-negative reals. $\rho(s)$ is a state reward, and defines the rate at which the reward is acquired when $\mathcal{C}$ remains in state s for t time units. The function $\iota(s,s')$, $s, s' \in S$, defines the transition reward acquired each time the transition (s,s') occurs.

All Markov processes with countable state spaces and continuous distributions of time may be described as CTMCs, since the exponential distribution is the only continuous distribution with the Markov property. We usually work with finite state CTMCs, though some of the analysis techniques generalise to countable CTMCs.

In the context of verification CTMCs are enriched with *atomic propositions* that label states. Formally, a *labelled* CTMC is a tuple $(AP, L, S, s_0, \mathbf{R})$ such that $(S, s_0, \mathbf{R})$ is a CTMC, AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function that assigns atomic propositions to the states. When considering molecular systems, the set of atomic propositions usually includes inequalities over the number of each molecule type, for example, "there are at least 5 molecules x" (written $x \geq 5$).

3.2 Quantitative Verification for CTMCs

To specify quantitative properties of CTMCs, a number of formalisms can be used. These are divided into two families, linear time and branching time. Linear time formalisms specify accepting paths, and contain a single outer probabilistic operator. They include:

- temporal logics LTL (linear-time temporal logic) [40] and BLTL (bounded linear-time temporal logic) [28];
- deterministic timed automata specifications [19,6], where timed paths are accepted only if they are in the language of the automaton;
- deterministic linear hybrid automata specifications [5], an extension of timed automata with clocks evolving at different speed;

– temporal logic MTL (metric temporal logic) [10]which is an extension of BLTL with real-time constraints on the until operator.

Branching-time formalisms, on the other hand, contain nested probabilistic operators, and include:

– temporal logics PCTL [24] and PCTL* (probabilistic computation tree logic) [1], based on CTL/CTL* with the probabilistic operator added (for untimed properties);
– temporal logic CSL (continuous stochastic logic) [2], an extension of PCTL where temporal operators are equipped with real-time constraints and with an additional operator to specify steady state distribution.

These formalism can be extends with rewards. In this paper we will work with linear-time properties that we introduce using random variables. The key properties of interest are $\mathbb{P}_{=?}[X_\phi]$, the probability of the path formula ϕ being satisfied from a given state over time (X_ϕ is a random variable defined over paths from s equal to 1 if the path satisfies ϕ and 0 otherwise); and $\mathbb{E}_{=?}[X_{(\rho,\iota)}]$, the expected cumulative reward in a given state over time ($X_{(\rho,\iota)}$ is a random variable defined over paths annotated with rewards (ρ, ι) that computes the total reward cumulated up to t). Path formulas ϕ include the temporal operators 'until' and 'future', both unbounded and time-bounded variants; for example, $(x \geq 1)\mathbb{U}(x = 0)$ denotes a path along which molecules x eventually degrade, and $\mathbb{F}^{\leq 100}(x \geq 1) \wedge (y = 0)$ a path which reaches a state where there is at least one x molecule and no y molecules within 100 time units.

A number of techniques are available to analyse CTMCs. Since precise solution of the CME is in general intractable, the prevailing method is stochastic simulation, e.g. using the Gillespie algorithm [23], which generates forward trajectories from the initial state or distribution. Quantitative verification aims to compute the probability or expectations of certain events specified using the above temporal logic or automata formalisms. For CSL formulas, the computation of probability over time reduces to transient analysis on a modified model. Given an automaton representation of the property (which can be derived from LTL formulas or provided directly, e.g. as a timed automaton), it is necessary first to build the product of the automaton and the model, and then compute transient probability distributions on the product. Quantitative verification of expected reward properties is similar.

Transient analysis usually proceeds through numerical methods or simulation-based analysis known as statistical model checking, which we describe next.

3.3 Numerical Verification Methods for CTMCs

Numerical methods require that the state space and rate matrix of the CTMC be constructed. Typically, the numerical computation of transient distribution proceeds through discretisation of the CTMC, resulting in approximate probability values. These methods are more efficient on branching-time formalisms, in particular for CSL, where they take advantage of the strict alternation of

probabilistic and temporal operators. On linear-time properties one first has to build an automaton from the specification, and then build the product of the automaton and the CTMC, which increases the state space. Two methods have been developed for transient analysis, *uniformisation* and its variant *fast adaptive uniformisation* that neglects states with insignificant probability mass, thus improving performance of the computation.

Uniformisation Uniformisation (see e.g. [35] translates the problem of computing transient distribution of a CTMC to the computation of transient distribution of its discretisation, called the *uniformised DTMC*, and can be summarised as follows.

- For any CTMCs where there exists a bound on the maximal exit rate, the uniformised DTMC can be computed over the same state space, where each step in the DTMC corresponds to one exponentially distributed delay in the CTMC with rate equal to the maximal exit rate.
- Transient probability of the CTMC at time t can be computed as an infinite summation of i jumps in the uniformised DTMC weighted by Poisson probabilities.
- The Poisson weights of the infinite summation are derived using the Fox & Glynn algorithm [22], which also determines the upper bound on the number of summation terms needed to meet a given error bound.

Uniformisation involves operations on the stochastic matrix of the uniformised DTMC, and thus can suffer from state-space explosion.

Fast adaptive uniformisation (FAU) Fast adaptive uniformisation (FAU) [18,39] can reduce the size of the explored state space by neglecting states with insignificant probability mass. It is an approximate method for computing the transient distribution of a CTMC and can be summarised as follows:

- Transient probability distributions are computed forward from the initial state in the embedded DTMC. States with low probability of occurrence (below a threshold δ) are discarded. Therefore, FAU only explores a subset of the state space.
- The maximal exit rate of the CTMC is approximated and computed on the fly on the set of states that have not been discarded. The upper bound on the infinite summation is also computed on the fly.

The FAU method can greatly improve the time and memory consumption of the transient probability computation when the state space is large and exit rates of states span several orders of magnitude. It has also been extended to the computation of rewards [14].

Numerical model checking for CTMCs against CSL probability and reward formulas has been implemented in PRISM [34] using uniformisation (symbolic, hybrid, sparse and explicit engine) and FAU (explicit engine).

3.4 Statistical Model Checking

Instead of constructing the rate matrix of the CTMC, an alternative when dealing with large state spaces is to use statistical methods. The *statistical model checking* approach relies on Monte Carlo simulation algorithm to estimate the probability of interest. These methods are better suited to linear-time properties due to the difficulty of dealing with nested probabilistic operators. In [53] an algorithm for nested probabilistic operators is provided, but the simulation time greatly increases with the depth of the nesting. More precisely, statistical methods are applied as follows: for a path formula ϕ, a Bernoulli random variable X is defined which takes 1 as value on a path that satisfies ϕ and 0 otherwise. The probability estimate is thus obtained as the ratio of the number of paths satisfying ϕ to the total number of paths, where the Monte Carlo algorithm simulates a large number, say N, of paths. The random variable Z is defined as the mean of N independent copies of X:

$$Z = \frac{1}{N} \sum_{i=1}^{N} X_i$$

An advantage of statistical model checking is that it can be parallelised very easily: it suffices to run several simulators of the system on different processors and take the mean result of paths from all simulators. Particular attention needs to be paid to the random number generator to ensure that all generated paths are independent, but otherwise the overhead of parallelisation is low. Statistical model checking can be naturally extended to computing expected rewards, that is, the expected value of a random variable whose values depend on rewards cumulated over simulated paths.

Confidence intervals When one computes a probability estimate for some random variable X, and the exact value cannot be computed, it is important to know how far this estimate is from the actual value. Statistical methods cannot guarantee that the numerical value we obtain is at a given distance to the actual value. This is because using a fixed number of samples the simulation may avoid certain parts of the system with non-zero probability, thus biasing the estimation. Nevertheless, probabilistic guarantees on the obtained result can be given in the form of confidence interval, namely, confidence in the fact that the actual value is close enough to the realisation, defined as follows.

Let $(X_i)_1^N$ be independent random variables following a common distribution including a parameter θ. Let $0 < \gamma < 1$ be a confidence level. Then a *confidence interval* for θ with level at least γ is given by two random variables $l(X_1, \ldots, X_N)$ and $u(X_1, \ldots, X_N)$ such that for all θ:

$$\mathbb{P}\left[l(X_1, \ldots, X_N) \leq \theta \leq u(X_1, \ldots, X_N)\right] \geq \gamma$$

Classical statistical inequality can be used to derive confidence intervals from a set of realisations of a random variable. As a general rule, these inequalities

link together the confidence *level*, the *number* of samples and the *width* of the confidence interval, such that the user specifies two of them and the third is derived from the inequality.

The simplest approach is Gaussian analysis, which uses central limit theorem to approximate the distribution of the mean value of the observations of the random variable. When a random variable follows normal distribution, confidence interval can be computed using Gaussian error function.

A variant of this approach is to approximate the distribution of the random variable to a normal distribution and use the cumulative distribution function of the student-t distribution to compute confidence intervals. This confidence interval is more conservative than the one of the normal distribution when the number of samples is small, but as the number of samples increases they converge to each other.

When more conservative results are required and bounds on the value taken by the random variable are known, then Chernoff-Hoeffding inequality can be used to produce confidence intervals.

Sequential estimation Using additional hypotheses, the required number of samples can be computed on the fly; the simulation is then stopped as soon as the number of samples is sufficient. These methods are called *sequential estimation*.

Given a confidence level and a confidence interval width, the Chow and Robbin algorithm [12] requires the same hypothesis as Gaussian analysis and that the width of the confidence interval tends to zero. The algorithm provides the optimal stopping rules for the simulation and returns a confidence interval with expected width and variance.

When one is not interested in the actual expected value of a random variable but rather in deciding whether this value is above or below a threshold, then hypothesis testing can be used. Given a confidence level, a threshold and an open interval around this threshold called indifference region, the Sequential Probability Ratio Test (SPRT) [46,53] returns whether the value is above or below the threshold. If the true expected value is in the indifference region the test result has no probabilistic guarantee. Otherwise, the test result is correct with a probability equal to the confidence level. Two different confidence levels may be used for values above and below the threshold to make the test asymmetric (type I and type II errors).

Rare events One of the main limitations of statistical model checking is the rare event problem. When the probability that we want to compute is very small (usually smaller than 10^{-6}, statistical model checking becomes inefficient. Recently, several methods have addressed this limitation using *importance sampling* [26,7,42] or *splitting* [27], summarised below.

- Importance sampling relies on biasing the model such that the satisfaction of the formula is no longer a rare event. During simulation an the overall bias is estimated to produce an accurate estimation of the rare event. The difficulty lies in the choice of bias.

– Splitting relies on defining a sequence of successive embeddedings of subsets of the state space. The smallest subset only contains states satisfying the property of interest, while the largest contains all states. During simulation, each time a path reaches the next subset it is split into several copies. The probability that one path reaches the smaller subset is higher than the initial probability. Appropriately choosing the subsets is crucial to obtain precise results.

Statistical model checking for CTMCs has been implemented in a number of tools, to mention PRISM and Cosmos. In particular, PRISM [34] implements the confidence interval and SPRT methods for time-bounded CSL properties (also known as approximate probabilistic model checking), whereas Cosmos [3,4] provides a range of statistical model checking methods, including importance sampling, for a more expressive specification language.

3.5 Stochastic Petri Nets

CTMC models of molecular systems are complex, and high-level modelling languages facilitate their construction. Such languages include stochastic extensions of process algebras (cf PEPA [13]), reactive modules (cf PRISM [34]) or Petri nets (cf Cosmos [3,4]). We focus here on *stochastic Petri nets (SPNs)*, a graph of places connected by transitions, where the time for a transition to fire is distributed according to exponential distributions. SPNs are naturally interpreted as CTMCs whose state space is the set of reachable marking. They have been widely studied, for example in [17].

Formally, an SPN is a tuple $\mathcal{N} = (P, T, W^-, W^+, m_0, \Lambda)$, where P is a finite set of places, T is a finite set of transitions, $W^- : P \times T \to \mathbb{N}$ is the pre incidence matrix, $W^+ : P \times T \to \mathbb{N}$ is the post incidence matrix, $m_0 \in \mathbb{N}^P$ is the initial marking, and $\Lambda : \mathbb{N}^P \times T \to \mathbb{R}$ is the rate function which associates a rate to each marking and transition.

SPNs can be endowed with CTMC semantics, which for $\mathcal{N} = (P, T, W^-, W^+, m_0, \Lambda)$ is given as the CTMC $\mathcal{C} = (S, s_0, \mathbf{R})$ defined by:

- $S = Reach(\mathcal{N}, m_0)$, the set of reachable markings
- $s_0(m_0) = 1,\ \forall s \in S \backslash \{m_0\}, s_0(s) = 0$
- $\mathbf{R}(m, m') = \sum_{t \in T,\ s.t.\ m \xrightarrow{t} m'} \Lambda(m, t)$.

A common extension of SPNs are *generalized stochastic Petri nets (GSPN)*, which additionally use immediate transitions with Dirac distributions. Weights are added to resolve concurrent firing of immediate transitions. As immediate transitions are memoryless, the semantics of a GSPN is still Markovian as long as there is no cycle of immediate transitions [20]. Such cycles can be detected by an analysis of the structure of the net. Adding such immediate transitions is convenient for modelling stochastic systems and may reduce the size of the set of reachable markings [30].

GSPNs are supported by a number of tools, including Cosmos and Marcie [25].

4 Modelling DNA Walkers

In [16,15], DNA walkers were modelled in the native language of PRISM, which represents each walker circuit as a synchronised parallel composition of reactive modules, each specified using guarded commands whose updates are annotated with rates. Since DNA walker circuits are planar, GSPNs are well suited to their modelling, with the layout of the GSPN closely corresponding to the layout of the original circuit: the state of each anchorage is modelled using an independent place, while the steps of the walker are modelled with transitions that are exponentially distributed.

The blocking mechanism used to steer the movement of the walker may fail with some probability; this failure occurs before the walker is released and thus before any walker movement. In PRISM, this is modelled with transitions with very high rates (a billion times larger than walker movement rates), which makes the computation intractable when uniformisation is used. In GSPN this blocking mechanism is modelled using instantaneous transitions.

More precisely, each DNA walker circuit comprises several tracks (sequences of anchorages) and transitions correspond to walker taking a step from one anchorage to another nearby. The states of each anchorage are modelled as follows:

- Each anchorage is modelled with a single place, to preserve the layout of the original circuit placement. The relative placement of each place corresponds to that of the corresponding anchorage on the origami.
- Intact anchorages are modelled with places containing one token.
- Anchorages where the top has melted away are modelled by empty places.
- The anchorage to which the walker is attached is modelled by a place with two tokens.
- Blocked anchorages are modelled like anchorages where the top has melted away, with empty places.

Fig. 3 illustrates a transition encoding a displacement reaction between two anchorages a and b. Place a encodes the anchorage to which to walker is currently bound. Place b encodes an intact, unblocked anchorage. The transition consumes two tokens in the place corresponding to a and one token in the place corresponding to b, and produces two tokens in the place corresponding to b. Indeed, after the transition is fired the place corresponding to a is left empty, which models the anchorage where the top has melted away.

The walker may move between two anchorages that are sufficiently close. Each such movement is modelled with independent transitions. The rate of each transition depends on the distance between the two anchorages as specified on page 5.

Blocked anchorages do not initially contain tokens. In order to model the possibility of failure of the blocking mechanism, a place with initially one token and two immediate concurrent transitions are added to each blocked anchorage.

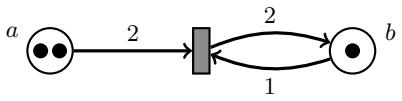

Fig. 3. Transition modelling movement of the walker from anchorage a to anchorage b.

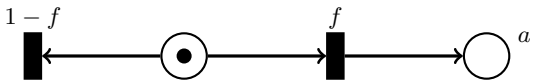

Fig. 4. Two transition model of the failure of the blocking mechanism of anchorage a

Fig. 4 illustrates this. With failure probability $f = 0.3$, a token is added to the place for anchorage a. In this case the anchorage is no longer blocked.

Our modelling approach ensures modularity of the design and that the layout of the Petri net closely resembles that of the original walker system. Different circuits can be composed together easily by merging together initial and final anchorages and by adding transitions between places encoding nearby transitions. Additional behaviours of the circuit, such as a missing anchorage, may be added easily by adding or removing tokens. An alternative modelling approach would have been to use colour to model the position of the walker, similarly to works in [37], but this would not preserve the layout of the original walker.

The walker models that we study are complex and have large number of transitions, but can nevertheless be viewed on screen and zoomed in. As an example, Fig. 5 shows one of the smallest models. In the following, for the sake of clarity we will hide most of the transitions when displaying a Petri net.

The state space of the walker systems increases exponentially with the number of anchorages. Table 1 shows the number of places and transitions, together with the size of the state space for models of DNA walkers that we studied. For the larger models it was not possible to build the state space due to memory limits.

The size of the state space makes the analysis of these models difficult. Qualitative analysis is still possible using symbolic representation of the state space for the smaller examples. For quantitative analysis, numerical computation of transient probability via uniformisation requires the storage of a vector of probabilities (one floating point number per state). Currently, the maximal amount of memory in a computer is in the order of 100 GByte using single precision, namely, at most $100 \cdot 1024 \cdot 1024 \cdot 1024/4 \approx 27 \cdot 10^9$ states. This neglects the amount of memory required to store the transition matrix, let alone the time to compute with such a large vector. Unfortunately, the state space of some of the models of DNA walkers that we consider exceeds this limit. This problem can be partially alleviated using FAU. However, when the state space is too large to

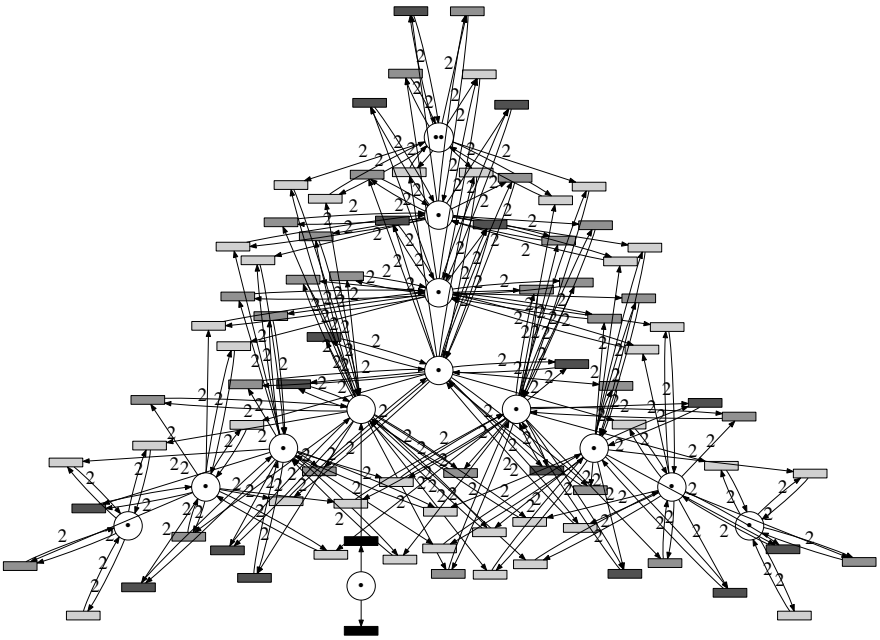

Fig. 5. A single junction circuit. The walker is initially in the upper anchorage. The anchorage on the left of the junction is blocked. Gray-scale used for the transitions indicate the tree possible rates $(k_s, k_s/50, k_s/100)$. Black transitions correspond to immediate transitions.

be constructed the only viable alternative is to use statistical model checking for the analysis.

5 Experiments and Results

In this paper, we model a variety of DNA walker circuits using stochastic Petri nets and analyse their reliability and performance. We focus on the application of statistical model checking, which we compare to numerical solution methods implemented in PRISM. We first briefly describe the tools used, followed by an overview of the results and analysis of the advantages of each method and the corresponding trade offs that can serve as guidelines when selecting software tools for quantitative modelling and verification of similar systems.

5.1 Tools

We use three modelling and analysis tools to perform computational experiments, namely Marcie [25], PRISM [34] and Cosmos [3,4]. We also use Graphviz for the visualisation of Petri nets.

Model	Places	Transitions	States	Model	Places	Transitions	States
control	8	34	172	ringLL	27	260	27,950,678
controlMissing1	7	22	50	ringRL	27	260	27,950,678
controlMissing2	6	13	13	ringLR	27	260	28,209,796
controlMissing7	7	27	82	ringRR	27	260	28,209,796
track12Block1	13	82	3,795	ringLLLarge	33	312	1,885,372,776
track12Block2	14	84	5,459	ringRLLarge	33	312	1,885,372,776
track12BlockBoth	14	84	5,248	ringLRLarge	33	312	1,860,879,029
track28LL	34	250	432,884,827	ringRRLarge	33	312	1,860,879,029
track28LR	34	250	435,340,831	redundantChoiceL	43	490	-
track28RL	34	250	435,340,831	redundantChoiceR	43	490	-
track28RR	34	250	432,884,827				

Table 1. Size of the state space of DNA walker models studied

Marcie supports qualitative and quantitative analysis of generalised stochastic Petri nets. The tool has been developed for the study of chemical reaction networks and thus facilitates the modelling of such systems. It employs Interval Decision Diagrams (IDD) to symbolically represent the state space of the Petri net. The implementation of IDD is mostly parallel, taking advantage of multicore architectures. The tool has recently been extended with a simulation engine for model checking of PLTL (propositional linear-time temporal logic) formulas. Marcie can deal with unbounded until properties as long as the user guarantees termination. We use Marcie to compute the size of the state space in our experiments.

PRISM is a probabilistic model checker that supports a variety of probabilistic models and probabilistic temporal logics, including CTMCs and temporal logic CSL. A CTMC model is provided in the PRISM modelling language as a synchronised parallel composition of reactive modules, but model imports, e.g. via SBML, are also supported. Verification of CSL properties can proceed via numerical methods (uniformisation or fast adaptive uniformisation) or statistical model checking (confidence interval and SPRT), known as approximate probabilistic model checking. We use PRISM to perform quantitative verification using numerical methods and approximate model checking.

Cosmos is a statistical model checker for generalised stochastic Petri nets with general distributions. It takes Hybrid Automata Stochastic Logic (HASL), based on linear hybrid automata, as a specification language. Efficient simulation is obtained using code generation that generates lightweight optimised C++ code. The generated code implements a simulator for the product of the model with the automaton underlying the specification. We use Cosmos for the evaluation of the models using statistical model checking.

5.2 The Setting

We perform experiments with Cosmos and PRISM on several circuit designs that have either been experimentally studied by biochemists or present design challenges. To ensure that the model given to each tool encodes the same system, the following workflow is used:

1. Each circuit defines a set of anchorages; for each anchorage, the position is specified as well as whether it is an initial or final anchorage. Additionally, the correct final anchorage is specified.
2. From this description of the circuit a GSPN is built.
3. The GSPN is exported in the GrML (Graph Markup Language) file format for Cosmos; the ANDL (Abstract Net Description Language) for Marcie; and the DOT language for Graphviz and the PRISM language. All these exports are simple except for PRISM, for which the GSPN is transformed into a single module, where each place is transformed into one variable and each transition into one guarded command. The property that the GSPN is 2-safe is used to bound variables.
4. From the GSPN and the initial description of the circuit, properties are provided as logical formulas for PRISM and as automata for Cosmos.

On each model we define atomic propositions of the form $Ax = y$, with Ax indicating an anchorage name, x either a place name or a PRISM variable, and y an integer. Labels and reward structures on states are also added to express certain properties, as described below.

For each model we perform the following initial analysis. The first four formulas are simple bounded reachability properties that can be expressed in many logics, for example BLTL or CSL with rewards. For the two remaining ones, we give the specification. We compute after 200 min the following properties:

1. The probability of reaching a deadlock state, where we assume an atomic proposition `deadlock` labelling deadlock states:

$$\text{Deadlock} := \mathbb{P}_{=?}\, \mathbb{F}^{\leq 12000}\, \texttt{deadlock}$$

2. The probability of reaching a final anchorage, define using a state label (`final`):

$$\text{Finish} := \mathbb{P}_{=?}\, \mathbb{F}^{\leq 12000}\, \texttt{final}$$

3. The average time spent in an anchorage that was supposed to be blocked but the failure mechanism failed; this is defined using reward structure `block` that increases linearly with the time spent in a blocked anchorage:

$$\text{Blockade} := \mathbb{R}\{\texttt{block}\}_{=?}\, C^{<=12000}$$

4. The expected number of steps of the walker, defined using reward structure `steps` that is increased by one for each firing of a transition:

$$\text{Steps} := \mathbb{R}\{\texttt{steps}\}_{=?}\, \mathbb{C}^{<=12000}$$

5. The reliability of the walker computation, written as an algebraic expression over (quantitative) CSL or BLTL formulas:

$$\text{Reliability} := \frac{\mathbb{P}_{=?}\, \mathbb{F}^{\leq 12000}\, \texttt{finalCorrect}}{\mathbb{P}_{=?}\, \mathbb{F}^{\leq 12000}\, \texttt{final}}$$

6. The probability that a path reaches the correct final anchorage while visiting a blocked anchorage, defined in BLTL using state label `blockAnchorage`:

$$\text{uB} := \mathbb{P}_{=?}\left((\mathbb{F}^{\leq 12000}\,\texttt{blockAnchorage}) \wedge (\mathbb{F}^{\leq 12000}\,\texttt{finalCorrect})\right)$$

The first five properties were already studied in [15] for some of the models.

The tool Cosmos takes deterministic Linear Hybrid Automata (LHA) as a specification formalism. Compared to timed automata, in LHA clocks are replaced by piecewise linear variables. Cosmos implements the synchronisation of a GSPN with an LHA. The main features of a property automaton are illustrated in Fig. 6. The locations of the automaton are labelled with invariants, which are atomic propositions of the GSPN. Locations are labelled with the rate of each variable; for clarity only rates different from 0 are labelled in the figure except for variable t, whose rate is always equal to 1. Accepting locations of the automaton are labelled with a name (in Fig. 6 names are **sl,fc,fnc**). The transitions of the automaton are of two types: *synchronized* transitions are labelled with a set of GSPN transition names, or the symbol A for any transition, and are synchronised with the firing of the GSPN transition. They can be labelled with a time guard; *autonomous* transitions, indicated with symbol #, are not synchronised and occur as soon as the time guard is satisfied. More detail on the synchronisation of GSPN and LHA can be found in [3].

In addition to the property automata, Cosmos relies on several HASL expressions that specify which value to estimate from the automaton. For our properties the expressions are as follows:

- The probability of reaching a deadlock state is expressed as follows:

$$\mathbb{P}_{=?}\,\mathbf{dl}$$

  and is interpreted as the probability for an accepting trajectory of the GSPN to end in location **dl**;
- The probability of reaching a final anchorage is expressed as follows:

$$\mathbb{P}_{=?}\,\mathbf{fc} \vee \mathbf{fnc}$$

- The average time spent in an anchorage that was supposed to be blocked, but the failure mechanism failed, is expressed as follows:

$$\mathbb{E}_{=?}\,bt$$

  and is interpreted as the average value of variable bt in accepting states;
- The reliability property is expressed as:

$$\frac{\mathbb{P}_{=?}\,\mathbf{fc}}{\mathbb{P}_{=?}\,\mathbf{fc} \vee \mathbf{fnc}}$$

- The probability that a path reaches the correct final anchorage while visiting a blocked anchorage is defined as:

$$\frac{\mathbb{P}_{=?}\,(ub = 1) \wedge \mathbf{fc}}{\mathbb{P}_{=?}\,\mathbf{fc}}$$

The sixth property is expressed with a slightly more involved automaton, which is not reported here for reasons of space. In this automaton a variable s is added to count the number of steps of the walker; it is increased by 1 on each synchronised transition except the one that loops over the initial state. The HASL expression is $\mathbb{E}_{=?}\, s$. More details and formal specification of HASL expressions can be found in [3].

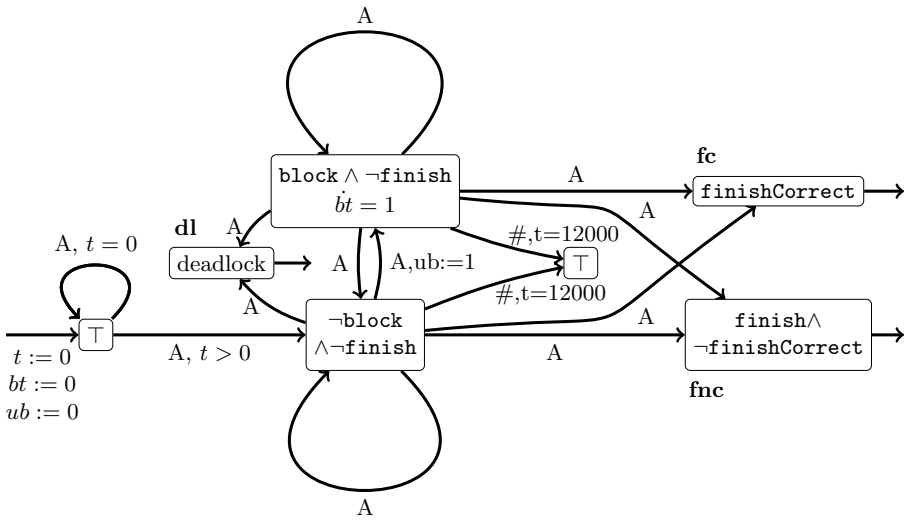

Fig. 6. LHA for the first four of the properties. This LHA contains two variables: t is a clock which is never reset, and bt is a piecewise linear variable with respect to time whose derivative is equal to 0 everywhere except in states where `block` holds. Symbol A indicates that the automaton reads any action of the model.

In the remainder of this section we describe the models and report on the results of verifying the above properties using statistical model checking methods. In Section 6 we compare and contrast the outcomes produced by the different tools and methods on each model, as well as the time and memory requirements.

5.3 Control Model

We begin by analysing the experimental designs of DNA walker models introduced in [52,47], which were modelled and analysed in [15,16]. These circuits comprise several anchorages in a straight line, where some anchorages have been removed. Fig. 7 depicts the control model where, for clarity, the transitions of the Petri net have been omitted except for the self-loops on final anchorages. The positioning of places corresponding to each anchorage is consistent with the

positioning of the anchorage on the circuit layout. The initial position of the walker is in the upper left corner and is encoded with two tokens. The final position is in the lower right corner. There are three variants of this model. In the first, the anchorage 4 is omitted. In the second, anchorages 4 and 5 are omitted. In the third anchorage, 7 is omitted.

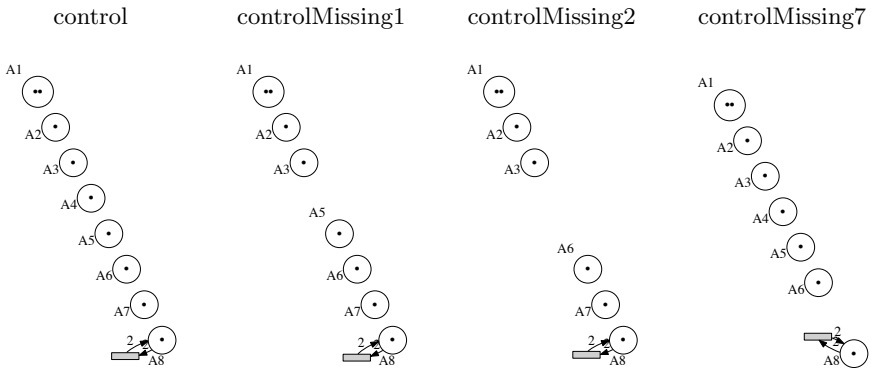

Fig. 7. Simplified Petri nets of control models.

The results for this control model are reported in Table 2. We compute the average number of steps (Steps), the probability of deadlock (Deadlock) and the probability to reach the final anchorage in 200 minutes (Finish).

Model	Steps	Deadlock	Finish
control	6.8756	0.0033	0.9618
controlMissing1	5.5141	0.0002	0.8528
controlMissing2	3.8529	0.0194	0.5909
controlMissing7	5.1453	0.0305	0.1755

Table 2. Experimental results for the control model

We observe that the number of steps is directly proportional to the number of anchorages, and the probability to reach the final anchorage within the time bound greatly decreases when an anchorage is missing. From these two observations, we can deduce that the predominant path in these models is the one that successively visits each anchorage, which is consistent with wetlab experiments [47] (Fig. 2).

5.4 Single Junction Circuit

The second set of models is a single junction circuit based on the experimental setup described in [49]. Blocked anchorages are used to steer the walker in a specific direction. Fig. 8 shows three model variants. In the first, one anchorage is blocked. The second model employs two blocked anchorages on the same branch, while the third blocks both branches.

For each branch, in the initial state all the anchorages are blocked. Unblocking DNA strands are added, which will selectively unblock anchorages on the designated branch.

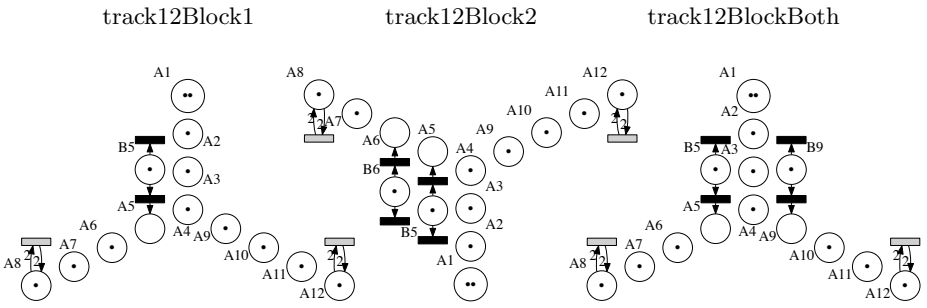

Fig. 8. Position and initial state of anchorage for the single junction circuit

Table 3 presents the results of our analysis for single junction models. We observe that the probability of deadlock (Deadlock) is very small, and with high probability the walker reaches a final anchorage within the time bound (Finish).

We observe that the model with two blockades is more reliable than that with only one: 0.84 instead of 0.77. In [16] (Fig. 11), an extensive study of the impact of blockade length on the reliability is presented.

For the third model, one of the final anchorages is chosen arbitrarily as the correct one.

Model	uB	Blockade	Steps	Deadlock	Finish	Reliability
track12Block1	0.1796	46.8615	7.0494	0.0009	0.9715	0.7746
track12Block2	0.2083	315.508	6.9504	0.0016	0.9592	0.8452
track12BlockBoth	0.4794	104.292	6.551	0.0007	0.9227	0.4999

Table 3. Experimental results for the single junction circuit

These results are consistent with those experimentally observed. In [49] (Fig. 2), for a single blockade the dependability of 0.76 is reported and for a double block-

ade the dependability of 0.87 is reported. When the two branches are blocked, no bias is observed between the two branches.

5.5 Two-Level Junction Circuit

These models are an extension of the junction circuit with two levels, which were also studied in the wetlab in [49] (Fig. 3).

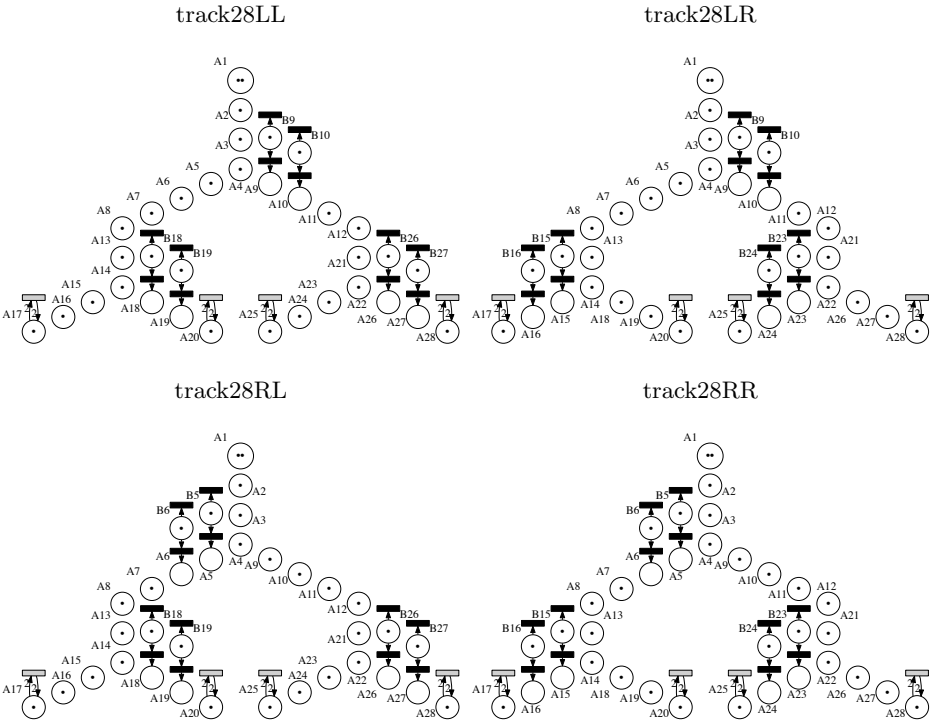

Fig. 9. Position and initial state of anchorage for the two level junction circuit

Table 4 shows numerical results. We observe that the reliability varies for the four different configurations. We note that that final anchorages reached on the outside of the model show greater reliability: 0.766 versus 0.7326. The other properties also show that the circuit with final anchorages on the inside are more likely to deadlock, that the walker performs more steps, spends more time in blocked anchorages, and that the probability for the walker to bind to a blocked anchorage before reaching the final location is higher. As expected by the symmetry, the two circuits with final anchorages on the outside (respectively

Model	uB	Blockade	Steps	Deadlock	Finish	Reliability
track28LL	0.3707	711.003	11.7339	0.0121	0.8959	0.7658
track28LR	0.3809	740.849	11.7688	0.0183	0.8847	0.7326
track28RL	0.3806	741.85	11.7684	0.0182	0.8853	0.7326
track28RR	0.3701	708.867	11.7363	0.012	0.8964	0.766

Table 4. Experimental results for the two-level binary tree

on the inside) have very similar results and the difference can be explained by the statistical error due to simulation. Fig. 10 shows the evolution of the probability of the presence of the walker on each final anchorage. This can be explained by the proximity of the two innermost tracks, which allows the walker to jump from one track to the other.

The plot of Fig. 10 corresponds to the wetlab experiments of [49] (Fig. 3) with similar qualitative results; the numerical value differs probably due to a different setting. The results in Fig. 10 have been computed statistically using Cosmos with $200,000$ simulations. The width of confidence intervals around each point of each graph is bounded by 0.006.

Even if this model can be built by composing three single junction circuits, the overall reliability cannot be computed as a composition of the reliability of the single junction, which will be equal for the four configurations. Thus, quantitative analysis of walker circuit cannot be performed at the level of each individual gate but has to be done at the global level.

5.6 Improving Reliability

In [16], the reliability of junction circuits is improved by increasing the length of blockades. We propose a different design based on a two-level junction with redundant choice and time constraints.

Results in Table 5 demonstrate that the reliability after 200 minutes does not increase compared to the single junction circuit, and in fact is even worse than the single junction circuit with two blockades. However, since paths which end in the incorrect final anchorage had to follow a longer path, in a short amount of time the reliability is much higher. In Figure 12 we plot the probability to reach each final anchorage over time T, as well as the reliability. At time $T = 1200s$ the reliability reaches 0.935, at the cost of fewer paths reaching the final anchorage. This demonstrates that timing constraints can play an important role in DNA computation designs, and can be used to produce small circuits with comparable reliability compared to those obtained by increasing blockade length.

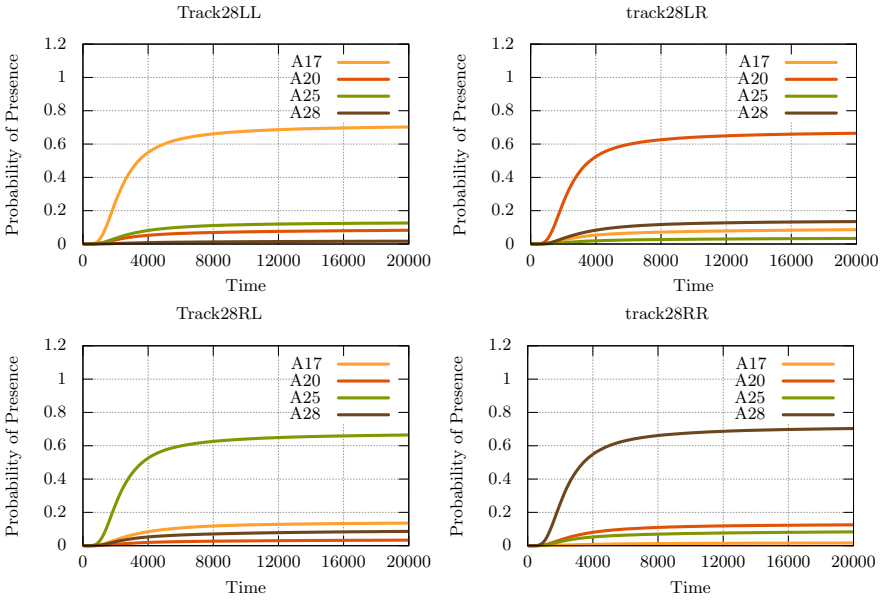

Fig. 10. Evolution over time of the probability of the presence of the walker on each final anchorage (Track28)

Model	uB	Blockade	Steps	Deadlock	Finish	Reliability
redundantChoice10	0.7296	1869.53	17.0304	0.2178	0.5554	0.7219
redundantChoice01	0.7298	1871.98	17.0307	0.2171	0.5559	0.7215

Table 5. Experimental results for the two-level choice

5.7 Exclusive Disjunction

This model implements the exclusive disjunction logical function (XOR) as a two-level junction circuit, where final anchoraged have been merged together forming a ring with the initial state in the middle (see Fig. 13). It illustrates the limits of increasing the length of tracks to improve the reliability. Fig. 13 shows two designs of the XOR function with different track length. The reliability does not always increase, as reported in table 6. First, note that the design is not symmetrical for the small ring, which explains the different results. The reliability increases (from 0.666 to 0.672) in two cases (LR) and (RR), but decreases (from 0.698 to 0.674) for (LL) and (RL). This model as been studied in [15] (Fig. 6) with various track lengths; for conciseness we report here only two different designs.

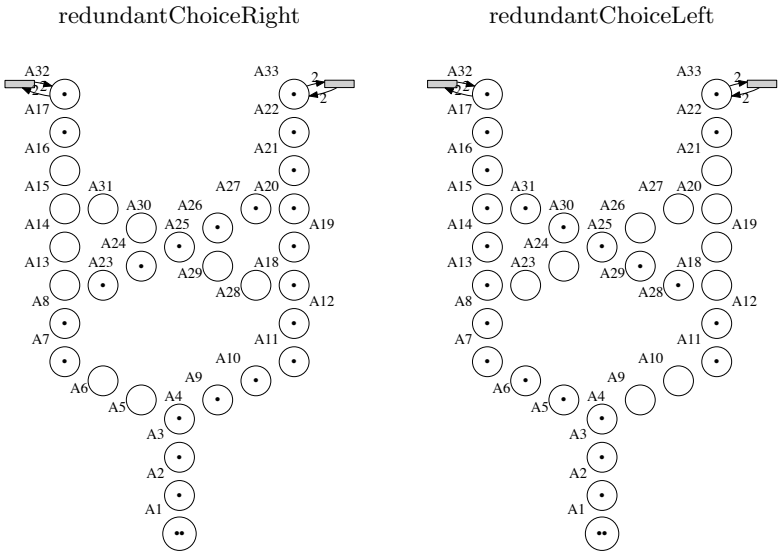

Fig. 11. Position and initial state of anchorage

6 Comparison of Model Checkers

In this section we compare the performance in time and memory of tools Cosmos and PRISM. All experiments have been conducted on a computer with "Core i7-2600" CPU with 4 cores and 8GB of RAM. The maximal execution time for each experiment is set to 20 hours.

When using PRISM we apply three different methods:

- PRISM Num: Numerical model checking using PRISM's hybrid engine, uniformisation and the default numerical solver for linear equation systems (Jacobi) with threshold $\varepsilon = 10^{-6}$.
- PRISM FAU: Numerical model checking using fast adaptive uniformisation (explicit engine) with $\delta = 10^{-10}$ and $\varepsilon = 10^{-8}$.
- PRISM Sim: Simulation-based approximate model checking that involves simulating $2,000,000$ paths with confidence level of 0.99.

For Cosmos we also use $2,000,000$ paths and 0.99 confidence level. This value allows us to obtain tight confidence intervals for the results we are interested in. Cosmos simulation times are reported with 1 thread (Cosmos 1T) and 8 threads (Cosmos 8T)[1] to show how statistical methods benefit from parallelisation.

[1] Experiments are performed on a machine with 4 cores and 2 threads per core. Experimentally, using 8 threads for the simulation is faster by around 30% than using 4 of them, whereas the speed up between 1 and 4 threads is around 370%.

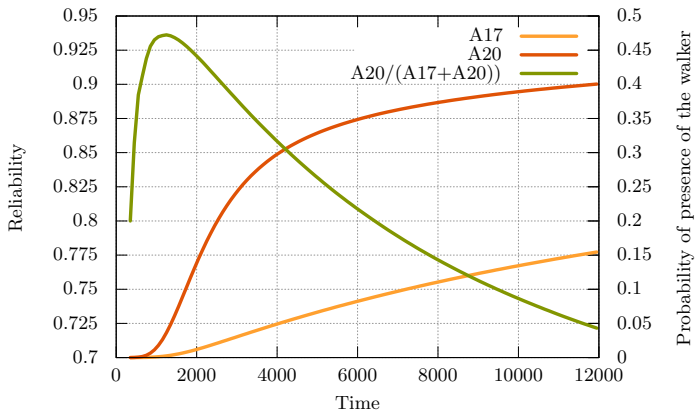

Fig. 12. Probability density function of the time required by the correct and incorrect path to reach final anchorages (redundantChoice).

Model	uB	Blockade	Steps	Finish	Reliability
ringLL	0.4509	607.693	7.8712	0.9273	0.6981
ringRL	0.4511	606.561	7.8714	0.9269	0.6975
ringLR	0.4527	664.121	7.651	0.8982	0.6661
ringRR	0.453	664.188	7.6565	0.8978	0.6665
ringLLLarge	0.429	584.601	9.7267	0.8622	0.6737
ringRLLarge	0.4291	585.6	9.7287	0.8624	0.6745
ringLRLarge	0.428	581.341	9.7391	0.865	0.6719
ringRRLarge	0.4283	579.661	9.7379	0.8651	0.6717

Table 6. Experimental results for the XOR models

Table 7 reports time and memory[2] requirement of the different methods. Memory is omitted for statistical methods, as it is negligible compared to numerical methods since there is no need to construct the matrix. All times are reported in seconds.

We observe that uniformisation takes a very long time except for the smallest model. This is due to a large summation bound computed by the Fox & Glynn algorithm. This is due to the following two factors:

- the time bound of properties (12000 time units) is very large;
- the PRISM language does not support instantaneous transitions, and thus instantaneous transitions are encoded using stochastic transitions with a large rate (1000000).

[2] The memory consumption for PRISM with the FAU method is measured with the Unix 'time' utility, which includes a large constant overhead due to the Java virtual machine GC. By comparison, the uniformisation method precisely reports the memory consumption.

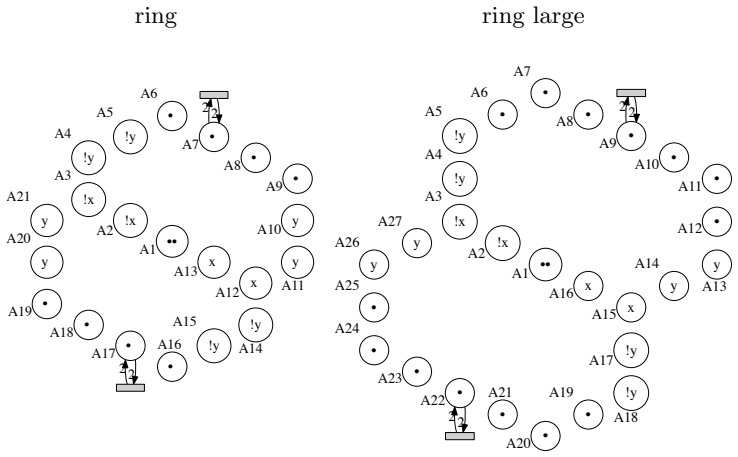

Fig. 13. Position and initial state of anchorage for XOR for a small and large design.

The FAU method, on the other hand, is the fastest on small models. Compared to standard uniformisation, which uses a very large time horizon in the uniformised DTMC, FAU performs a small number of iterations. This is due to the algorithm neglecting states that did not fire transitions that encode the failure of blockade after some time. As these states have transitions with large rates, neglecting them allows FAU to use smaller uniformisation constants, resulting in fewer iterations.

For the two-level junction circuits the statistical methods are faster, while uniformisation fails due to excessive time or memory requirements. Comparing the two statistical model checking tools, Cosmos achieved better runtimes. The parallel version of Cosmos, in particular, shows that statistical methods can take advantage of parallel architecture. The difference of runtime between the two statistical tools can be explained by two main factors.

- For small models, Cosmos is twice as fast as PRISM, which can be explained by the programming language of each tool. PRISM's simulator is written in Java, while that of Cosmos consists of C++ code generated from the model and compiled into a native executable. The choice of the language can be explained by the history of the two tools. PRISM was designed to perform numerical model checking, whereas Cosmos was designed from the start as a statistical model checker.
- Cosmos exploits the structure of Petri nets to generate code, which results in fast simulation performance. In Petri nets, all the possible events of a system are the firings of the transitions. In the PRISM language, events are firings of guarded transitions with possible synchronisations between component. To avoid checking whether each event is enabled, Cosmos analyses the Petri net structure to compute how transitions affect each other. This is used to produce a simulator that checks only a subset of the transitions after each

	Cosmos 8T	Cosmos 1T	Prism Sim	Prism Num		Prism FAU	
	Time	Time	Time	Time	Memory(KB)	Time	Memory(MB)
control	2.4	11.1	29.4	0.04	40.2	1	127.18
controlMissing1	1.5	6.8	18.9	0.01	7.8	0.9	98.82
controlMissing2	0.8	3.6	11.2	0	7.3	0.09	81.58
controlMissing7	1.9	8.6	22.6	0.02	26	0.92	113.76
track12Block1	4.7	22.4	68	>20H	388.3	3.95	245.11
track12Block2	4.7	22.2	76.4	-	530.8	4.84	296.03
track12BlockBoth	4.4	20.8	71.9	-	509.4	4.76	279.55
track28LL	10.6	51.5	395.5		>8GB	879.35	1,042.62
track28LR	11.1	53.3	393.8		-	927.59	1,061.37
track28RL	11.2	54.4	382		-	929.39	1,076.57
track28RR	10.6	52.2	393.3		-	870.48	1,047.32
ringLL	10.9	53.7	360	>20H	700MB	748.83	1,139.35
ringRL	11.1	55	339.1	-	700MB	755.69	1,131.96
ringLR	11	54.9	348.3	-	700MB	754.01	1,164.21
ringRR	10.9	53.4	346.3	-	700MB	757.72	1,154.43
ringLLLarge	12.7	62.7	423.7		>8GB	1,864.83	2,206.84
ringRLLarge	12.9	63.7	471.7		-	1,802.69	2,201.96
ringLRLarge	13	64.1	442		-	1,994.9	2,206.27
ringRRLarge	12.7	62.2	555.2		-	2,056.98	2,211.3
redundantChoice10	27.1	131.8	1,258.2		>8GB		>8GB
redundantChoice01	26.6	128.9	1,239.8		-		-

Table 7. Time and memory measurements for statistical and numerical methods.

firing of transitions and explains the difference of runtime of about a tenth on the larger models.

The analysis of the dependencies between transitions is easier to perform on Petri nets, as they can be expressed as graph properties. There is no theoretical limitation to adapt the same ideas in the PRISM simulator by building a dependency graph between commands, based on the content of their guard and updates.

Table 8 present comparison of the quality of results for statistical model checking and FAU. Two properties are used for the comparison: the expected number of steps before reaching a final anchorage (Steps) and the probability to reach the correct final anchorage (FinishCorrect). As numerical methods are not available due to time or memory constraints, only the statistical model checking methods can be compared with each other.

The results obtained by Cosmos and the approximate model checking procedure of PRISM are indistinguishable. Thanks to the large number of paths, the confidence intervals of the two tools converge to a very similar value.

For each property, we report the expected value for each method, as well as a measure of the error. For the statistical methods the error is measured with the width of the absolute confidence interval, which is reported. For the FAU method, the total probability lost is reported.

The two error bounds are of rather different nature, and thus only their order of magnitude can be compared. There are three distinct behaviours in the results:

	Steps				Finish Correct			
	Statistical		FAU		Statistical		FAU	
Model	Value	Width	Value	Lost P	Value	Width	Value	Lost P
control	6.876	1.62E-3	6.876	1.94E-7	0.962	6.98E-4	0.9618	1.88E-7
controlMissing1	5.514	3.53E-3	5.514	3.17E-8	0.853	1.29E-3	0.8528	3.17E-8
controlMissing2	3.853	5.20E-3	3.855	1.79E-8	0.591	1.79E-3	0.5918	1.79E-8
controlMissing7	5.145	1.56E-3	5.145	3.18E-8	0.175	1.39E-3	0.1751	2.58E-8
track12Block1	7.049	3.39E-3	7.049	2.19E-5	0.753	1.57E-3	0.7526	1.55E-5
track12Block2	6.95	2.97E-3	6.95	1.96E-5	0.811	1.43E-3	0.811	1.80E-5
track12BlockBoth	6.551	3.67E-3	6.552	2.17E-5	0.461	1.82E-3	0.4613	1.54E-5
track28LL	11.734	5.00E-3	11.707	9.49E-3	0.686	1.69E-3	0.6843	9.42E-3
track28LR	11.769	5.20E-3	11.74	1.00E-2	0.648	1.74E-3	0.6465	9.96E-3
track28RL	11.768	5.20E-3	11.74	1.00E-2	0.649	1.74E-3	0.6465	9.96E-3
track28RR	11.736	5.00E-3	11.707	9.49E-3	0.687	1.69E-3	0.6843	9.42E-3
ringLL	7.871	9.94E-3	7.869	9.40E-3	0.647	1.74E-3	0.643	7.87E-4
ringRL	7.871	9.94E-3	7.869	9.40E-3	0.647	1.74E-3	0.643	7.87E-4
ringLR	7.651	1.06E-2	7.65	1.08E-2	0.598	1.78E-3	0.5948	8.80E-4
ringRR	7.656	1.06E-2	7.65	1.08E-2	0.598	1.78E-3	0.5948	8.80E-4
ringLLLarge	9.727	1.40E-2	9.708	3.10E-2	0.581	1.80E-3	0.5725	4.81E-3
ringRLLarge	9.729	1.40E-2	9.708	3.10E-2	0.582	1.80E-3	0.5725	4.81E-3
ringLRLarge	9.739	1.40E-2	9.723	3.16E-2	0.581	1.80E-3	0.5716	4.89E-3
ringRRLarge	9.738	1.40E-2	9.723	3.16E-2	0.581	1.80E-3	0.5716	4.89E-3
redundantChoiceL	17.03	1.30E-2	15.709	0.205	0.401	1.79E-3	0.3374	0.205
redundantChoiceR	17.031	1.30E-2	15.709	0.205	0.401	1.79E-3	0.3374	0.205

Table 8. Comparison of the quality of result return by statistical and numerical methods.

- For the control and the single junction models, the FAU method is more precise by two orders of magnitude. The returned values for this model are almost indistinguishable, and the result returned by FAU is inside the confidence interval of the statistical methods.
- For the two-level junction models and exclusive disjunction, confidence interval width and lost probability have the same order of magnitude. Most of the time the value computed with FAU is smaller than the left bound of the confidence interval.
- For the remaining models, statistical methods are more precise, with significantly smaller confidence intervals width than probability lost, 0.0018 against 0.2. The value computed by FAU is significantly smaller than the left bound of the confidence interval, which is due to FAU neglecting small probability values during the computation.

7 Conclusion

We have analysed a range of DNA walker circuits against quantitative properties using the Cosmos tool, and established its usefulness as part of design automation technologies for molecular programming. Petri net models closely reflect the spatial designs of walker systems devised by experimentalists. The efficiency and

accuracy of the analysis, as well its alignment with experimental observations, has been demonstrated, improving over the numerical techniques implemented in PRISM for very large models.

However, there are significant challenges ahead for this field. These include programming languages and abstractions tailored to molecular programming and nanorobotics, which need to account for not just molecular kinetics, but also thermodynamics of molecular systems; scalability of the verification, for example via modular designs and compositional analysis; and synthesis techniques, including controller synthesis and circuit synthesis from quantitative specifications. Finally, integration of the verification and synthesis tools with molecular programming toolkits such as CADNANO is desirable.

Acknowledgements This paper has been supported by ERC Advanced Grant VERIWARE.

References

1. C. Baier. On algorithmic verification methods for probabilistic systems. Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.
2. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.
3. P. Ballarini, B. Barbot, M. Duflot, S. Haddad, and N. Pekergin. Hasl: a new approach for performance evaluation and model checking from concepts to experimentation. Technical Report LSV-15-04, LSV, ENS Cachan, March 2015.
4. P. Ballarini, H. Djafri, M. Duflot, S. Haddad, and N. Pekergin. Cosmos: A statistical model checker for the hybrid automata stochastic logic. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 143–144. IEEE, 2011.
5. P. Ballarini, H. Djafri, M. Duflot, S. Haddad, and N. Pekergin. HASL: An expressive language for statistical verification of stochastic models. In P. H. Samson Lasaulce, Dieter Fiems and L. Vandendorpe, editors, *Proceedings of the 5th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'11)*, pages 306–315, Cachan, France, May 2011. ICST.
6. B. Barbot, T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Efficient CTMC model checking of linear real-time objectives. In P. A. Abdulla and K. R. M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2011.
7. B. Barbot, S. Haddad, and C. Picaronny. Coupling and importance sampling for statistical model checking. In C. Flanagan, B. K. Flanagan, and B. König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2012.
8. J. Bath, S. J. Green, and A. J. Turberfield. A free-running DNA motor powered by a nicking enzyme. *Angewandte Chemie*, 44:4358–61, 2005.
9. H. Chandran, N. Gopalkrishnan, A. Phillips, and J. Reif. Localized hybridization circuits. *DNA Computing and Molecular Programming*, 6937:64–83, 2011.

10. T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Time-bounded verification of CTMCs against real time specifications. In *Proc. 8th International Conference, Formal Modeling and Analysis of Timed Systems*, volume 6919 of *LNCS*, pages 26–42. Springer, 2011.
11. Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Soloveichik, and G. Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.
12. Y. S. Chow and H. Robbins. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *The Annals of Mathematical Statistics*, pages 457–462, 1965.
13. F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009.
14. F. Dannenberg, E. M. Hahn, and M. Kwiatkowska. Computing cumulative rewards using fast adaptive uniformisation. In *Proc. 11th Conference on Computational Methods in Systems Biology (CMSB'13)*, 2013.
15. F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. DNA walker circuits: computational potential, design, and verification. In D. Soloveichik and B. Yurke, editors, *Proc. 19th International Conference on DNA Computing and Molecular Programming (DNA 19)*, volume 8141 of *LNCS*, pages 31–45. Springer, 2013.
16. F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. Dna walker circuits: Computational potential, design, and verification. *Natural Computing*, 2014.
17. M. Diaz. *Petri Nets: Fundamental models, verification and applications*. Wiley, 2010.
18. F. Didier, T. A. Henzinger, M. Mateescu, and V. Wolf. SABRE: A tool for stochastic analysis of biochemical reaction networks. In *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010*, pages 193–194, 2010.
19. S. Donatelli, S. Haddad, and J. Sproston. CSL TA: an expressive logic for continuous-time Markov chains. In *Quantitative Evaluation of Systems, 2007. QEST 2007. Fourth International Conference on the*, pages 31–40. IEEE, 2007.
20. C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. A semantics for every gspn. In *Application and Theory of Petri Nets and Concurrency*, pages 90–109. Springer, 2013.
21. E. M. H. F. Dannenberg and M. Kwiatkowska. Computing cumulative rewards using fast adaptive uniformisation. In *ACM Transactions on Modeling and Computer Simulation, Special Issue in Computational Methods in Systems Biology*, 2014.
22. B. L. Fox and P. W. Glynn. Computing poisson probabilities. *Commun. ACM*, 31(4):440–445, 1988.
23. D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 93555:2340–2361, 1977.
24. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.
25. M. Heiner, C. Rohr, and M. Schwarick. MARCIE - Model checking And Reachability analysis done effiCIEntly. In J. Colom and J. Desel, editors, *Proc. PETRI NETS 2013*, volume 7927 of *LNCS*, pages 389–399. Springer, June 2013.
26. C. Jégourel, A. Legay, and S. Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In *Computer Aided Verification*, pages 327–342, 2012.

27. C. Jegourel, A. Legay, and S. Sedwards. Importance splitting for statistical model checking rare properties. In *Computer Aided Verification*, pages 576–591. Springer, 2013.
28. S. K. Jha, E. M. Clarke, C. J. Langmead, A. Legay, A. Platzer, and P. Zuliani. A bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology, 7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings*, pages 218–234, 2009.
29. C. Jung and A. D. Ellington. Diagnostic applications of nucleic acid circuits. *Accounts of Chemical Research*, 2014. To appear.
30. D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with generalized stochastic Petri nets*. Wiley, 1994.
31. M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449–458. ACM Press, September 2007.
32. M. Kwiatkowska. Challenges in automated verification and synthesis for molecular programming. In M. Abadi, P. Gardner, A. D. Gordon, and R. Mardare, editors, *Essays for the Luca Cardelli Fest*, volume MSR-TR-2014-104 of *Technical Report*, pages 155–170. Microsoft Research, 2014.
33. 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.
34. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
35. M. Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *SFM*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer, 2007.
36. 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*, 9:1470–1485, 2012.
37. F. Liu and M. Heiner. Colored petri nets to model and simulate biological systems. In *Proceedings of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010) and of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010), Braga, Portugal, June, 2010*, pages 71–85, 2010.
38. M. K. M. Ceska, F. Dannenberg and N. Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *Proc. 12th Conference on Computational Methods in Systems Biology (CMSB'14)*, volume 8859 of *LNCS*, pages 86–89. Springer, 2014.
39. M.-E.-C. Mateescu. *Propagation Models for Biochemical Reaction Networks*. PhD thesis, 2011.
40. A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.
41. L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.
42. D. Reijsbergen, P.-T. de Boer, W. Scheinhardt, and B. Haverkort. Automated rare event simulation for stochastic Petri nets. In *Quantitative Evaluation of Systems*, pages 372–388. Springer, 2013.

43. P. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440:297–302, 2006.
44. G. Seelig, D. Soloveichik, D. Zhang, and E. Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314:1585–1588, 2006.
45. 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.
46. A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 06 1945.
47. S. F. Wickham, M. Endo, Y. Katsuda, K. Hidaka, J. Bath, H. Sugiyama, and A. J. Turberfield. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature nanotechnology*, 6(3):166–169, 2011.
48. S. F. J. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. J. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature nanotechnology*, 7:169–73, 2012.
49. S. F. J. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. J. Turberfield. A dna-based molecular motor that can navigate a network of tracks. *Nat Nano*, 7(3):169–173, 03 2012.
50. S. F. J. Wickham, M. Endo, Y. Katsuda, K. Hidaka, J. Bath, H. Sugiyama, and A. J. Turberfield. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature nanotechnology*, 6:166–9, 2011.
51. P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif. A unidirectional DNA walker that moves autonomously along a track. *Angewandte Chemie International Edition*, 43:4906–4911, 2004.
52. P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif. A unidirectional dna walker that moves autonomously along a track. *Angewandte Chemie International Edition*, 43(37):4906–4911, 2004.
53. H. Younes and R. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368–1409, 2006.
54. B. Yurke, A. Turberfield, A. Mills, F. Simmel, and J. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406(6796):605–8, 2000.
55. D. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand displacement reactions. *Nature Chemistry*, 3:103–113, 2011.
56. 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.