A

Computing Cumulative Rewards using Fast Adaptive Uniformisation

FRITS DANNENBERG, University of Oxford, Department of Computer Science
ERNST MORITZ HAHN, University of Oxford, Department of Computer Science[1]
MARTA KWIATKOWSKA, University of Oxford, Department of Computer Science

The computation of transient probabilities for continuous-time Markov chains often employs uniformisation, also known as the Jensen's method. The fast adaptive uniformisation method introduced by Mateescu et al. approximates the probability by neglecting insignificant states, and has proven to be effective for quantitative analysis of stochastic models arising in chemical and biological applications. However, this method has only been formulated for the analysis of properties at a given point of time t. In this paper, we extend fast adaptive uniformisation to handle expected reward properties which reason about the model behaviour until time t, for example, the expected number of chemical reactions that have occurred until t. To show the feasibility of the approach, we integrate the method into the probabilistic model checker PRISM and apply it to a range of biological models. The performance of the method is enhanced by the use of interval splitting. We compare our implementation to standard uniformisation implemented in PRISM and to fast adaptive uniformisation without support for cumulative rewards implemented in MARCIE, demonstrating superior performance.

ACM Reference Format:
Frits Dannenberg, Ernst Moritz Hahn and Marta Kwiatkowska. 2013. Computing Cumulative Rewards using Fast Adaptive Uniformisation. *ACM Trans. Model. Comput. Simul.* V, N, Article A (January YYYY), 23 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Model checking of continuous-time Markov chains (CTMCs) [Baier et al. 2003] is an established method that has been successfully used for quantitative analysis of a variety of models, ranging from biochemical reaction networks [Heath et al. 2008; Mateescu 2011] to performance analysis of computer systems [Baier et al. 2010]. The analysis typically involves computing the *transient probability* of the system residing in a state at a given time t, or, for a model annotated with rewards, the *expected reward* that can be obtained. Transient probabilities for finite-state CTMCs can be computed through the *uniformisation* method, also known as the Jensen's method. Uniformisation involves discretising the CTMC with respect to a fixed rate, which enables reduction of the transient probability calculation to an infinite summation of Poisson distributed

[1]Current address: State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences

The authors are supported in by the ERC Advanced Grant VERIWARE, a Microsoft Research PhD Studentship (FD), a Chinese Academy of Sciences Fellowship (Grant No. 2013Y1GB0006), and the research fund for International Young Scientists (Grant No. 61350110518). We would like to thank Taolue Chen and Andrew Phillips for useful discussion, and Chris Thachuk for his help in preparing the DSD models.
Correspondence may be addressed to Marta.Kwiatkowska@cs.ox.ac.uk.

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

steps of the derived discrete-time Markov chain, and approximating the probability by truncating to a finite summation. The number of terms required can be precomputed for a specified precision using the Fox-Glynn method [Fox and Glynn 1988].

Many biochemical reaction networks, however, induce CTMC models whose state space is potentially infinite. To handle such cases, [Mateescu 2011] introduced *continuous-time propagation models*, a generalisation of continuous-time Markov chains. The idea of this model is to propagate the (probability or expectation) mass values along the system execution. In order to analyse propagation models, which can be infinite, the *fast adaptive uniformisation (FAU)* method, a generalisation of adaptive uniformisation [van Moorsel and Sanders 1994], was formulated [Mateescu et al. 2010]. Similarly to standard uniformisation, FAU applies discretisation, except that it does so dynamically, starting from some initial condition wrt to a sequence of rates (a birth process) rather than a single rate, and truncates the computation of the probability to a finite summation, although the number of summation terms cannot be precomputed. To deal with the unbounded state space, FAU explores only the relevant states, ignoring the probability of the insignificant states. Thus, the number of states to be maintained in memory can be kept small, at a cost of some loss of precision. Importantly, the FAU method can also speed up the analysis of very large finite models.

Fast adaptive uniformisation was implemented [Didier et al. 2010] and applied successfully on a variety of biological systems [Didier et al. 2010; Mateescu 2011], but for transient probabilities and a limited set of reward-based properties only. Many useful quantitative analyses involve the computation of expected rewards, which can be *instantaneous* (incurred at time t) or *cumulated* (until time t). An example of an instantaneous reward property is the number of molecules of a given species at time $100s$, and of a cumulative property the expected number of reactions that occurred for the duration of $100s$. Although one can express cumulative reward properties by adding additional species to the model, for example by increasing the reward by 1 every time a reaction occurs, this has the disadvantage of introducing an additional dimension into the model and, as we show later, can severely affect the performance, resulting in higher memory requirement and a consequent loss of precision.

In this paper, we extend fast adaptive uniformisation for CTMCs to allow for efficient computation of cumulative reward properties, thus avoiding the overhead of adding the additional dimension. We cast our results in the framework of propagation models of [Mateescu 2011]. We implement the method, including the reward extension, and integrate it into the probabilistic model checker PRISM [Kwiatkowska et al. 2011]. To show the practical applicability of FAU, we have applied it to a range of case studies from biology, demonstrating superior performance compared to existing techniques. We also implement *interval splitting* [van Moorsel and Wolter 1998], which involves applying FAU to smaller sub-intervals instead of a single, large interval, which is known to benefit uniformisation and the analysis of stiff models. Finally, we benchmark our tool against the implementation of FAU in the tool MARCIE and discuss the differences in performance.

1.1. Related Work

FAU generalises adaptive uniformisation [van Moorsel and Sanders 1994] by accelerating the discretisation and neglecting states with insignificant probability. Standard uniformisation is implemented in a number of tools, including PRISM, which we enhance with the FAU functionality in this paper. SABRE [Didier et al. 2010] is the first tool to implement FAU, without cumulative rewards. Both PRISM and SABRE support models written in Systems Biology Markup Language (SBML) as input, in addition to their native modelling languages. SABRE is a stand-alone tool available for download or as a web interface; it additionally offers deterministic approxima-

tions using differential equations (by the Runge-Kutta fourth order method), which is faster and leads to accurate results for large numbers of molecules. PRISM does not support deterministic approximations, but provides extensive support for temporal logic model checking, including both probabilistic and reward properties [Kwiatkowska et al. 2007]. These are appropriate for molecular networks where some species occur in small numbers, or the encoding of spatial information is needed, as we consider in this paper. PRISM also provides statistical model checking and Gillespie simulation [Kwiatkowska et al. 2011]. The tool MARCIE [Schwarick et al. 2011] implements FAU, but does not support cumulative rewards for the FAU method. In addition to FAU, MARCIE also supports a symbolic method called Interval Decision Diagrams, which is more efficient than Binary Decision Diagrams in some cases [Schwarick and Heiner 2009], but not yet integrated with their FAU implementation. Further tools that support reward properties but not FAU include, for instance, MÖBIUS [Clark et al. 2001] and MRMC [Katoen et al. 2011]. MÖBIUS allows one to express measures closely related to the ones we consider here, such as the average, instantaneous, cumulative, or time-averaged values, or their variance. However, MÖBIUS and MRMC also support measures that are distinct from the ones we consider here, namely, reward-bounded properties, where the probability of paths whose total reward value must be below a given bound.

A related approach to the analysis of transient properties of CTMCs with large state spaces is finite state projection (FSP) [Munsky and Khammash 2006], which has been specifically designed for biological models. In this approach, similarly to FAU, the starting point is a set of initial states that have a significant probability at time zero. Then, the FSP algorithm adds states which are reachable from the current set of states. In its simplest version, these states are just the new states reachable within one transition from the current set of states. However, more elaborate ways of deciding which states to add are possible. The state successor computation is repeated until the probability that, within the given time bound, one reaches states outside the current set of states is negligible. Thus, an approximation of the transient property of interest is computed in the finite projection of the state space obtained this way.

In contrast to FAU, the FSP does not discard states that have negligible probability. The disadvantage of this is that potentially many more states have to be stored in memory. On the other hand, not discarding states allows for a simpler representation of state to state transitions. It also allows one to handle nested CSL formulas [Hahn et al. 2009; Spieler et al. 2014], for which FAU does not seem appropriate. The reason is that, when discarding a state, the information about which subformulae of the CSL formula under consideration are fulfilled is lost. This information might, however, be required to decide the validity of the formula under consideration.

One can also apply interval splitting, a method to divide the analysis for a given time bound t into several analyses for time bounds $t_1, \ldots, t_n$ with $\sum_i t_i = t$ [Munsky and Khammash 2007]. This way, one can discard states with negligible probabilities after the analysis of each t_i, so as to reduce memory consumption in a way similar to how this is done in FAU. Further improvements of the method have been considered in [Munsky and Khammash 2008; Tapia et al. 2012].

2. PRELIMINARIES

We begin by giving an overview of the main definitions and results based on [Kwiatkowska et al. 2007; Didier et al. 2010; Mateescu 2011]. A *continuous-time Markov chain (CTMC)* is given by a set of discrete states S and the transition rate matrix $\mathcal{R}\colon S \times S \to \mathbb{R}_{\geq 0}$ where $\mathcal{R}(s,s) = 0$ for all $s \in S$. The rate $\mathcal{R}(s,s')$ determines the delay before the transition can occur, i.e. the probability of this transition being

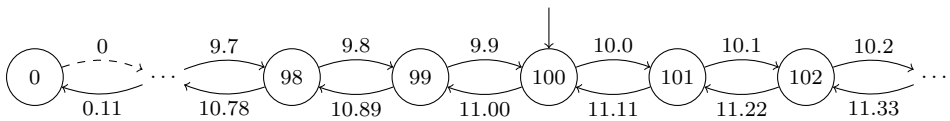

Fig. 1: Birth-death process.

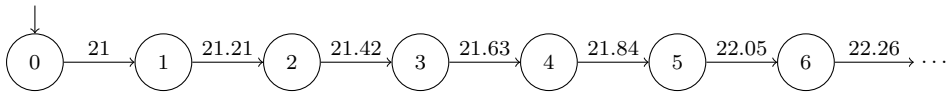

Fig. 2: Birth process derived from the CTMC in Fig. 1.

triggered within t time-units is $1 - e^{-\mathcal{R}(s,s')\cdot t}$. Let $E(s) \stackrel{\text{def}}{=} \sum_{s' \in S} \mathcal{R}(s, s')$ be the exit rate and define the generator matrix Q by $Q \stackrel{\text{def}}{=} R - \mathrm{diag}(E)$, where $\mathrm{diag}(E)$ is the $S \times S$ matrix with E on its diagonal and zero everywhere else. Then $\pi_t \colon S \to \mathbb{R}_{\geq 0}$, the transient probability vector at time t, can be expressed as $\pi_t = \pi \cdot e^{Qt}$ given the initial probability vector π.

We cast our method in the framework of *continuous-time (linear) propagation models* [Mateescu 2011, Section 2.3.3], which generalise continuous-time Markov chains. We now recall the relevant results from [Mateescu 2011].

Definition 2.1 (*Continuous-time propagation model*). A *continuous-time propagation model (CTPM)* is a tuple $\mathcal{M} = (S, \pi, \mathcal{R})$, where

— S is a countable or finite set of *states*,
— $\pi \colon S \to \mathbb{R}_{\geq 0}$ where $|\{s \in S \mid \pi(s) > 0\}| < \infty$ is an *initialisation vector*, and
— $\mathcal{R} \colon S \times S \to \mathbb{R}_{\geq 0}$ is a *transition matrix*, such that for all $s \in S$ we have $|\{s' \in S \mid \mathcal{R}(s, s') > 0\}| < \infty$.

The transition matrix $\mathcal{R}$ assigns a rate $\mathcal{R}(s, s')$ to each pair of states, as for CTMCs, and the initialisation vector π assigns an initial mass value $\pi(s)$ to each state $s \in S$. There are only finitely many states to which a positive mass is assigned initially. The models we consider are *finitely branching*, that is, for each state there are only finitely many states to which this state has a positive transition rate.

A CTPM is a CTMC if $\sum_{s \in S} \pi(s) = 1$ and $\mathcal{R}(s, s) = 0$ for all $s \in S$.

Example 2.2 (*Continuous-time Markov chain*). In Fig. 1, we depict a CTPM, a so-called *birth-death process* [Evans et al. 2008]. Each state s is a natural number describing the number s of molecules of a given species. In each state s, a new molecule can appear with rate $\lambda \cdot s$, and disappear with rate $\mu \cdot s$ for $\lambda \stackrel{\text{def}}{=} 0.1, \mu \stackrel{\text{def}}{=} 0.11$. We thus have $\mathcal{R}(s, s+1) \stackrel{\text{def}}{=} \lambda \cdot s$ for all $s \geq 0$, $\mathcal{R}(s, s-1) \stackrel{\text{def}}{=} \mu \cdot s$ for $s \geq 1$ and $\mathcal{R}(\cdot, \cdot) \stackrel{\text{def}}{=} 0$ otherwise. We assume that $\pi(100) \stackrel{\text{def}}{=} 1$ and for the other states $\pi(\cdot) \stackrel{\text{def}}{=} 0$. Thus, the model is a CTMC.

To reason about the timed behaviour of a CTPM, we now define its *generator matrix* which generalises that for CTMCs.

Definition 2.3 (*Generator matrix*). The *generator matrix* $Q(\mathcal{M}) \colon S \times S \to \mathbb{R}$ of a CTPM $\mathcal{M}$ is defined so that

— $Q(\mathcal{M})(s, s') \stackrel{\text{def}}{=} \mathcal{R}(s, s')$ for $s, s' \in S$ with $s \neq s'$, and
— $Q(\mathcal{M})(s, s) \stackrel{\text{def}}{=} \mathcal{R}(s, s) - \sum_{\substack{s' \in S, \\ s' \neq s}} \mathcal{R}(s, s')$.

The *propagation process*, which propagates probability mass or expectation values, is then defined as follows. Note that, for CTMCs, $\pi_t(s)$ is the (transient) probability that the model resides in state s at time t.

Definition 2.4 (*Propagation process*). Given a CTPM $\mathcal{M} = (S, \pi, \mathcal{R})$, the *propagation process at time* t, $\pi_t(\mathcal{M})\colon S \to \mathbb{R}$, is defined as the solution of the differential equation

$$\dot{\pi}(s') \stackrel{\text{def}}{=} \sum_{s \in S} \pi(s) \cdot Q(\mathcal{M})(s, s')$$

at time t, for $s' \in S$, given the initial value π.

The *standard uniformisation* [Jensen 1953] method for CTMCs splits the CTMC into a discrete-time Markov chain (DTMC) and a Poisson process as follows. Define the DTMC P by $P \stackrel{\text{def}}{=} I + \frac{1}{\Lambda} \cdot Q$ where Λ is the uniformisation rate such that $\Lambda \geq \max_{s \in S} E(s)$. Then π_t can be computed as $\sum_{n=0}^{\infty} \pi_t(\Psi^\Lambda)(n) \cdot \tau_n(\mathcal{M})$, where $\pi_t(\Psi^\Lambda)(n)$ is the value of the Poisson distribution with rate $\Lambda \cdot t$ at point n, and $\tau_n(\mathcal{M}) = \tau_{n-1}(\mathcal{M}) \cdot P$ for $n > 0$, $\tau_0(\mathcal{M}) = \pi_0$. For a given precision ϵ, the summation can be truncated using the Fox-Glynn method [Fox and Glynn 1988].

The *fast adaptive uniformisation (FAU)* [Didier et al. 2010; Mateescu 2011] is a variant of the *adaptive uniformisation* [van Moorsel and Sanders 1994], which splits the CTMC into a DTMC and a birth process. For an infinite sequence $\mathbf{\Lambda} = (\Lambda_0, \Lambda_1, \ldots)$ of rates with $\Lambda_n \in \mathbb{R}_{\geq 0}$ for all $n \in \mathbb{N}$, the *birth process* is defined as the CTMC $\Phi^{\mathbf{\Lambda}} \stackrel{\text{def}}{=} (S, \pi, \mathcal{R})$, where

— $S \stackrel{\text{def}}{=} \mathbb{N}$,
— $\pi(0) \stackrel{\text{def}}{=} 1$ and $\pi(\cdot) \stackrel{\text{def}}{=} 0$ otherwise, and
— $\mathcal{R}(n, n+1) \stackrel{\text{def}}{=} \Lambda_n$ for $n \in \mathbb{N}$ and $\mathcal{R}(\cdot, \cdot) \stackrel{\text{def}}{=} 0$ otherwise.

Note that the Poisson process is a special case of the birth process with constant rates $\mathbf{\Lambda} = (\Lambda, \Lambda, \ldots)$.

Transient probabilities of birth processes can be approximated efficiently using specialised techniques [Mateescu 2011, Section 4.3.2, Solution of the birth process]. This is possible by applying *standard uniformisation* [Jensen 1953] in a way which takes advantage of the particular structure of the process. Finally, transient probabilities of general CTPMs can be computed as follows, where we reformulate P_n in terms of the rate matrix, rather than the generator matrix used in [Mateescu 2011].

THEOREM 2.5 (SOLVING PROPAGATION MODELS USING A BIRTH PROCESS). *Consider*

— *a CTPM* $\mathcal{M} = (S, \pi, \mathcal{R})$,
— *an infinite sequence of subsets* $\mathbf{S} = (S_0, S_1, \ldots)$ *with* $S_n \subseteq S$ *denoting* active states,
— *an infinite sequence* $\mathbf{\Lambda} = (\Lambda_0, \Lambda_1, \ldots)$ *with* $\Lambda_n \geq \sup_{s \in S_n} \sum_{\substack{s' \in S, \\ s' \neq s}} \mathcal{R}(s, s')$ *of* uniformisation rates,
— probability matrices $P_n(\mathcal{M})\colon S \times S \to \mathbb{R}_{\geq 0}$ *for* $n \in \mathbb{N}$, *where for* $s, s' \in S$ *we have*

$$P_n(\mathcal{M})(s, s') \stackrel{\text{def}}{=} \begin{cases} \frac{\mathcal{R}(s,s')}{\Lambda_n} & \textit{if } s \neq s', \textit{ and} \\ \frac{\mathcal{R}(s,s')}{\Lambda_n} + 1 - \sum_{\substack{s'' \in S, \\ s'' \neq s}} \frac{\mathcal{R}(s,s'')}{\Lambda_n} & \textit{otherwise,} \end{cases}$$

— discrete-time distributions $\tau_n(\mathcal{M})\colon S \to \mathbb{R}_{\geq 0}$ *for* $n \in \mathbb{N}$ *with*

$$\tau_n(\mathcal{M})(s') \stackrel{\text{def}}{=} \begin{cases} \pi(s') & \textit{if } n = 0\textit{, and} \\ \sum_{s \in S} \tau_{n-1}(\mathcal{M})(s) \cdot P_{n-1}(s, s') & \textit{otherwise.} \end{cases}$$

We further require that $\{s \in S \mid \tau_n(\mathcal{M})(s) > 0\} \subseteq S_n$ *for* $n \in \mathbb{N}$.

Then we have that, at time t*, for each* $s \in S$*:*

$$\pi_t(\mathcal{M})(s) = \sum_{n=0}^{\infty} \pi_t(\Phi^{\mathbf{\Lambda}})(n) \cdot \tau_n(\mathcal{M})(s).$$

The *fast adaptive uniformisation* method [Mateescu 2011] builds on the result above and works as follows. Starting with the initial distribution at step $n = 0$, at each step n the FAU explores a subset $\hat{S}_n$ of the states S_n. The sets $\hat{S}_n$ are constructed by taking $\hat{S}_{n-1}$, adding the successor states $\{s' \in S \mid \exists s \in \hat{S}_{n-1}.\ \mathcal{R}(s, s') > 0\}$ of this set, and discarding states s with $\tau_n(\mathcal{M})(s) < \delta$, where δ is a fixed precision threshold. This process is repeated until step m, for instance so that $(1 - \sum_{n=0}^{m} \pi_t(\Phi^{\mathbf{\Lambda}})(n)) < \varepsilon$. Thus, we add the probability from the birth process at each step, and stop the state space exploration as soon as this sum is at least $1-\varepsilon$. In contrast to standard uniformisation, where the Fox-Glynn [Fox and Glynn 1988] algorithm can be utilised, we do not have an a priori step bound, but are still able to decide in a straightforward way when the infinite sum can be safely truncated.

Definition 2.6 (*Fast Adaptive Uniformisation*). Let $\mathcal{M}$, $\mathbf{S} = (S_0, S_1, \ldots)$, and $\mathbf{\Lambda} = (\Lambda_0, \Lambda_1, \ldots)$ be as in Theorem 2.5. Further, consider

— a *truncation point* $m \in \mathbb{N}$,

— a finite sequence of subsets $\hat{\mathbf{S}} = (\hat{S}_0, \ldots, \hat{S}_m)$ with $\hat{S}_n \subseteq S_n$ denoting *active states* for $n \in \{1, \ldots, m\}$,

— *probability matrices* $\hat{P}_n(\mathcal{M})\colon S \times S \to \mathbb{R}_{\geq 0}$ for $n \in \{0, \ldots, m\}$ where

$$\hat{P}_n(\mathcal{M})(s, s') \stackrel{\text{def}}{=} \begin{cases} P_n(\mathcal{M})(s, s') & \text{if } s \in \hat{S}_n\text{, and} \\ 0 & \text{otherwise,} \end{cases}$$

— *discrete-time distributions* $\hat{\tau}_n(\mathcal{M})\colon S \to \mathbb{R}_{\geq 0}$ for $n \in \mathbb{N}$ with

$$\hat{\tau}_n(\mathcal{M})(s') \stackrel{\text{def}}{=} \begin{cases} \pi(s') & \text{if } n = 0\text{, and} \\ \sum_{s \in S} \hat{\tau}_{n-1}(\mathcal{M}) \cdot \hat{P}_{n-1}(s, s') & \text{otherwise.} \end{cases}$$

We define the *fast adaptive uniformisation (FAU)* value at time t for each $s \in S$ as

$$\hat{\pi}_t(\mathcal{M}, \hat{\mathbf{S}}, \mathbf{\Lambda})(s) \stackrel{\text{def}}{=} \sum_{n=0}^{m} \pi_t(\Phi^{\mathbf{\Lambda}})(n) \cdot \tau_n(\mathcal{M})(s).$$

Example 2.7 (*Fast Adaptive Uniformisation*). We sketch how one can perform FAU for the CTMC from Example 2.2 according to Definition 2.6 and Theorem 2.5: only for state $s = 100$ the initial distribution is positive, so we can use $S_0 \stackrel{\text{def}}{=} \{100\}$. Then, we use $S_n \stackrel{\text{def}}{=} \{\max\{0, 100 - n\}, \ldots, n\}$ and $\Lambda_n \stackrel{\text{def}}{=} (\lambda + \mu) \cdot n$. We chose these sets in such a way that S_n contains all states which can have a positive probability in the discrete-time probability distribution $\tau_n(\mathcal{M})$. The corresponding birth process is sketched in Fig. 2.

In Fig. 3, we depict S_n together with the relevant parts of the matrices $P_n(\mathcal{M})$ for $n = 0, 1, 2, \ldots, \infty$ (rounding-off the numbers). For each step n, we label each state s with $\tau_n(\mathcal{M})(s)$. For $n = 0$ we have $\tau_n(\mathcal{M})(100) = 1$ and $\tau_n(\mathcal{M})(s) = 0$ for all $s \neq 100$ according to the initial distribution of the Markov chain. We have $P_0(\mathcal{M})(100, 99) =$

$\frac{\mathcal{R}_0(\mathcal{M})(100,99)}{\Lambda_0} = \frac{11}{10+11} \approx 0.524$, $P_0(\mathcal{M})(100,101) \approx 0.476$, $P_0(\mathcal{M})(100,100) = \frac{0}{10+11} + 1 - (\frac{10}{10+11} + \frac{11}{10+11}) = 0$. According to the definition of S_0, we have $P_0(\mathcal{M})(s,s') = 0$ otherwise. Therefore, we have $\tau_1(\mathcal{M})(99) \approx 1 \cdot 0.524 = 0.524$, $\tau_1(\mathcal{M})(101) \approx 0.476$, and $\tau_1(\mathcal{M})(s) = 0$ otherwise. The subsequent probability distributions are computed in the same way. For instance, we have $\tau_2(\mathcal{M})(100) = \tau_1(\mathcal{M})(100) \cdot P_1(\mathcal{M})(100,100) + \tau_1(\mathcal{M})(99) \cdot P_1(\mathcal{M})(99,100) + \tau_1(\mathcal{M})(101) \cdot P_1(\mathcal{M})(101,100) \approx 0 \cdot 0.010 + 0.524 \cdot 0.467 + 0.476 \cdot 0.524 \approx 0.493$.

For $t = 0.1$ we provide the transient probabilities $\pi_t(\Phi^{\mathbf{\Lambda}})(n)$ of being in the nth state of the birth process. We also provide the first n summands of $\pi_{0.1}(\mathcal{M})(100) = \sum_{n=0}^{\infty} \pi_{0.1}(\Phi^{\mathbf{\Lambda}})(n) \cdot \tau_n(\mathcal{M})(100)$ (probability to reside in state 100 at time $t = 0.1$). For instance, we have $\sum_{n=0}^{2} \pi_{0.1}(\Phi^{\mathbf{\Lambda}})(n) \cdot \tau_n(\mathcal{M})(100) \approx 0.122 \cdot 1 + 0.254 \cdot 0 + 0.267 \cdot 0.493 \approx 0.254$.

State $s = 0$ is absorbing, that is, once entered it cannot be left, and rates leading to a decrease in molecule count are higher than those leading to an increase. Thus, in the last line ("Step ∞") we see that, as n increases, the probability concentrates on the state $s = 0$. Thus, we can discard states with a high number of molecules from the reduced state sets $\hat{S}_n$, while retaining a sufficient amount of the total probability.

We now define instantaneous rewards, which can be used to express expected reward properties incurred at a given time. We annotate the models with state rewards.

Definition 2.8 (*State reward structure*). A *state reward structure* for a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$ is a function $\mathbf{r}\colon S \to \mathbb{R}_{\geq 0}$.

Definition 2.9 (*Instantaneous rewards*). Consider a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$ with state reward structure $\mathbf{r}\colon S \to \mathbb{R}_{\geq 0}$ and a time point $t \in \mathbb{R}_{\geq 0}$. The *instantaneous reward* is defined as

$$\mathcal{I}_t(\mathcal{M}, \mathbf{r}) \stackrel{\text{def}}{=} \sum_{s \in S} \pi_t(\mathcal{M})(s) \cdot \mathbf{r}(s).$$

We show that instantaneous rewards can be easily accommodated within the FAU method, and we can approximate the expected mass value by terminating the state space exploration using a criterion similar to the probability mass calculation in [Mateescu 2011].

Definition 2.10 (*Instantaneous reward approximation*). Let $\mathcal{M}$, $\mathbf{S} = (S_0, S_1, \ldots)$, and $\mathbf{\Lambda} = (\Lambda_0, \Lambda_1, \ldots)$ be as in Theorem 2.5 and let $\hat{\mathbf{S}}$ be as in Definition 2.6. Then we define

$$\mathcal{I}_t(\mathcal{M}, \mathbf{r}, \hat{\mathbf{S}}, \mathbf{\Lambda}) \stackrel{\text{def}}{=} \sum_{s \in S} \hat{\pi}_t(\mathcal{M}, \mathbf{S}, \mathbf{\Lambda})(s) \cdot \mathbf{r}(s).$$

COROLLARY 2.11 (ERROR BOUNDS FOR FAU). *Consider a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$ for which we have the uniformisation rates $\mathbf{\Lambda}$ (cf. Theorem 2.5) and a state reward structure $\mathbf{r}$. Consider $m \in \mathbb{N}$, $\hat{\mathbf{S}}$, and $\hat{\tau}(\mathcal{M})$ as in Definition 2.6. Set $\mathbf{\Lambda} \stackrel{\text{def}}{=} (\Lambda, \Lambda, \ldots)$. Then if*

$$\max_{s \in S} \mathbf{r}(s) \cdot \left(1 - \sum_{n=0}^{m} \pi_t(\Phi^{\mathbf{\Lambda}})(n)\right) < \frac{\epsilon}{2} \textit{ and } \max_{s \in S} \mathbf{r}(s) \cdot \left(1 - \sum_{s \in \hat{S}_m} \hat{\tau}_m(\mathcal{M})(s)\right) < \frac{\epsilon}{2}$$

it follows that

$$\mathcal{I}_t(\mathcal{M}, \mathbf{r}) - \mathcal{I}_t(\mathcal{M}, \mathbf{r}, \hat{\mathbf{S}}, \mathbf{\Lambda}) < \epsilon.$$

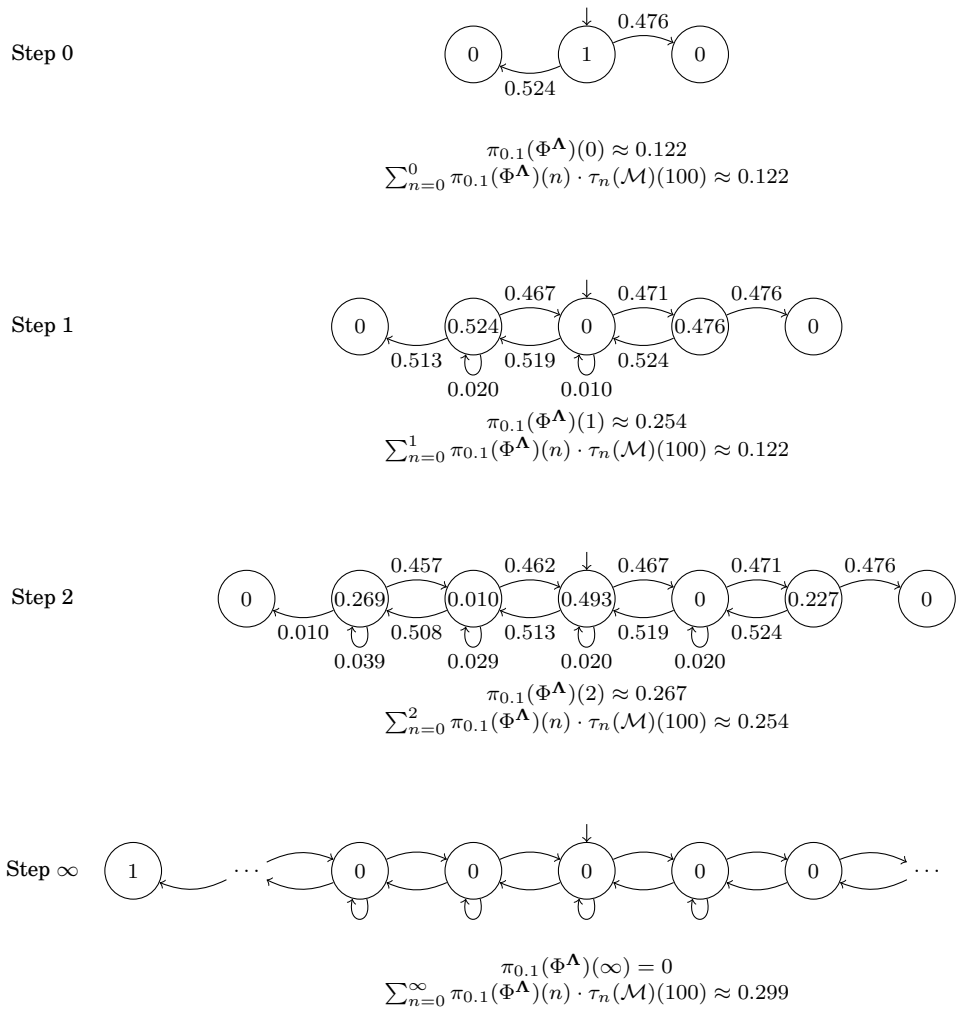

Fig. 3: Applying Fast Adaptive Uniformisation to the CTMC from Fig. 1.

PROOF. Part of the expected reward value is lost due to the approximation of the infinite sum. This is accounted for by first inequality. By discarding states while exploring the state space, we lose further mass. This is accounted for by the second inequality. Adding up the maxima of the two errors, we can bound the error. □

Example 2.12 (*FAU for Instantaneous Rewards*). Consider the CTMC from Example 2.2 with a reward structure $\mathbf{r}$ assigning to each state s the number of molecules s. When using the transient probability values computed in Example 2.7, for the computation of the exact values we have $\mathcal{I}_{0.1}(\mathcal{M}, \mathbf{r}) = \sum_{s \in S} \hat{\pi}_t(\mathcal{M}, \mathbf{S}, \mathbf{\Lambda})(s) \cdot \mathbf{r}(s) \approx \ldots + 100 \cdot 0.299 + \ldots \approx 99.900$.

3. CUMULATIVE REWARDS

In this section, we extend the FAU method to reason about properties of the behaviour of a CTMC model cumulating the rewards *until* a given point of time. The correct-

ness of the method is proved using the framework of CTPMs [Mateescu 2011], where cumulative rewards were not considered.

For a given CTMC, we first extend its state space by adding *time-accumulating* states to remember how much time was spent in a specific state, and then, noting that the time-extended CTPM is not a CTMC, show how the expected reward computation can be approximated.

Definition 3.1 (*Time-extended CTPM*). Given a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$, the *time-extended CTPM* is defined as

$$\mathrm{ext}(\mathcal{M}) \stackrel{\text{def}}{=} (S_{\mathrm{ext}}, \pi_{\mathrm{ext}}, \mathcal{R}_{\mathrm{ext}}), \text{ where}$$

— $S_{\mathrm{ext}} \stackrel{\text{def}}{=} S \uplus S_{\mathrm{acc}}$, where for each $s \in S$ we have exactly one corresponding *time-accumulating* $s_{\mathrm{acc}} \in S_{\mathrm{acc}}$, that is, $S_{\mathrm{acc}} \stackrel{\text{def}}{=} \{s_{\mathrm{acc}} \mid s \in S\}$,
— $\pi_{\mathrm{ext}}(s) \stackrel{\text{def}}{=} \pi$ for $s \in S$ and $\pi_{\mathrm{ext}}(\cdot) \stackrel{\text{def}}{=} 0$ otherwise, and
— the transition matrix $\mathcal{R}_{\mathrm{ext}} \colon S_{\mathrm{ext}} \times S_{\mathrm{ext}} \to \mathbb{R}_{\geq 0}$ is defined such that, for $s_{\mathrm{ext}}, s'_{\mathrm{ext}} \in S_{\mathrm{ext}}$, we have

$$\mathcal{R}_{\mathrm{ext}}(s_{\mathrm{ext}}, s'_{\mathrm{ext}}) \stackrel{\text{def}}{=} \begin{cases} \mathcal{R}(s_{\mathrm{ext}}, s'_{\mathrm{ext}}) & \text{if } s_{\mathrm{ext}}, s'_{\mathrm{ext}} \in S \text{ and } s_{\mathrm{ext}} \neq s'_{\mathrm{ext}}, \\ 1 & \text{if } s_{\mathrm{ext}} = s \in S \text{ and } s'_{\mathrm{ext}} \in \{s_{\mathrm{ext}}, s_{\mathrm{acc}}\}, \\ 0 & \text{otherwise.} \end{cases}$$

We now use time-extended CTPMs to prove the central theorem of the paper. This is achieved by first approximating the residence time, and then the cumulative reward, by considering the reward per time unit of residing in a given state. We use the *mixed birth process probability* $\Psi^{\mathbf{\Lambda}}(n) = \frac{1}{\Lambda_n} \cdot \sum_{i=n+1}^{\infty} \pi_t(\Phi^{\mathbf{\Lambda}})(i)$, which denotes the probability that more than n state changes happen within time t in the birth process, divided by the nth uniformisation rate. This is used to collect the time *spent* in a given state, as opposed to the probability $\pi_t(\Phi^{\mathbf{\Lambda}})(i)$ *to be in* a state at a given point of time.

THEOREM 3.2 (RESIDENCE TIME). *Consider a CTMC* $\mathcal{M} = (S, \pi, \mathcal{R})$ *and let* $\rho_t(\mathcal{M}) \colon S \to \mathbb{R}_{\geq 0}$ *be defined as*

$$\rho_t(\mathcal{M})(s) \stackrel{\text{def}}{=} \int_0^t \pi_u(s)\,\mathrm{d}u$$

for $s \in S$. *Then we have*

$$\rho_t(\mathcal{M})(s) = \sum_{n=0}^{\infty} \Psi^{\mathbf{\Lambda}}(n) \cdot \tau_n(\mathcal{M})(s),$$

for $s \in S$, $\tau(\mathcal{M})$ *and* $\mathbf{\Lambda}$ *as in Theorem 2.5 and* $\Psi^{\mathbf{\Lambda}}(n) \stackrel{\text{def}}{=} \frac{1}{\Lambda_n} \cdot \sum_{i=n+1}^{\infty} \pi_t(\Phi^{\mathbf{\Lambda}})(i)$.

PROOF. Assume $\mathrm{ext}(\mathcal{M}) = (S_{\mathrm{ext}}, \pi_{\mathrm{ext}}, \mathcal{R}_{\mathrm{ext}})$. Then, by definition of the structure of the time-extended CTPM, we have for $s \in S$ that

$$\pi_t(\mathcal{M})(s) = \pi_t(\mathcal{M}_{\mathrm{ext}})(s), \text{ and } \rho_t(\mathcal{M})(s) = \pi_t(\mathcal{M}_{\mathrm{ext}})(s_{\mathrm{acc}}).$$

By the structure of the time-extended CTPM, we obtain

$$\tau_0(\mathcal{M}_{\mathrm{ext}})(s_{\mathrm{acc}}) = 0,$$

$$\tau_{n+1}(\mathcal{M}_{\mathrm{ext}})(s_{\mathrm{acc}}) = \frac{1}{\Lambda_n} \cdot \tau_n(\mathcal{M}_{\mathrm{ext}})(s) + \tau_n(\mathcal{M}_{\mathrm{ext}})(s_{\mathrm{acc}}) = \frac{1}{\Lambda_n} \cdot \tau_n(\mathcal{M})(s) + \tau_n(\mathcal{M}_{\mathrm{ext}})(s_{\mathrm{acc}})$$

and thus

$$\tau_n(\mathcal{M}_{\text{ext}})(s_{\text{acc}}) = \sum_{i=0}^{n-1} \frac{1}{\Lambda_i} \cdot \tau_i(\mathcal{M})(s).$$

From this and by Theorem 2.5 we have

$$\begin{aligned}
\pi_t(\mathcal{M}_{\text{ext}})(s_{\text{acc}}) &= \sum_{n=0}^{\infty} \pi_t(\Phi^{\mathbf{\Lambda}})(n) \cdot \tau_n(\mathcal{M}_{\text{ext}})(s_{\text{acc}}) \\
= \sum_{n=0}^{\infty} \pi_t(\Phi^{\mathbf{\Lambda}})(n) \cdot \sum_{i=0}^{n-1} \frac{1}{\Lambda_i} \cdot \tau_i(\mathcal{M})(s) &= \sum_{n=0}^{\infty} \sum_{i=0}^{n-1} \frac{1}{\Lambda_i} \cdot \tau_i(\mathcal{M})(s) \cdot \pi_t(\Phi^{\mathbf{\Lambda}})(n) \\
= \sum_{i=0}^{\infty} \sum_{n=i+1}^{\infty} \frac{1}{\Lambda_i} \cdot \tau_i(\mathcal{M})(s) \cdot \pi_t(\Phi^{\mathbf{\Lambda}})(n) &= \sum_{i=0}^{\infty} \left(\frac{1}{\Lambda_i} \cdot \sum_{n=i+1}^{\infty} \pi_t(\Phi^{\mathbf{\Lambda}})(n) \right) \cdot \tau_i(\mathcal{M})(s) \\
&= \sum_{i=0}^{\infty} \Psi^{\mathbf{\Lambda}}(i) \cdot \tau_i(\mathcal{M})(s) = \sum_{n=0}^{\infty} \Psi^{\mathbf{\Lambda}}(n) \cdot \tau_n(\mathcal{M})(s). \qquad \square
\end{aligned}$$

The above theorem splits the behaviour of a CTMC into a birth process and a discrete-time process that determines the time spent in specific states of the CTMC. Thus, we can now apply the FAU method to compute cumulative reward properties. To do this, we accumulate rewards for being in a state over time. We stress that time-extended CTPMs are used here only in the proof, and never constructed in our method.

Transition rewards $\mathbf{r}_t \colon S \times S \to \mathbb{R}_{\geq 0}$ are obtained for moving from one state to another. We do not explicitly consider transition rewards for CTMCs here. However, given state rewards $\mathbf{r}$ and transition rewards $\mathbf{r}_t$, we can define cumulative reward rates $\mathbf{r}'$ as $\mathbf{r}'(s) \stackrel{\text{def}}{=} \mathbf{r}(s) + \sum_{s' \in S} \mathcal{R}(s, s') \cdot \mathbf{r}_t(s, s')$. For the properties under consideration, this new reward structure is equivalent to using transition rewards, as shown in [Kwiatkowska et al. 2006, Equation 6].

Definition 3.3 (*Cumulative rewards*). Consider a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$ with state reward structure $\mathbf{r}$: $S \to \mathbb{R}_{\geq 0}$ and a time duration $t \in \mathbb{R}_{\geq 0}$. The *cumulative reward value* is defined as

$$\mathcal{C}_t(\mathcal{M}, \mathbf{r}) \stackrel{\text{def}}{=} \int_0^t \sum_{s \in S} \pi_u(\mathcal{M})(s) \cdot \mathbf{r}(s) \, \mathrm{d}u.$$

We now use the results from Theorem 3.2 to compute the reward obtained until a given point of time. The following corollary follows directly from Theorem 3.2 and can be used to approximate cumulative rewards.

COROLLARY 3.4 (COMPUTING REWARDS). *For a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$ with state reward structure $\mathbf{r}$: $S \to \mathbb{R}_{\geq 0}$ and a time duration $t \in \mathbb{R}_{\geq 0}$, we have*

$$\mathcal{C}_t(\mathcal{M}, \mathbf{r}) = \sum_{s \in S} \rho_t(\mathcal{M})(s) \cdot \mathbf{r}(s) = \sum_{n=0}^{\infty} \sum_{s \in S} \Psi^{\mathbf{\Lambda}}(n) \cdot \tau_n(\mathcal{M})(s) \cdot \mathbf{r}(s).$$

Definition 3.5 (*Cumulative reward approximation*). Let $\mathcal{M}$, $\mathbf{S} = (S_0, S_1, \ldots)$, and $\mathbf{\Lambda} = (\Lambda_0, \Lambda_1, \ldots)$ be as in Theorem 2.5 and let $m \in \mathbb{N}$ and $\hat{\mathbf{S}}$ be as in Definition 2.6. Then

we define

$$\mathcal{C}_t(\mathcal{M}, \mathbf{r}, t, \hat{\mathbf{S}}, \mathbf{\Lambda}) \stackrel{\text{def}}{=} \sum_{n=0}^{m} \sum_{s \in S} \Psi^{\mathbf{\Lambda}}(n) \cdot \hat{\tau}_n(\mathcal{M})(s) \cdot \mathbf{r}(s).$$

Calculating the cumulative rewards is of similar complexity to calculating the instantaneous rewards. After each step n, we multiply the probability in the discrete-time process by the corresponding cumulative reward and the value from Ψ, and then sum up the values obtained this way. The time overhead to compute the accumulated reward values is negligible. More importantly, it is not necessary to extend the state space, and hence the space complexity compared to FAU is not increased.

The corollary can be seen as a generalisation of a previous result [Kwiatkowska et al. 2006, Theorem 1], where the computation of cumulative reward-based properties is also considered. However, the analysis in [Kwiatkowska et al. 2006] relies on complete exploration of the state space and uses the special case $\mathbf{\Lambda} = (\Lambda, \Lambda, \ldots)$, which reduces the birth process to a Poisson process.

The computation of the error and the bounds on the number of steps is more involved for cumulative rewards than for instantaneous rewards, as shown in Corollary 2.11. The precision which can be achieved depends on the structure of the CTMC and the state rewards. We often have models in which, for each state, the sum of the rates to new states (further away from initial states) is bounded. We remark that this does not restrict the rates back to previously visited states. For this class of models, which includes many realistic examples as shown below, we derive error bounds as follows.

COROLLARY 3.6 (ERROR BOUND CUMULATIVE). *Consider a CTMC $\mathcal{M} = (S, \pi, \mathcal{R})$ for which we have a fixed Λ so that for each $n \in \mathbb{N}$ and $s \in S_n$ (cf. Theorem 2.5) we have that $\sum_{s' \in S_{n+1}} \mathcal{R}(s, s') \leq \Lambda$. Further, consider a state reward structure $\mathbf{r}$ so that we have fixed constants $c, d \in \mathbb{R}_{\geq 0}$ where for all $n \in \mathbb{N}$ and $s \in S_n$ we have $\mathbf{r}(s) \leq c + dn$. Consider $m \in \mathbb{N}$, $\hat{\mathbf{S}}$, and $\hat{\tau}(\mathcal{M})$ as in Definition 2.6. Set $\mathbf{\Lambda}_b \stackrel{\text{def}}{=} (\Lambda, \Lambda, \ldots)$ and $B \stackrel{\text{def}}{=} t \cdot (c + d + d\Lambda t)$. If*

$$B - \sum_{n=0}^{m} (c + dn) \cdot \Psi^{\mathbf{\Lambda}_b}(n) < \frac{\epsilon}{2} \text{ and } B \cdot \left(1 - \sum_{s \in \hat{S}_m} \hat{\tau}_m(\mathcal{M})(s)\right) < \frac{\epsilon}{2}$$

then we have

$$\mathcal{C}_t(\mathcal{M}, \mathbf{r}) - \mathcal{C}_t(\mathcal{M}, \mathbf{r}, \hat{\mathbf{S}}, \mathbf{\Lambda}) < \epsilon.$$

PROOF. When applying the FAU method, the worst case of reward loss is when we have the birth process $\Phi^{\mathbf{\Lambda}_b} = (\mathbb{N}, \pi, \mathcal{R})$ with reward structure $\mathbf{r}$, so that, for all $n \in \mathbb{N}$, we have $\mathbf{r}(n) \stackrel{\text{def}}{=} c + dn$. Denote the total accumulated reward until time t for this model by B. Thus, we lose no more reward than $B - \sum_{n=0}^{m} (c + dn) \cdot \Psi^{\mathbf{\Lambda}_b}(n)$ in case we use $\hat{S}_n = S_n$ and perform m steps in the FAU.

To take into account the loss of rewards from using $\hat{S}_n \subseteq S_n$, we consider the total probability lost $(1 - \sum_{s \in \hat{S}_m} \hat{\tau}_m(\mathcal{M})(s))$. In the worst case, this probability was already lost at the beginning. In this case, we lose up to $B \cdot (1 - \sum_{s \in \hat{S}_m} \hat{\tau}_m(\mathcal{M})(s))$.

By adding up the two sources of error, we obtain the result. □

If the rates or rewards are increasing more quickly, e.g., if we have a quadratic increase in the rewards, that is, $\mathbf{r}(s) \leq c + dn^2$ for $s \in \hat{S}_n$, the bounds on the error can be obtained using similar reasoning for a different value of B. Because of the simple structure of birth processes, it is possible to quickly approximate $\sum_{n=0}^{m} (c + dn) \cdot \Psi^{\mathbf{\Lambda}}(n)$ to find the value m to terminate the approximation in the worst case.

Example 3.7 (*FAU for Cumulative Rewards*). We reconsider the CTMC from Example 2.2 for which we computed transient probabilities in Example 2.7. We are interested in the expected total number of changes to the number of molecules until time $t = 0.1$, and thus assign a reward of 1 to each state change. As discussed, we transform these transition rewards into a state reward structure $\mathbf{r}$. For instance, state $s = 100$ has two transitions with rates 10 and 11, both with reward of 1, so that the state reward here is $10 \cdot 1 + 11 \cdot 1 = 21$. We have $\Psi^{\mathbf{\Lambda}}(0) \approx 0.042$, $\Psi^{\mathbf{\Lambda}}(1) \approx 0.029$, $\Psi^{\mathbf{\Lambda}}(2) \approx 0.017$. To compute cumulative rewards, according to Corollary 3.4 we can proceed as follows: after each step n of Example 2.7 and Fig. 3, for each $s \in S_n$ ($s \in \hat{S}_n$) we compute the product $\Psi^{\mathbf{\Lambda}}(n) \cdot \hat{\tau}_n(\mathcal{M})(s) \cdot \mathbf{r}(s)$ and build the sum $v(n) = \sum_{s \in S} \Psi^{\mathbf{\Lambda}}(n) \cdot \hat{\tau}_n(\mathcal{M})(s) \cdot \mathbf{r}(s)$ of these values. States $s \in S \setminus S_n$ need not be considered, because for those $\tau_n(\mathcal{M})(s) = 0$. This value $v(n)$ is then added to the partially computed total cumulative reward. In the example, we have $v(0) \approx 0.042 \cdot 1 \cdot 21$, $v(1) \approx 0.029 \cdot (0.524 \cdot 20.79 + 0.476 \cdot 21.21)$, $v(2) = 0.017 \cdot (0.269 \cdot 20.58 + 0.010 \cdot 20.79 + 0.493 \cdot 21 + 0.227 \cdot 21.42)$. Finally, we obtain $\mathcal{C}_{0.1}(\mathcal{M}, \mathbf{r}) \approx 2.099$.

4. INTERVAL SPLITTING

So far we considered FAU for a single time horizon. However, it is often advantageous to consider several smaller time intervals instead, analysing each of them and combining the results. This procedure can improve performance and enhance the ability of FAU to deal with stiff models. Interval splitting is known to benefit uniformisation-based methods, as described in [van Moorsel and Wolter 1998], and our experimental results in Section 5 confirm this for two out of three case studies. We argue that interval splitting should be considered an integral part of the FAU method.

LEMMA 4.1 (INTERVAL SPLITTING). *Consider*

— *a time duration* $t > 0$,
— *time durations* $t_i > 0$ *for* $i \in \{1, \ldots, n\}$, $n \in \mathbb{N}$, *with* $t = \sum_i t_i$,
— *a CTPM* $\mathcal{M} = (S, \pi, \mathcal{R})$,
— *CTPMs* $\mathcal{M}_i = (S, \pi^i, \mathcal{R})$ *such that* $\pi^1 = \pi$, *and* $\pi^{i+1} = \pi^i_{t_i}(\mathcal{M}_i)$ *(cf. Definition 2.4),*
— *a state reward structure* $\mathbf{r}\colon S \to \mathbb{R}_{\geq 0}$.

Then we have that

— $\pi_t(\mathcal{M}) = \pi_{t_n}(\mathcal{M}_n)$,
— $\mathcal{I}_t(\mathcal{M}, \mathbf{r}) = \mathcal{I}_{t_n}(\mathcal{M}_n, \mathbf{r})$,
— $\mathcal{C}_t(\mathcal{M}, \mathbf{r}) = \sum_i \mathcal{C}_{t_i}(\mathcal{M}_i, \mathbf{r})$.

The lemma follows from Definition 2.4, Definition 2.9, and Definition 3.3 by considering $\pi_t(\cdot)$ as the solution to a differential equation. The transient probabilities and reward values are found by computing them first for the time bound t_1. Then we proceed to calculate them for t_2, but use the transient probabilities at time t_1 as the initial distribution. This procedure is repeated until t_n is reached. For cumulative rewards, we also have to compute the reward accumulated during each of the intervals.

The transient probabilities obtained this way are the same as the ones we would have obtained if we had performed a single analysis up to t. For the instantaneous rewards, the rewards are weighted by the transient probabilities as in Definition 2.9, while the cumulative rewards are obtained as the sum $\sum_i \mathcal{C}_{t_i}(\mathcal{M}_i, \mathbf{r})$. To use FAU in combination with interval splitting, we have to take into account the error introduced by FAU in each t_i.

COROLLARY 4.2. *Consider*

— *a time duration* $t > 0$,

— *time durations* $t_i > 0$ *for* $i \in \{1, \ldots, n\}$, $n \in \mathbb{N}$, *with* $t = \sum_i t_i$,
— *a CTPM* $\mathcal{M} = (S, \pi, \mathcal{R})$,
— *CTPMs* $\mathcal{M}_i = (S, \pi^i, \mathcal{R})$ *such that* $\pi^1 = \pi$, *and* $\pi^{i+1} = \hat{\pi}^i_{t_i}(\mathcal{M}_i, \hat{\mathbf{S}}^i, \mathbf{\Lambda}^i)$, *where* $\hat{\pi}^i, \hat{\mathbf{S}}^i, \mathbf{\Lambda}^i$ *are as* $\hat{\pi}, \hat{\mathbf{S}}, \mathbf{\Lambda}$ *in Definition 2.6,*
— *a state reward structure* $\mathbf{r}\colon S \to \mathbb{R}_{\geq 0}$.

Assume that we have error bounds of $\frac{\epsilon}{2n}$ *according to Corollary 2.11 and Corollary 3.6 for each* t_i, $i \in \{1, \ldots, n\}$, *and further that*

$$\max_{s \in S} \mathbf{r}(s) \cdot \left(\sum_{s \in S} \pi_i(s) - \sum_{s \in S} \pi_{i+1}(s) \right) < \frac{\epsilon}{2n}.$$

Then we have

— $\mathcal{I}_t(\mathcal{M}, \mathbf{r}) - \mathcal{I}_{t_n}(\mathcal{M}_n, \mathbf{r}, \hat{\mathbf{S}}^n, \mathbf{\Lambda}^n) < \epsilon$,
— $\mathcal{C}_t(\mathcal{M}, \mathbf{r}) - \sum_i \mathcal{C}_{t_i}(\mathcal{M}_i, \mathbf{r}, \hat{\mathbf{S}}^i, \mathbf{\Lambda}^i) < \epsilon$.

PROOF. From the above we find $\max_{s \in S} \mathbf{r}(s) \cdot \left(1 - \sum_{s \in S} \pi_n(s)\right) < \frac{\epsilon}{2}$ and conclude the correctness of instantaneous rewards. For the cumulative rewards, we have an error of no more than $\frac{\epsilon}{2}$ from the loss of cumulative rewards during the individual analyses for each t_i. Note that the values of initial distributions π_i do not necessarily add up to 1. By the above equation, we can bound these errors to $\frac{\epsilon}{2}$. Thus, the total error stays below ϵ. □

Example 4.3 (*Interval splitting*). We revisit the birth-death process of Example 2.2 and compute the average number of molecules at time 0.1, given initial distribution $\pi(100) = \pi^1(100) = 1$. First, we compute the transient probability distribution at time 0.05, for which we find $\pi^1_{0.05}(\mathcal{M}_1)(99) \approx 0.221, \pi^1_{0.05}(\mathcal{M}_1)(100) \approx 0.453, \pi^1_{0.05}(\mathcal{M}_1)(101) \approx 0.199$. Then, we use $\pi^2 = \pi^1_{0.05}(\mathcal{M})$ as the initial distribution of the second analysis. This results in the transient probability values $\pi_{0.1}(\mathcal{M})(99) = \pi^2_{0.05}(\mathcal{M}_2)(99) \approx 0.225, \pi_{0.1}(\mathcal{M})(100) = \pi^2_{0.05}(\mathcal{M}_2)(100) \approx 0.299, \pi_{0.1}(\mathcal{M})(101) = \pi^2_{0.05}(\mathcal{M}_2)(101) \approx 0.202$, equal to those in Example 2.12. The same instantaneous reward $\mathcal{I}_{0.1}(\mathcal{M}, \mathbf{r}) \approx 99.900$ is found as well.

We now consider the cumulative reward structure of Example 3.7 and again split the time 0.1 into two intervals of length 0.05 each. The accumulated reward is computed for both intervals. In the first part a reward of $\mathcal{C}_{0.05}(\mathcal{M}_1, \mathbf{r}) \approx 1.050$ is accumulated, and in the second part we accumulate $\mathcal{C}_{0.05}(\mathcal{M}_2, \mathbf{r}) \approx 1.049$. The total reward is given by $\mathcal{C}_{0.1}(\mathcal{M}, \mathbf{r}) = \mathcal{C}_{0.05}(\mathcal{M}_1, \mathbf{r}) + \mathcal{C}_{0.05}(\mathcal{M}_2, \mathbf{r}) \approx 2.099$, which matches Example 3.7.

As per [van Moorsel and Wolter 1998], the number of matrix-vector multiplications in the i-th interval, N_i, depends on the number of jumps in the birth process and satisfies $1 - \sum_{n=0}^{N_i} \pi_t(\Phi^{\mathbf{\Lambda}})(n) < \frac{\varepsilon}{2}$. The total number of required matrix-vector multiplications can therefore be minimised with respect to the splitting. For the FAU method specifically, the cost of the matrix-vector multiplication changes per iteration, and depends on the number of significant states. The cost of solving the birth process $\Phi^{\mathbf{\Lambda}}$ has to be considered as well. The birth process itself is solved using standard uniformisation with a uniformisation rate $\lambda \geq \sup\{\Lambda_0, \Lambda_1, \ldots, \Lambda_n\}$. We note that the uniformisation rate for the birth process can be significantly larger than the adaptive uniformisation rate, which occurs when states with large exit rates become insignificant. Depending on the time bound, this can result in significant computational costs. By splitting the interval, we make sure that standard uniformisation is executed with a uniformisation rate that is not unnecessarily high with respect to the relevant states.

5. CASE STUDIES AND IMPLEMENTATION

We have integrated the fast adaptive uniformisation method in the probabilistic model checker PRISM [Kwiatkowska et al. 2011]. It is available along with the current development release of PRISM. Our implementation builds on top of the "explicit" engine, and is written in Java. Models can be input in the native language of PRISM. Alternatively, they can be given in Systems Biology Markup Language (SBML) and then transformed into the input language of PRISM. Properties are specified as non-nested continuous stochastic logic (CSL) [Aziz et al. 2000] formulae extended with the reward operator [Kwiatkowska et al. 2007], as either time-bounded until, or instantaneous or cumulative reward properties.

To show the practical applicability of our method, we apply it to three case studies and compare to standard uniformisation in PRISM. We terminate the state space exploration once we obtain $(1 - \sum_{n=0}^{m} \pi_t(\Phi^{\mathbf{\Lambda}}(n))) < \varepsilon$ for an adequate ε, and discard states with probability of less than δ in the discrete-time process. Experiments were performed on a Linux computer with an Intel i7-3770 processor with 3.40GHz and 32GB of RAM. For one of the case studies, we compare the performance of our tool to the FAU method implemented in the tool MARCIE [Schwarick et al. 2011].

Wherever possible, we have compared our results to the PRISM engine which performs best for that particular model. This includes comparison with the symbolic engines of PRISM ("mtbdd" and "hybrid"), which tended to perform worse than the "explicit" engine on our examples, likely due to loss of regularity. We note that symbolic engines cannot handle infinite-state models employed here, but the "explicit" engine is able to, provided that the reachable state space is finite. Conventional methods could perform better than FAU when the state space is sufficiently small, in view of the additional overhead necessary for FAU. We anticipate that the performance of PRISM is indicative of modern probabilistic model checkers, and therefore our conclusions are more generally applicable.

In this paper, we do not compare against simulation-based approaches, such as approximate probabilistic model checking available in PRISM (probability estimation and statistical hypothesis testing); while simulation has the advantage of not requiring the generator matrix to be constructed, and hence does not suffer from state space explosion, it is sensitive to the size of time bounds and can only guarantee error bounds with a given confidence interval. FAU can provide guarantees for an arbitrary precision by controlling δ, although reducing δ will generally incur higher memory requirements. Investigating the trade-off between FAU and simulation-based techniques deserves further study.

5.1. PRISM Language

The input language of PRISM is a simple guarded-command language based on the Reactive Modules formalism of Alur and Henzinger [Alur and Henzinger 1999]. PRISM provides native support for discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), Markov decision processes (MDPs) and probabilistic timed automata (PTAs). As the contribution of this paper is based on CTMCs, we only describe the mechanisms to support this model type. A PRISM model consists of one or more *modules*. Each such module contains local *variables* and *commands*. In addition, global variables can be defined. A command contains a *guard*, which is an expression over the variables stating under which conditions it can be executed. Further, it contains a number of *updates*, each of which consists of a *rate* describing the speed with which it will be executed, as well as an *assignment* setting the model variables to new values. Optionally, a command can have a *label*. It also

contains a description of one or more *initial states* in terms of a boolean expression over the variables.

A state in the CTMC semantics of a model is then an assignment of valid values to all the variables. The potential initial distributions of the CTMC semantics are Dirac distributions assigning probability 1 to one of the initial states. The transition matrix, $\mathcal{R}$, is induced from the commands. A command is *active* in a given state s if the variable assignment of the state fulfills the guard of the command. If s' is obtained from s by the assignment of an update with rate λ, we add λ to the transition rate $\mathcal{R}(s, s')$.

In addition, a PRISM model can also contain *reward structures*, which are used to express instantaneous and cumulative rewards. Each reward structure contains one or more *reward items*, which are either state or transition reward items. A state reward item has a *guard* stating under which conditions its value is incurred, and a *reward expression* over the model variables stating the reward obtained per time unit if the guard is fulfilled. A transition reward item also has a guard and a reward expression. In addition, it can also contain a label, such that the reward specified by the label is obtained instantaneously on a transition from one state to the other, under the condition that the guard is fulfilled. If the state reward item does not contain a label, it is applied to transitions resulting from unlabelled commands. If it is labelled, it is applied to commands with the corresponding label. If the guards of several reward items are fulfilled, their values are added up. In particular, if the guard of no reward item is fulfilled, a reward of 0 results. A reward structure can also have a *name* by which it can be referred to in a property specification.

The properties supported for CTMCs are given in a variant of CSL. In particular, in addition to probability quantifiers `P` there are now reward quantifiers `R`. Instantaneous rewards can be expressed as `R<reward-structure><probability-bound>[I=<time-point>]`. Here, `<reward-structure>` specifies the reward structure to use, either as the number of the reward structure in terms of the order they are specified, or as its name (if `<reward-structure>` is left out, the first reward structure will be used). The `<time-point>` corresponds to the value of t in Definition 2.9. The probability bound can be of the form `<r`, `>r`, `<=r`, `>=r` for some real number `r`, used to check whether the reward computed is bounded by `r`. Alternatively, it can be equal to `=?`, used to obtain the reward computed as a real number. Cumulative rewards can be expressed as `R<reward-structure><probability-bound>[C<=<time-bound>]`, where `<time-bound>` corresponds to t of Definition 3.5.

Example 5.1 (*CTMCs in* PRISM). Reconsider the birth-death process of Example 2.2. In Fig. 4, we give a description in the PRISM language. The line with `ctmc` denotes the model type. The model has a single module `birthdeath`. This module contains a single variable `molecules`. States of the CTMC semantics thus consist of an assignment of values to this variable. The type of the variable is `int`, which means that the variable must be assigned an integer number. Using `init 100` we specify that in the initial state the value of `molecules` is `100`. Note that there is no upper bound on the number of states. The module contains two commands, labelled with `birth` and `death` respectively. The guard of each of them is `molecules>0`, which means that both can only be executed if the number of molecules is positive. The first command has a single update with a rate of `0.10*molecules` increasing the number of molecules by one, whereas the single update of the second command decreases the number of molecules with a rate of `0.11*molecules`. Ignoring the labels, the two commands could also be combined into an equivalent single command with two updates:

```
[] molecules>0 -> 0.10*molecules : (molecules'=molecules+1)
                + 0.11*molecules : (molecules'=molecules-1);
```

```
ctmc

module birthdeath
  molecules : int init 100;
  [birth] molecules>0 -> 0.10*molecules : (molecules'=molecules+1);
  [death] molecules>0 -> 0.11*molecules : (molecules'=molecules-1);
endmodule
```

Fig. 4: Birth-death process in PRISM.

```
rewards "molecules"   rewards "death"       rewards "timeeven"
  true : n;             [death] true : 1;     mod(n,molecules)=0 : 1;
rewards               endrewards            endrewards

rewards "birth"       rewards "reactions"   rewards "smaller"
  [birth] true : 1;     [birth] true : 1;     molecules<=100 : molecules;
endrewards              [death] true : 1;   endrewards
                      endrewards
```

Fig. 5: Reward structures for the birth-death process of Fig. 4.

In Fig. 5 we provide some possible reward structures for this example. Reward structure `molecules` expresses the number of molecules in a given state: the guard `true` is always fulfilled and the expression `molecules` expresses exactly this value. Thus, using `R{``molecules''}=?[I=0.1]`, we can express the expected number of molecules present at time $0.1$, that is, the instantaneous reward value of Example 2.12. Next, `birth` expresses the number of births, `death` the number of deaths, and `reactions` the number of all types of reactions using transition reward items. Thus, by `R{``reactions''}=?[C<=0.1]` we can express the total number of reactions which have happened until time $0.1$, that is, the cumulative reward value of Example 3.7. Reward structure `timeeven` assigns a reward of `1` to states with an even number of molecules and `0` to those with an odd number. Thus, using `R{``timeeven''}=?[I=0.1]` we can ask for the probability that at time `0.1` the model is in a state with an even number of molecules. With `R{``timeeven''}=?[C<=0.1]` we can ask for the time spent in a state with an even number of molecules accumulated until time `0.1`. The reward structure `smaller` is a state reward stating the number of molecules in a given state, but ignoring states in which this number is larger than `100`.

5.2. Using Fast Adaptive Uniformisation in PRISM

The algorithm described in this paper has been integrated in the `explicit` engine of PRISM. Thus, to use it, this engine has to be activated. Further, the transient probability computation has to be set to use the fast adaptive uniformisation method. These values can be changed in the PRISM settings in the GUI (menu "Options" $\rightarrow$ "Options"), or via the command line if the GUI is not used. In Fig. 6 we give an overview of the relevant options. "Engine (-transientmethod fau)" and "transient probability (-explicit)" are as described above. The meaning of the other options is as follows:

— "FAU epsilon (-fauepsilon)": corresponds to ε in this paper,
— "FAU delta (-faudelta)": corresponds to δ in this paper,

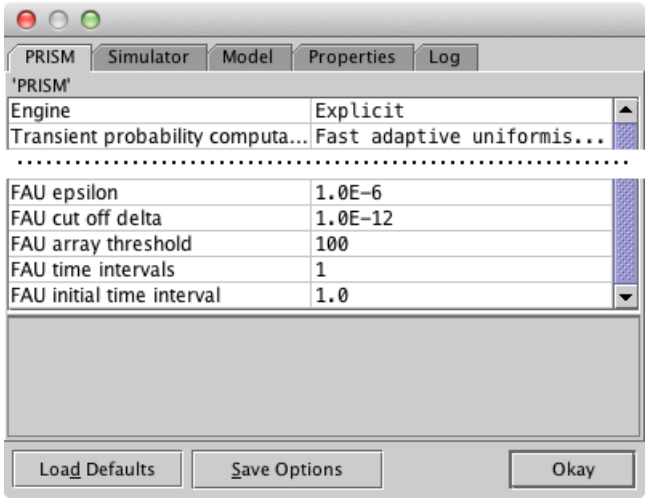

Fig. 6: PRISM options dialogue box.

— "FAU array threshold (-fauarraythreshold)": if after this number of steps in the discrete-time Markov chain the state space does not change any more, the algorithm will switch to a different data structure. This data structure cannot be easily extended with new states, but allows for faster iteration steps. If it turns out that new states have to be added, the algorithm has to switch back to the usual data structure, which is costly.
— "FAU time intervals (-fauintervals)": in some cases, analyses are faster when dividing the time interval in which the analysis is performed into several parts. For instance, a time interval 10 could be divided into 5 time intervals of length 2.
— "FAU initial time interval (-fauinitival)": with this option an additional short time interval can be specified in addition to the other time intervals from the option described above. Thus, for a time interval 10, initial time interval 1 and number of time intervals 3, the analysis will be divided into four parts: the first will be over a time interval of 1, followed by three parts of length 3. If the time interval to analyse is shorter than the value of this option, the initial time interval will not be used.

5.3. Discrete Stochastic Model Test Suite

The Discrete Stochastic Model Test Suite [Evans et al. 2008] is a test suite of models encoded in the Systems Biology Markup Language (SBML), for which values of certain properties have been computed up to a given precision. It is aimed at stochastic simulator developers who can evaluate the accuracy of their tools against known results.

We used PRISM's SBML import functionality[2] to convert SBML to PRISM files. The models have infinitely many states, and so cannot be handled by existing PRISM engines (except "explicit", providing the reachable state space is finite). As the SBML import does not yet support the SBML feature of *events*, we were only able to analyse 35 out of the 39 test models. For this case study, we apply analyses for a time bound of 50, which is the largest one for which results are included in the SBML models. We chose parameters as follows:

[2]http://www.prismmodelchecker.org/manual/RunningPRISM/SupportForSBML

Model	Time (s)	States	Lost	Molecules	Reactions
001-01	2	321	9.6707E-09	60.6531	826.2856
001-03	2	347	1.0541E-08	0.6738	2,085.8503
001-04	2	163	8.6868E-09	6.0653	82.6286
001-05	21	2,999	1.1885E-08	6,065.3065	82,628.5610
001-06	2	321	9.6707E-09	60.6531	826.2856
001-07	44	161,486	2.1713E-08	60.6531	826.2856
001-08	2	321	9.6707E-09	60.6531	826.2856
001-18	2	277	8.8953E-09	77.8801	464.5184
001-19	2	321	9.6707E-09	60.6531	826.2856
002-01	2	44	6.6105E-09	9.9326	90.0674
002-02	2	151	9.6617E-09	99.3262	900.6738
002-03	2	107	7.8426E-09	49.6631	450.3369
002-04	18	1,376	1.1896E-08	9,932.6204	90,067.3791
002-05	2	151	9.6617E-09	99.3262	900.6738
002-06	4	36,177	1.1379E-08	99.3262	900.6738
002-07	2	151	9.6617E-09	99.3262	900.6738
002-08	2	44	6.6105E-09	9.9326	90.0674
003-01	2	49	6.1194E-09	28.5423	64.7560
003-02	2	158	8.4320E-09	144.9960	573.9888
003-05	2	49	6.1194E-09	35.7289	64.7560
004-01	2	124	8.8187E-09	24.9989	275.0011
004-02	2	173	1.0418E-08	25.0000	525.0000
004-03	4	773	3.6533E-08	25.0000	5,024.9999
ext. 001-01	181	297,825	4.8164E-08	60.6531	826.2856

Table I: Discrete Stochastic Model Test Suite Results.

— "FAU epsilon (-fauepsilon)":1e-9
— "FAU delta (-faudelta)": 1e-14
— "FAU array threshold (-fauarraythreshold)": 100 (default)
— "FAU time intervals (-fauintervals)": 10
— "FAU initial time interval (-fauinitival)": 1.0 (default)

The results for a selection of the models are given in Table I. For each "Model", we give the "Time (s)" in seconds needed to perform the analysis, the maximal number of "States" in memory, and the probability "Lost" through approximation. The column "Molecules" is an instantaneous reward property, `R=?[I=50]`, which returns the expected number of molecules of the first species of the model under consideration. The column "Reactions" is the expected number of reactions until time 50, which is a cumulative reward property, `R=?[C<=50]`. In the table, each row corresponds to two analyses; however, the computation time is the same for both since the same number of states had to be explored.

All analyses (with the exception of "ext. 001-01" not originally from the test suite, see below) took less than a minute. The results we obtain for "Molecules" agree with those provided by the test suite, for the number of decimal places given there (values for "Reactions" are not provided by the test suite). For the model "001-01", we attempted a naive approach to compute the number of reactions by adding a new species "Reactions", increasing the dimensionality. As can be seen from results given in the last row ("ext. 001-01") of Table I, these performance figures were much worse than for our implementation. We remark that these figures are similar to those for the (unmodified) "001-07", in which also a species tracking a specific reaction is introduced.

N	T	Fastest PRISM engine - explicit				FAU				
		Time (s)	States	Finished	Reactions	Time (s)	States	Lost	Finished	Reactions
1	10000	3	169	0.0593	15.3193	**2**	169	1.6882E-09	0.0593	15.3193
1	50000	**1**	169	0.9999	26.9997	32	169	1.7133E-09	0.9999	26.9997
1	100000	**2**	169	1.0000	27.0000	14	169	1.7892E-09	1.0000	27.0000
2	10000	**1**	5,748	0.0224	37.1224	6	**5,299**	1.3024E-08	0.0224	37.1224
2	50000	**2**	5,748	0.9999	51.2958	39	**5,299**	1.5028E-08	0.9999	51.2958
2	100000	**2**	5,748	1.0000	51.2963	151	**5,299**	1.5090E-08	1.0000	51.2963
3	10000	**15**	93,538	0.0138	59.7229	123	**67,292**	1.0059E-07	0.0138	59.7229
3	50000	**52**	93,538	0.9999	75.0530	151	**67,292**	1.0994E-07	0.9999	75.0530
3	100000	**96**	93,538	1.0000	75.0536	326	**67,292**	1.1002E-07	1.0000	75.0536
4	10000	**268**	970,539	0.0103	82.6250	737	**514,414**	5.6703E-07	0.0103	82.6250
4	50000	1,039	970,539	0.9998	98.6211	**773**	**514,414**	5.8001E-07	0.9998	98.6211
4	100000	1,976	970,539	1.0000	98.6218	**1019**	**514,414**	5.8009E-07	1.0000	98.6218
5	10000	3,463	7,377,039	0.0085	105.6602	**2111**	**2,814,235**	2.9759E-06	0.0085	105.6602
5	50000	-	-	-	-	**2209**	**2,814,235**	2.9907E-06	0.9998	122.0891
5	100000	-	-	-	-	**2614**	**2,814,235**	2.9907E-06	1.0000	122.0897
6	10000	-	-	-	-	**5073**	**12,163,811**	1.3377E-05	0.0074	128.7586
6	50000	-	-	-	-	**5268**	**12,163,811**	1.3393E-05	0.9998	145.4913
6	100000	-	-	-	-	**5614**	**12,163,811**	1.3393E-05	1.0000	145.4920

Table II: DNA Strand Displacement Results.

5.4. DNA Strand Displacement

DNA strand displacement (DSD) [Seelig et al. 2006] is a mechanism for performing computation with DNA molecules. A variety of logic circuits can be designed and implemented using DSD. Initial species of DNA are mixed together in a reaction tube, and then strand displacement reactions proceed autonomously, relying solely on hybridisation between complementary nucleotide sequences to perform computational steps. In this case study, we consider transducer gates modelled and analysed in [Lakin et al. 2012, Section 2] (example `transducer_K=3.sm`). This model features the parameter N, which corresponds to the number of copies for initial species, and K, the number of transducers placed in series.

We are interested in the probability that the computation is finished by time T, given by `P=?[F[0,50] Finished]`. The expected total number of reactions ("Reactions") is given by a cumulative reward property, `R=?[C<=50]`. For the analysis of this model, we chose the following parameters:

— "FAU epsilon (-fauepsilon)": 1e-9
— "FAU delta (-faudelta)": 1e-13
— "FAU array threshold (-fauarraythreshold)": 100 (default)
— "FAU time intervals (-fauintervals)": 1 (default)
— "FAU initial time interval (-fauinitival)": 1.0 (default)

We fix $K = 3$ and provide results for different N and T in Table II. The state space of this case study is small enough to be compared against existing methods in PRISM. We included the results for the "explicit" engine because it was the fastest. In each row, the best performance in terms of state space size or time is highlighted in boldface.

Note also that the FAU method is able to handle larger models than existing PRISM engines, and obtains better performance for larger model instances.

5.5. DNA Walkers

We consider models of *DNA walkers* [Wickham et al. 2011], which can be used to design logic circuits on the nanoscale. A significant difference from DSD designs is that a DNA walker operates on a track of DNA strands (called anchorages) tethered to a surface, rather than in solution, and thus the model has to incorporate spatial information. An example of an XOR circuit is shown in Fig. 7. The walker starts in the Initial position and can navigate down a series of junctions [Wickham et al. 2012]. An enzyme cuts the anchorage when the walker is attached, allowing the walker to step onto the next anchorage. When the walker steps onto an absorbing anchorage, here labelled with True and False, the computation ends. The prior input unblocks certain anchorages, which in turn directs the walker at each junction. Occasionally, the blockade mechanism fails to block an anchorage, which can cause the walker to output the wrong answer. In addition, the walker sometimes steps over blockades or between tracks, which is another source of error.

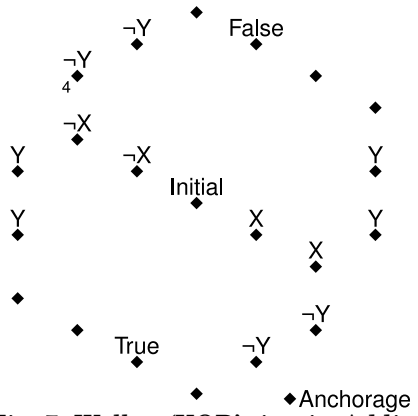

Fig. 7: Walker 'XOR' circuit. Adding the input X will unblock the anchorages labelled X.

A CTMC model of the walker was developed [Dannenberg et al. 2013; Dannenberg et al. 2014] previously[3], and in this paper we apply model checking with FAU. We use the following parameter set:

— "FAU epsilon (-fauepsilon)": 1e-6
— "FAU delta (-faudelta)": 1e-8
— "FAU array threshold (-fauarraythreshold)": 100 (default)
— "FAU time intervals (-fauintervals)": 1 (default)
— "FAU initial time interval (-fauinitival)": 1.0 (default)

We analyse three variants of the XOR-circuit and summarise the results in Table III. The unmodified track, shown in Fig. 7, is "xor", and the suffix "-S" indicates that only one blocker is used instead of two consecutive ones, whereas suffix "-large" indicates a track with more anchorages. (X, Y) or $(X, \neg Y)$ indicates the input to the computation, which opens up the blocked anchorages that match the labels of the input. Because the track has a point-symmetry, the results for inputs X, Y and $\neg X, Y$ are the same, as well as for inputs $\neg X, \neg Y$ and $X, \neg Y$. The unmodified track has 2.9×10^7 states; it can be constructed, but not analysed, with PRISM's standard engines. The larger circuit has 1.9×10^9 states. All three variants are too large to model check with standard uniformisation.

We model check the expected number of steps (column "Steps", `R=?[C<=T]`) and the probability of walkers reaching the desired anchorage (column "Signal", `P=?[F[T,T] Finished]`) by time $T = 200$ min. The expected number of steps correlates well with the track layout: when fewer anchorages are blocked ("-S"), the walker takes more steps on average. A larger track also results in more steps taken on average. Column "Blocked", a cumulative reward property, shows how much time the walker spends on anchorages that were supposed to be blocked, and is in line with expectations.

[3]also see www.prismmodelchecker.org/casestudies/dna_walkers.php

Model	Time (s)	States	Lost	Signal	Steps	Blocked (s)
PRISM	*fau-delta 1e-8*					
xor(X, Y)	20	228,803	1.9736E-02	0.6455	7.7696	606.2731
xor$(X, \neg Y)$	22	239,680	2.2587E-02	0.5979	7.5610	659.3715
xor-S-(X, Y)	14	215,544	1.6719E-02	0.5374	8.8363	133.1672
xor-S-$(X, \neg Y)$	15	233,063	1.8775E-02	0.5473	8.4049	146.7377
xor-large-(X, Y)	43	443,584	5.1855E-02	0.5661	9.5020	577.2680
xor-large-$(X, \neg Y)$	45	455,685	5.2995E-02	0.5674	9.4983	567.3420
MARCIE	*appr-delta 1e-9*					
xor(X, Y)	1614	216,661	1.541E-2	0.6473	-	-
xor-S-(X, Y)	795	214,379	1.377E-2	0.5386	-	-
xor-large-(X, Y)	2641	507,756	3.572E-2	0.5712	-	-
PRISM	*fau-delta 1e-14, fau-epsilon 1e-9*					
xor(X, Y)	366	2,660,829	1.1838E-07	0.6527	7.8371	627.9572
xor-large-(X, Y)	6923	62,648,566	6.0992E-06	0 .5816	9.7201	623.4739

Table III: DNA Walkers.

When we decrease the parameter fau-delta, the number of explored states goes up and the amount of probability lost goes down. For fau-delta equal to 1e-14, at most 9 percent of the total state space is concurrently loaded in memory, but only 1.2e-7 of the probability is lost.

5.6. Comparison to Other Implementations

The fast adaptive unformisation method, without support for cumulative rewards, is implemented in the tools SABRE [Didier et al. 2010] and MARCIE [Schwarick et al. 2011]. The latter supports model checking of CSL-style properties for stochastic Petri nets using the FAU method. MARCIE is developed at the Brandenburg University of Technology in Cottbus and features macro support for Petri nets that describe biochemical mass-action reactions.

We have converted the DNA walker models into MARCIE scripts to allow a direct comparison between the two tools. For the "xor" circuit, we find that the symbolic engine in MARCIE reports the same number of reachable states as reported by PRISM. The DNA walker models can be analysed with the FAU method in MARCIE using an updated version of the tool that was provided by the authors. The behaviour of MARCIE is shown in Table III, where the columns "steps" and "blocked" are empty because MARCIE does not support cumulative rewards for the FAU method.

For MARCIE we have used the following settings:

— "Approximative numerical delta (–appr-delta)": 1e-9
— "Assumed maximal exit rate (–appr-lambda)": 1000

To make the number of states explored by MARCIE comparable, we have used, somewhat surprisingly, a cut-off delta that is one order of magnitude smaller than that used in PRISM. Using the same cut-off delta results in significantly fewer states explored, and a higher amount of probability lost.

Although the number of states explored is comparable, the performance of the two tools is different: PRISM is faster while MARCIE retains more probability during the analysis. The difference in speed is explained by the default interval splitting parameters in PRISM, as a short initial interval is particularly beneficial to this model. At the time of writing, the FAU method in MARCIE had no documented support for in-

terval splitting. The difference in retained probability cannot be accurately explained without further analysis.

6. CONCLUSION

In this paper, we have extended fast adaptive uniformisation so that it can also be applied to cumulative reward properties. Cumulative measures allow one to express many important quantitative properties, such as the expected number of times a certain reaction happens and the average percentage of time the system spends in a given state. Our method does not introduce a significant overhead to the analysis, and in particular does not require the explicit construction of the extended state space of the underlying continuous-time propagation model. In contrast to simulation-based approaches, we can compute guaranteed error bounds for properties, as opposed to ensuring a given confidence interval. We have applied it to several case studies, obtaining superior performance in virtually all cases compared to existing methods implement in PRISM and MARCIE, and demonstrated how interval splitting benefits the FAU method.

REFERENCES

R. Alur and T. Henzinger. 1999. Reactive Modules. *Formal Methods in System Design* 15, 1 (1999), 7–48.

A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. 2000. Model-checking continuous-time Markov chains. *ACM TCS* 1, 1 (2000), 162–170.

C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. 2010. Performance evaluation and model checking join forces. *Commun. ACM* 53, 9 (2010), 76–85.

C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. 2003. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE TSE* 29, 6 (2003), 524–541.

G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. 2001. The Möbius Modeling Tool. In *PNPM*. 241–250.

F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield. 2013. DNA walker circuits: computational potential, design, and verification. In *Proc. 19th International Conference on DNA Computing and Molecular Programming (DNA 19) (LNCS)*, D. Soloveichik and B. Yurke (Eds.), Vol. 8141. Springer, 31–45.

F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield. 2014. DNA walker circuits: computational potential, design, and verification. *Natural Computing* (2014). To appear.

F. Didier, T. A. Henzinger, M. Mateescu, and V. Wolf. 2010. SABRE: A Tool for Stochastic Analysis of Biochemical Reaction Networks. In *QEST*. 193–194.

T. W. Evans, C. S. Gillespie, and D. J. Wilkinson. 2008. The SBML discrete stochastic models test suite. *Bioinformatics* 24, 2 (2008), 285–286.

B. L. Fox and P. W. Glynn. 1988. Computing Poisson Probabilities. *Comm. ACM* 31, 4 (1988), 440–445.

E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. 2009. INFAMY: An Infinite-State Markov Model Checker. In *CAV*. 641–647.

J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. 2008. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science* 319, 3 (2008), 239–257.

A. Jensen. 1953. Markoff chains as an aid in the study of Markoff processes. *Skand. Aktuarietidskr.* 36 (1953), 87–91.

J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. 2011. The ins and outs of the probabilistic model checker MRMC. *PEVA* 68, 2 (2011), 90–104.

M. Kwiatkowska, G. Norman, and A. Pacheco. 2006. Model checking expected time and expected reward formulae with random time bounds. *CMA* 51 (2006), 305–316. Issue 2.

M. Kwiatkowska, G. Norman, and D. Parker. 2007. Stochastic Model Checking. In *SFM'07 (LNCS (Tutorial Volume))*, Vol. 4486. Springer, 220–270.

M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*. 585–591.

M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. 2012. Design and Analysis of DNA Strand Displacement Devices using Probabilistic Model Checking. *Journal of the Royal Society Interface* 9, 72 (2012), 1470–1485.

M. Mateescu. 2011. *Propagation Models for Biochemical Reaction Networks*. Ph.D. Dissertation. EPFL.

M. Mateescu, V. Wolf, F. Didier, and T. A. Henzinger. 2010. Fast adaptive uniformisation of the chemical master equation. *IET Syst Biol* 4, 6 (2010), 441–52.

B. Munsky and M. Khammash. 2006. The Finite State Projection Algorithm for the Solution of the Chemical Master Equation. *Journal of Chemical Physics* 124, 044104 (2006).

B. Munsky and M. Khammash. 2007. A multiple time interval finite state projection algorithm for the solution to the chemical master equation. *J. Comput. Physics* 226, 1 (2007), 818–835.

B. Munsky and M. Khammash. 2008. Computation of Switch Time Distributions in Stochastic Gene Regulatory Networks. In *American Control Conference*. 2761–2766.

M. Schwarick and M. Heiner. 2009. CSL model checking of biochemical networks with Interval Decision Diagrams. In *Proc. 7th International Conference on Computational Methods in Systems Biology (CMSB 2009) (LNCS/LNBI)*, Vol. 5688. Springer, 296–312.

M. Schwarick, M. Heiner, and C. Rohr. 2011. MARCIE - Model Checking and Reachability Analysis Done EffiCIEntly. In *QEST*. 91–100.

G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree. 2006. Enzyme-Free Nucleic Acid Logic Circuits. *Science* 314, 5805 (2006), 1585–1588.

D. Spieler, E. M. Hahn, and L. Zhang. 2014. Model Checking CSL for Markov Population Models. In *QAPL*. 93–107.

J. J. Tapia, J. R. Faeder, and B. Munsky. 2012. Adaptive coarse-graining for transient and quasi-equilibrium analyses of stochastic gene regulation. In *CDC*. 5361–5366.

A. P. A. van Moorsel and W. H. Sanders. 1994. Adaptive Uniformization. *ORSA Communications in Statistics: Stochastic Models* 10, 3 (1994), 619–648.

A. P. A. van Moorsel and K. Wolter. 1998. Numerical Solution of Non-Homogeneous Markov Processes through Uniformization. In *12th European Simulation MultiConference. Simulation - Past, Present and Future.*

S. F. J. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. J. Turberfield. 2012. A DNA-based molecular motor that can navigate a network of tracks. *Nature nanotechnology* 7, 3 (March 2012), 169–73.

S. F. J. Wickham, M. Endo, Y. Katsuda, K. Hidaka, J. Bath, H. Sugiyama, and A. J. Turberfield. 2011. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature nanotechnology* 6, 3 (2011), 166–169.