Stochastic Model Checking*

Marta Kwiatkowska, Gethin Norman, and David Parker

School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, United Kingdom

Abstract. This tutorial presents an overview of model checking for both discrete and continuous-time Markov chains (DTMCs and CTMCs). Model checking algorithms are given for verifying DTMCs and CTMCs against specifications written in probabilistic extensions of temporal logic, including quantitative properties with rewards. Example properties include the probability that a fault occurs and the expected number of faults in a given time period. We also describe the practical application of stochastic model checking with the probabilistic model checker PRISM by outlining the main features supported by PRISM and three real-world case studies: a probabilistic security protocol, dynamic power management and a biological pathway.

1 Introduction

Probability is an important component in the design and analysis of software and hardware systems. In distributed algorithms electronic coin tossing is used as a symmetry breaker and as a means to derive efficient algorithms, for example in randomised leader election [38,26], randomised consensus [3,18] and root contention in IEEE 1394 FireWire [37,47]. Traditionally, probability has also been used as a tool to analyse system performance, where typically queueing theory is applied to obtain steady-state probabilities in order to arrive at estimates of measures such as throughput and mean waiting time [30,61]. Probability is also used to model unreliable or unpredictable behaviour, as in e.g. fault-tolerant systems and multi-media protocols, where properties such as frame loss of 1 in every 100 can be described probabilistically.

In this tutorial, we summarise the theory and practice of stochastic model checking. There are a number of probabilistic models, of which we will consider two in detail. The first, discrete-time Markov chains (DTMCs), admit *probabilistic choice*, in the sense that one can specify the probability of making a transition from one state to another. Second, we consider continuous-time Markov chains (CTMCs), frequently used in performance analysis, which model *continuous real time* and probabilistic choice: one can specify the rate of making a transition from one state to another. Probabilistic choice, in this model, arises through *race conditions* when two or more transitions in a state are enabled.

* Partly supported by EPSRC grants EP/D07956X and EP/D076625 and Microsoft Research Cambridge contract MRL 2005-44.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 220–270, 2007.

Stochastic model checking is a method for calculating the likelihood of the occurrence of certain events during the execution of a system. Conventional model checkers input a description of a model, represented as a state transition system, and a specification, typically a formula in some temporal logic, and return 'yes' or 'no', indicating whether or not the model satisfies the specification. In common with conventional model checking, stochastic model checking involves reachability analysis of the underlying transition system, but, in addition, it must entail the calculation of the actual likelihoods through appropriate numerical or analytical methods.

The specification language is a *probabilistic* temporal logic, capable of expressing temporal relationships between events and likelihood of events and usually obtained from standard temporal logics by replacing the standard path quantifiers with a probabilistic quantifier. For example, we can express the probability of a fault occurring in a given time period during execution, rather than whether it is possible for such a fault to occur. As a specification language for DTMCs we use the temporal logic called Probabilistic Computation Tree Logic (PTCL) [29], which is based on well-known branching-time Computation Tree Logic (CTL) [20]. In the case of CTMCs, we employ the temporal logic Continuous Stochastic Logic (CSL) developed originally by Aziz et al. [4,5] and since extended by Baier et al. [10], also based on CTL.

Algorithms for stochastic model checking were originally introduced in [62,23,29,5,10], derive from conventional model checking, numerical linear algebra and standard techniques for Markov chains. We describe algorithms for PCTL and CSL and for extensions of these logics to specify reward-based properties, giving suitable examples. This is followed by a description of the PRISM model checker [36,53] which implements these algorithms and the outcome of three case studies that were performed with PRISM.

Outline. We first review a number of preliminary concepts in Section 2. Section 3 introduces DTMCs and PCTL model checking while Section 4 considers CTMCs and CSL model checking. Section 5 gives an overview of the probabilistic model checker PRISM and case studies that use stochastic model checking. Section 6 concludes the tutorial.

2 Preliminaries

In the following, we assume some familiarity with probability and measure theory, see for example [16].

Definition 1. *Let Ω be an arbitrary non-empty set and $\mathcal{F}$ a family of subsets of Ω. We say that $\mathcal{F}$ is a* field *on Ω if:*

1. *the empty set $\varnothing$ is in $\mathcal{F}$;*
2. *whenever A is an element of $\mathcal{F}$, then the complement $\Omega \setminus A$ is in $\mathcal{F}$;*
3. *whenever A and B are elements of $\mathcal{F}$, then $A \cup B$ is in $\mathcal{F}$.*

A field of subsets $\mathcal{F}$ is called a σ-algebra *if it is field which is closed under countable union: whenever $A_i \in \mathcal{F}$ for $i \in \mathbb{N}$, then $\cup_{i \in \mathbb{N}} A_i$ is also in $\mathcal{F}$.*

The elements of a σ-algebra are called *measurable sets*, and $(\Omega, \mathcal{F})$ is called a *measurable space*. A σ-algebra *generated* by a family of sets $\mathcal{A}$, denoted $\sigma(\mathcal{A})$, is the smallest σ-algebra that contains $\mathcal{A}$ which exists by the following proposition.

Proposition 1. *For any non-empty set Ω and $\mathcal{A}$ a family of subsets of Ω, there exists a unique smallest σ-algebra containing $\mathcal{A}$.*

Definition 2. *Let $(\Omega, \mathcal{F})$ be a measurable space. A function $\mu : \mathcal{F} \to [0, 1]$ is a* probability measure *on $(\Omega, \mathcal{F})$ and $(\Omega, \mathcal{F}, \mu)$ a* probability space*, if μ satisfies the following properties:*

1. $\mu(\Omega) = 1$
2. $\mu(\cup_i A_i) = \sum_i \mu(A_i)$ *for any countable disjoint sequence $A_1, A_2, \ldots$ of $\mathcal{F}$.*

The measure μ is also referred to as a *probability distribution*. The set Ω is called the *sample space*, and the elements of $\mathcal{F}$ *events*.

In order to go from a notion of size defined on a family of subsets $\mathcal{A}$ to an actual measure on the σ-algebra generated by $\mathcal{A}$, we need an extension theorem. The following [16] is a typical example.

Definition 3. *A family $\mathcal{F}$ of subsets of Ω is called a* semi-ring *if*

1. *the empty set $\varnothing$ is in $\mathcal{F}$;*
2. *whenever A and B are elements of $\mathcal{F}$, then $A \cap B$ is also in $\mathcal{F}$;*
3. *if $A \subseteq B$ are in $\mathcal{F}$, then there are* finitely *many* pairwise disjoint *subsets $C_1, \ldots, C_k \in \mathcal{F}$ such that $B \setminus A = \cup_{i=1}^{k} C_i$.*

This is not the form of the definition most commonly used in field because of the strange last condition but it is precisely the property that holds for 'hyper-rectangles' in $\mathbb{R}^n$ and, more importantly here, for the *cylinder sets* defined later in this tutorial.

Theorem 1. *If $\mathcal{F}$ is a semi-ring on X and $\mu : \mathcal{F} \to [0, \infty]$ satisfies*

1. $\mu(\varnothing) = 0$
2. $\mu(\cup_{i=1}^{k} A_i) = \sum_{i=1}^{k} \mu(A_i)$ *for any finite disjoint sequence $A_1, \ldots, A_k \in \mathcal{F}$*
3. $\mu(\cup_i A_i) \leqslant \sum_i \mu(A_i)$ *for any countable sequence $A_1, A_2, \ldots \in \mathcal{F}$,*

then μ extends to a unique measure on the σ-algebra generated by $\mathcal{F}$.

The proof of this theorem may be found in a standard text on probability and measure, for example [16]. It is straightforward to check that the 'measures' we define on cylinder sets later in the tutorial satisfy the hypotheses of the above theorem. Hence, these can be extended to measures used for the interpretation of the logics PCTL and CSL without ambiguity.

Definition 4. *Let $(\Omega, \mathcal{F}, \mu)$ be a* probability space*. A function $X : \Omega \to \mathbb{R}_{\geqslant 0}$ is said to be a* random variable*.*

Given a random variable $X : \Omega \to \mathbb{R}$ and the probability space $(\Omega, \mathcal{F}, \mu)$ the expectation or average value with respect to the measure μ is given by the following integral:

$$E[X] \stackrel{\text{def}}{=} \int_{\omega \in \Omega} X(\omega) \, \mathrm{d}\mu \, .$$

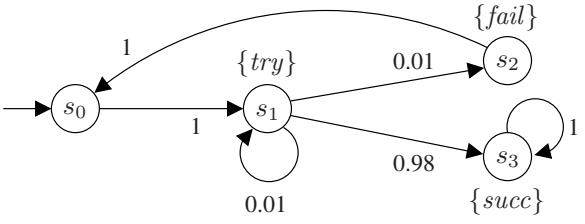


Fig. 1. The four state DTMC $\mathcal{D}_1$

3 Model Checking Discrete-Time Markov Chains

In this section we give an overview of the probabilistic model checking of *discrete-time Markov chains* (DTMCs). Let AP be a fixed, finite set of atomic propositions used to label states with properties of interest.

Definition 5. *A (labelled)* DTMC $\mathcal{D}$ *is a tuple* $(S, \bar{s}, \mathbf{P}, L)$ *where*

- S *is a finite set of* states*;*
- $\bar{s} \in S$ *is the* initial state*;*
- $\mathbf{P} : S \times S \to [0, 1]$ *is the* transition probability matrix *where* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ *for all* $s \in S$*;*
- $L : S \to 2^{AP}$ *is a* labelling *function which assigns to each state* $s \in S$ *the set* $L(s)$ *of atomic propositions that are valid in the state.*

Each element $\mathbf{P}(s, s')$ of the transition probability matrix gives the probability of making a transition from state s to state s'. Note that the probabilities on transitions emanating from a single state must sum to one. Terminating states, i.e. those from which the system cannot move to another state, can be modelled by adding a self-loop (a single transition going back to the same state with probability 1).

Example 1. Fig. 1 shows a simple example of a DTMC $\mathcal{D}_1 = (S_1, \bar{s}_1, \mathbf{P}_1, L_1)$. In our graphical notation, states are drawn as circles and transitions as arrows, labelled with their associated probabilities. The initial state is indicated by an additional incoming arrow. The DTMC $\mathcal{D}_1$ has four states: $S_1 = \{s_0, s_1, s_2, s_3\}$, with initial state $\bar{s} = s_0$. The transition probability matrix $\mathbf{P}_1$ is given by:

$$\mathbf{P}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The atomic propositions used to label states are taken from the set $AP = \{try, fail, succ\}$. Here, the DTMC models a simple process which tries to send a message. After one time-step, it enters the state s_1 from which, with probability 0.01 it waits another time-step, with probability 0.98 it successfully sends

the message, and with probability 0.01 it tries but fails to send the message. In the latter case, the process restarts. The labelling function allows us to assign meaningful names to states of the DTMC:

$$L_1(s_0) = \varnothing, \quad L_1(s_1) = \{try\}, \quad L_1(s_2) = \{fail\} \quad \text{and} \quad L_1(s_3) = \{succ\}\,.$$

3.1 Paths and Probability Measures

An execution of a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ is represented by a *path*. Formally, a path ω is a non-empty sequence of states $s_0 s_1 s_2 \ldots$ where $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geqslant 0$. A path can be either finite or infinite. We denote by $\omega(i)$ the ith state of a path ω, $|\omega|$ the length of ω (number of transitions) and for a finite path ω_{fin}, the last state by $last(\omega_{fin})$. We say that a finite path ω_{fin} of length n is a *prefix* of the infinite path ω if $\omega_{fin}(i) = \omega(i)$ for $0 \leqslant i \leqslant n$. The sets of all infinite and finite paths of $\mathcal{D}$ starting in state s are denoted $Path^{\mathcal{D}}(s)$ and $Path^{\mathcal{D}}_{fin}(s)$, respectively. Unless stated explicitly, we always deal with infinite paths.

In order to reason about the probabilistic behaviour of the DTMC, we need to determine the probability that certain paths are taken. This is achieved by defining, for each state $s \in S$, a probability measure Pr_s over the set of infinite paths $Path^{\mathcal{D}}(s)$. Below, we give an outline of this construction. For further details, see [41]. The probability measure is induced by the transition probability matrix $\mathbf{P}$ as follows. For any finite path $\omega_{fin} \in Path^{\mathcal{D}}_{fin}(s)$, we define the probability $\mathbf{P}_s(\omega_{fin})$:

$$\mathbf{P}_s(\omega_{fin}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n = 0 \\ \mathbf{P}(\omega(0), \omega(1)) \cdots \mathbf{P}(\omega(n-1), \omega(n)) & \text{otherwise} \end{cases}$$

where $n = |\omega_{fin}|$. Next, we define the *cylinder set* $C(\omega_{fin}) \subseteq Path^{\mathcal{D}}(s)$ as:

$$C(\omega_{fin}) \stackrel{\text{def}}{=} \{\omega \in Path^{\mathcal{D}}(s) \,|\, \omega_{fin} \text{ is a prefix of } \omega\}$$

that is, the set of all infinite paths with prefix ω_{fin}. Then, let $\Sigma_{Path^{\mathcal{D}}(s)}$ be the smallest σ-algebra (see Section 2) on $Path^{\mathcal{D}}(s)$ which contains all the sets $C(\omega_{fin})$, where ω_{fin} ranges over the finite paths $Path^{\mathcal{D}}_{fin}(s)$. As the set of cylinders form a semi-ring over $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$, we can apply Theorem 1 and define Pr_s on $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$ as the unique measure such that:

$$Pr_s(C(\omega_{fin})) = \mathbf{P}_s(\omega_{fin}) \text{ for all } \omega_{fin} \in Path^{\mathcal{D}}_{fin}(s)\,.$$

Note that, since $C(s) = Path^{\mathcal{D}}(s)$ and $\mathbf{P}_s(s) = 1$, it follows that Pr_s is a probability measure. We can now quantify the probability that, starting from a state $s \in S$, the DTMC $\mathcal{D}$ behaves in a specified fashion by identifying the set of paths which satisfy this specification and, assuming that this set is measurable, using the measure Pr_s.

Example 2. Consider again the DTMC $\mathcal{D}_1$ in Example 1 (see Fig. 1). There are five distinct paths of length 3 starting in state s_0. The probability measure of the cylinder sets associated with each of these is:

$$\begin{aligned}
Pr_{s_0}(C(s_0s_1s_1s_1)) &= 1.00 \cdot 0.01 \cdot 0.01 = 0.0001\\
Pr_{s_0}(C(s_0s_1s_1s_2)) &= 1.00 \cdot 0.01 \cdot 0.01 = 0.0001\\
Pr_{s_0}(C(s_0s_1s_1s_3)) &= 1.00 \cdot 0.01 \cdot 0.98 = 0.0098\\
Pr_{s_0}(C(s_0s_1s_2s_0)) &= 1.00 \cdot 0.01 \cdot 1.00 = 0.01\\
Pr_{s_0}(C(s_0s_1s_3s_3)) &= 1.00 \cdot 0.98 \cdot 1.00 = 0.98\,.
\end{aligned}$$

3.2 Probabilistic Computation Tree Logic (PCTL)

Specifications for DTMC models can be written in PCTL (Probabilistic Computation Tree Logic) [29], a probabilistic extension of the temporal logic CTL. PCTL is essentially the same as the logic pCTL of [6].

Definition 6. *The syntax of PCTL is as follows:*

$$\begin{aligned}
\Phi &::= \mathtt{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{P}_{\sim p}[\phi]\\
\phi &::= \mathtt{X}\ \Phi \mid \Phi\ \mathtt{U}^{\leqslant k}\ \Phi
\end{aligned}$$

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

PCTL formulae are interpreted over the states of a DTMC. For the presentation of the syntax, above, we distinguish between state formulae Φ and path formulae ϕ, which are evaluated over states and paths, respectively. To specify a property of a DTMC, we always use a state formula: path formulae only occur as the parameter of the $\mathtt{P}_{\sim p}[\cdot]$ operator. Intuitively, a state s of $\mathcal{D}$ satisfies $\mathtt{P}_{\sim p}[\phi]$ if the probability of taking a path from s satisfying ϕ is in the interval specified by $\sim p$. For this, we use the probability measure Pr_s over $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$ introduced in the previous section.

As path formulae we allow the $\mathtt{X}$ ('next') and $\mathtt{U}^{\leqslant k}$ ('bounded until') operators which are standard in temporal logic. The unbounded until is obtained by taking k equal to ∞, i.e. $\Phi\ \mathtt{U}\ \Psi = \Phi\ \mathtt{U}^{\leqslant\infty}\ \Psi$.

Intuitively, $\mathtt{X}\ \Phi$ is true if Φ is satisfied in the next state and $\Phi\ \mathtt{U}^{\leqslant k}\ \Psi$ is true if Ψ is satisfied within k time-steps and Φ is true up until that point.

For a state s and PCTL formula Φ, we write $s \models \Phi$ to indicate that s satisfies Φ. Similarly, for a path ω satisfying path formula ϕ, we write $\omega \models \phi$. The semantics of PCTL over DTMCs is defined as follows.

Definition 7. *Let $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ be a labelled DTMC. For any state $s \in S$, the satisfaction relation $\models$ is defined inductively by:*

$$\begin{aligned}
s \models \mathtt{true} \quad & \textit{for all } s \in S\\
s \models a \quad &\Leftrightarrow \quad a \in L(s)\\
s \models \neg\Phi \quad &\Leftrightarrow \quad s \not\models \Phi\\
s \models \Phi \wedge \Psi \quad &\Leftrightarrow \quad s \models \Phi \wedge s \models \Psi\\
s \models \mathtt{P}_{\sim p}[\phi] \quad &\Leftrightarrow \quad Prob^{\mathcal{D}}(s, \phi) \sim p
\end{aligned}$$

where:

$$Prob^{\mathcal{D}}(s,\phi) \stackrel{\text{def}}{=} Pr_s\{\omega \in Path^{\mathcal{D}}(s) \mid \omega \models \phi\}$$

and for any path $\omega \in Path^{\mathcal{D}}(s)$:

$$\begin{array}{rcl} \omega \models \mathtt{X}\ \Phi & \Leftrightarrow & \omega(1) \models \Phi \\ \omega \models \phi\ \mathtt{U}^{\leqslant k}\ \Psi & \Leftrightarrow & \exists i \in \mathbb{N}.\,(\, i \leqslant k \wedge \omega(i) \models \Psi \wedge \forall j{<}i.\,(\omega(j) \models \Phi)\,)\,. \end{array}$$

Note that, for any state s and path formula ϕ, the set $\{\omega \in Path^{\mathcal{D}}(s) \mid \omega \models \phi\}$ is a *measurable* set of $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$, see for example [62], and hence Pr_s is well defined over this set. From the basic syntax of PCTL, given above, we can derive a number of additional useful operators. Among these are the well known logical equivalences:

$$\begin{aligned} \mathtt{false} &\equiv \neg\mathtt{true} \\ \Phi \vee \Psi &\equiv \neg(\neg\Phi \wedge \neg\Psi) \\ \Phi \rightarrow \Psi &\equiv \neg\Phi \vee \Psi\,. \end{aligned}$$

We also allow path formulae to contain the $\Diamond$ ('diamond' or 'eventually') operator, which is common in temporal logic. Intuitively, $\Diamond\Phi$ means that Φ is eventually satisfied and its bounded variant $\Diamond^{\leqslant k}\Phi$ means that Φ is satisfied within k time units. These can be expressed in terms of the PCTL 'until' operator as follows:

$$\begin{aligned} \mathtt{P}_{\sim p}[\Diamond\ \Phi] &\equiv \mathtt{P}_{\sim p}[\mathtt{true}\ \mathtt{U}^{\leqslant\infty}\ \Phi] \\ \mathtt{P}_{\sim p}[\Diamond^{\leqslant k}\ \Phi] &\equiv \mathtt{P}_{\sim p}[\mathtt{true}\ \mathtt{U}^{\leqslant k}\ \Phi]\,. \end{aligned}$$

Another common temporal logic operator is $\Box$ ('box' or 'always'). A path satisfies $\Box\Phi$ when Φ is true in every state of the path. Similarly, the bounded variant $\Box^{\leqslant k}\Phi$ means that Φ is true in the first k states of the path. In theory, one can express $\Box$ in terms of $\Diamond$ as follows:

$$\begin{aligned} \Box\Phi &\equiv \neg\Diamond\neg\,\Phi \\ \Box^{\leqslant k}\Phi &\equiv \neg\Diamond^{\leqslant k}\neg\,\Phi\,. \end{aligned}$$

However, the syntax of PCTL does not allow negation of path formulae. Observing, though, that for a state s and path formula Φ, $Prob^{\mathcal{D}}(s,\neg\phi) = 1 - Prob^{\mathcal{D}}(s,\phi)$, we can show for example that:

$$\begin{aligned} \mathtt{P}_{\geqslant p}[\Box\ \Phi] &\Leftrightarrow Prob^{\mathcal{D}}(s,\Box\ \Phi) \geqslant p \\ &\Leftrightarrow 1 - Prob^{\mathcal{D}}(s,\Diamond\ \neg\Phi) \geqslant p \\ &\Leftrightarrow Prob^{\mathcal{D}}(s,\Diamond\ \neg\Phi) \leqslant 1 - p \\ &\Leftrightarrow \mathtt{P}_{\leqslant 1-p}[\Diamond\ \neg\Phi]\,. \end{aligned}$$

In fact, we have the following equivalences:

$$\begin{aligned} \mathtt{P}_{\sim p}[\Box\ \Phi] &\equiv \mathtt{P}_{\overline{\sim} 1-p}[\Diamond\ \neg\Phi] \\ \mathtt{P}_{\sim p}[\Box^{\leqslant k}\ \Phi] &\equiv \mathtt{P}_{\overline{\sim} 1-p}[\Diamond^{\leqslant k}\ \neg\Phi] \end{aligned}$$

where $\overline{<} \equiv >$, $\overline{\leqslant} \equiv \geqslant$,$\overline{\geqslant} \equiv \leqslant$ and $\overline{>} \equiv <$.

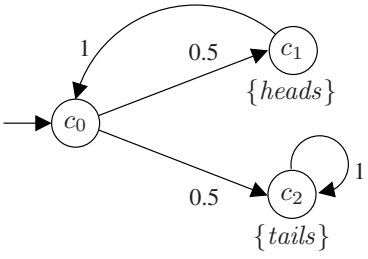


Fig. 2. Example demonstrating difference between $\mathtt{P}_{\geqslant 1}[\Diamond \Phi]$ and $\forall \Diamond \Phi$

The $\mathtt{P}_{\sim p}[\cdot]$ operator of PCTL can be seen as the probabilistic analogue of the path quantifiers of CTL. For example, the PCTL formula $\mathtt{P}_{\sim p}[\Diamond\ \Phi]$, which states that the probability of reaching a Φ-state is $\sim p$, is closely related to the CTL formulae $\forall \Diamond \Phi$ and $\exists \Diamond \Phi$ (sometimes written AF Φ and EF Φ), which assert that *all* paths or *at least one* path reach a Φ-state, respectively. In fact, we have the following equivalence:

$$\exists \Diamond \Phi \equiv \mathtt{P}_{>0}[\Diamond\ \Phi]$$

as the probability is greater than zero if and only if there exists a finite path leading to a Φ-state. Conversely, $\forall \Diamond \Phi$ and $\mathtt{P}_{\geqslant 1}[\Diamond\ \Phi]$ are not equivalent. For example, consider the DTMC in Fig. 2 which models a process which repeatedly tosses a fair coin until the result is 'tails'. State c_0 satisfies $\mathtt{P}_{\geqslant 1}[\Diamond\ \mathit{tails}]$ since the probability of reaching c_2 is one. There is, though, an (infinite) path $c_0c_1c_0c_1\ldots$ which never reaches state c_2. Hence, $\forall \Diamond\ \mathit{tails}$ is not satisfied in state c_0.

Example 3. Below are some typical examples of PCTL formulae:

- $\mathtt{P}_{\geqslant 0.4}[\mathtt{X}\ \mathit{delivered}]$ - the probability that a message has been delivered after one time-step is at least 0.4;
- $\mathit{init} \rightarrow \mathtt{P}_{\leqslant 0}[\Diamond\ \mathit{error}]$ - from any initial configuration, the probability that the system reaches an error state is 0;
- $\mathtt{P}_{\geqslant 0.9}[\neg \mathit{down}\ \mathtt{U}\ \mathit{served}]$ - the probability the system does not go down until after the request has been served is at least 0.9;
- $\mathtt{P}_{<0.1}[\neg \mathit{done}\ \mathtt{U}^{\leqslant 10}\ \mathit{fault}]$ - the probability that a fault occurs before the protocol finishes and within 10 time-steps is strictly less than 0.1.

A perceived weakness of PCTL is that it is not possible to determine the actual probability with which a certain path formula is satisfied, only whether or not the probability meets a particular bound. In fact, this restriction is in place purely to ensure that each PCTL formula evaluates to a Boolean. In practice, this constraint can be relaxed: if the outermost operator of a PCTL formula is $\mathtt{P}_{\sim p}$, we can omit the bound $\sim p$ and simply compute the probability instead. Since the algorithm for PCTL model checking proceeds by computing the actual probability and then comparing it to the bound, no additional work is needed. It is also often useful to study a range of such values by varying one or more parameters, either of the model or of the property. Both these observations can be seen in practice in Section 5.

3.3 Model Checking PCTL

We now summarise a model checking algorithm for PCTL over DTMCs, which was first presented in [22,29,23]. The inputs to the algorithm are a labelled DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ and a PCTL formula Φ. The output is the set of states $Sat(\Phi) = \{s \in S \mid s \models \Phi\}$, i.e. the set containing all the states of the model which satisfy Φ. In a typical scenario, we may only be interested in whether the initial state $\bar{s}$ of the DTMC satisfies Φ. However, the algorithm works by checking whether each state in S satisfies the formula.

The overall structure of the algorithm is identical to the model checking algorithm for CTL [20], the non-probabilistic temporal logic on which PCTL is based. We first construct the parse tree of the formula Φ. Each node of this tree is labelled with a subformula of Φ, the root node is labelled with Φ itself and leaves of the tree will be labelled with either $\mathtt{true}$ or an atomic proposition a. Working upwards towards the root of the tree, we recursively compute the set of states satisfying each subformula. By the end, we have determined whether each state in the model satisfies Φ. The algorithm for PCTL formulae can be summarised as follows:

$$\begin{aligned} Sat(\mathtt{true}) &= S \\ Sat(a) &= \{s \mid a \in L(s)\} \\ Sat(\neg\Phi) &= S \backslash Sat(\Phi) \\ Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\ Sat(\mathtt{P}_{\sim p}[\phi]) &= \{s \in S \mid Prob^{\mathcal{D}}(s, \phi) \sim p\}\,. \end{aligned}$$

Model checking for the majority of these formulae is trivial to implement and is, in fact, the same as for the non-probabilistic logic CTL. The exceptions are formulae of the form $\mathtt{P}_{\sim p}[\phi]$. For these, we must calculate, for all states s of the DTMC, the probability $Prob^{\mathcal{D}}(s, \phi)$ and then compare these values to the bound in the formula. In the following sections, we describe how to compute these values for the two cases: $\mathtt{P}_{\sim p}[\mathtt{X}\ \Phi]$ and $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leqslant k}\ \Psi]$. Because of the recursive nature of the PCTL model checking algorithm, we can assume that the relevant sets, $Sat(\Phi)$ or $Sat(\Phi)$ and $Sat(\Psi)$, are already known.

$\mathtt{P}_{\sim p}[\mathtt{X}\ \Phi]$ formulae. In this case, we need to compute the probability $Prob^{\mathcal{D}}(s, \mathtt{X}\ \Phi)$ for each state $s \in S$. This requires the probabilities of the immediate transitions from s:

$$Prob^{\mathcal{D}}(s, \mathtt{X}\ \Phi) = \sum_{s' \in Sat(\Phi)} \mathbf{P}(s, s')\,.$$

We determine the vector $\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \Phi)$ of probabilities for all states as follows. Assuming that we have a state-indexed column vector $\underline{\Phi}$ with

$$\underline{\Phi}(s) = \begin{cases} 1 & \text{if } s \in Sat(\Phi) \\ 0 & \text{otherwise,} \end{cases}$$

then $\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \Phi)$ is computed using the single matrix-vector multiplication:

$$\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \Phi) = \mathbf{P} \cdot \underline{\Phi}\,.$$

Example 4. Consider the PCTL formula $\mathtt{P}_{\geqslant 0.9}[\mathtt{X}\ (\neg try \vee succ)]$ and the DTMC $\mathcal{D}_1$ from Fig. 1. Proceeding recursively from the innermost subformulae, we compute:

$$\begin{aligned} Sat(try) &= \{s_1\} \\ Sat(succ) &= \{s_3\} \\ Sat(\neg succ) &= S \setminus Sat(succ) = \{s_0, s_1, s_2\} \\ Sat(try \wedge \neg succ) &= Sat(try) \cap Sat(\neg succ) = \{s_1\} \cap \{s_0, s_1, s_2\} = \{s_1\} \\ Sat(\neg try \vee succ) &= Sat(\neg(try \wedge \neg succ)) = S \setminus Sat(try \wedge \neg succ) = \{s_0, s_2, s_3\}\,. \end{aligned}$$

This leaves the $\mathtt{X}$ operator, and from above we have $\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \neg try \vee succ)$ equals:

$$\mathbf{P}_1 \cdot \underline{\neg try \vee succ} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.99 \\ 1 \\ 1 \end{pmatrix},$$

and hence $Sat(\mathtt{P}_{\geqslant 0.9}[\mathtt{X}\ (\neg try \vee succ)]) = \{s_1, s_2, s_3\}$.

$\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leqslant k}\ \Psi]$ formulae. For such formulae we need to determine the probabilities $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ for all states s where $k \in \mathbb{N} \cup \{\infty\}$. We begin by considering the case when $k \in \mathbb{N}$.

Case when $k \in \mathbb{N}$. This amounts to computing the solution of the following set of equations. For $s \in S$ and $k \in \mathbb{N}$: $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ equals

$$\begin{cases} 1 & \text{if } s \in Sat(\Psi) \\ 0 & \text{if } k{=}\,0 \text{ or } s \in Sat(\neg\Phi \wedge \neg\Psi) \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot Prob^{\mathcal{D}}(s', \Phi\ \mathtt{U}^{\leqslant k-1}\ \Psi) & \text{otherwise.} \end{cases} \quad (1)$$

We now show how these probabilities can be expressed in terms of the transient probabilities of a DTMC. We denote by $\pi^{\mathcal{D}}_{s,k}(s')$ the transient probability in $\mathcal{D}$ of being in state s' after k steps when starting in s, that is:

$$\pi^{\mathcal{D}}_{s,k}(s') = \Pr_s\{\omega \in Path^{\mathcal{D}}(s) \mid \omega(k) = s'\}\,,$$

and require the following PCTL driven transformation of DTMCs.

Definition 8. *For any DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ and PCTL formula Φ, let $\mathcal{D}[\Phi] = (S, \bar{s}, \mathbf{P}[\Phi], L)$ where, if $s \not\models \Phi$, then $\mathbf{P}[\Phi](s, s') = \mathbf{P}(s, s')$ for all $s' \in S$, and if $s \models \Phi$, then $\mathbf{P}[\Phi](s, s) = 1$ and $\mathbf{P}[\Phi](s, s') = 0$ for all $s' \neq s$.*

Using the transient probabilities and this transformation we can characterise the probabilities $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ as follows.

Proposition 2. *For any DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$, state $s \in S$, PCTL formulae Φ and Ψ, and $k \in \mathbb{N}$:*

$$Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi) = \sum_{s' \models \Psi} \pi^{\mathcal{D}[\neg\Phi \vee \Psi]}_{s,k}(s')\,.$$

Note that $\mathcal{D}[\neg\Phi \vee \Psi] = \mathcal{D}[\neg(\Phi \wedge \Psi)][\Psi]$, that is all states in $Sat(\neg(\Phi \wedge \Psi))$ and $Sat(\Psi)$ are made absorbing (the only transitions available in these states are self-loops). States in $Sat(\neg(\Phi \wedge \Psi))$ are made absorbing because, for any state s in this set, $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ is trivially 0 as neither Φ nor Ψ is satisfied in s, since no path starting in s can possibly satisfy the path formula $\Phi\ \mathtt{U}^{\leqslant k}\ \Psi$. On the other hand, states in $Sat(\Psi)$ are made absorbing because, for any state s in this set, we have $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ is trivially 1 since all paths leaving s clearly satisfy $\Phi\ \mathtt{U}^{\leqslant k}\ \Psi$.

Using Proposition 2, the vector of probabilities $\underline{Prob}^{\mathcal{D}}(\Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ can then be computed using the following matrix and vector multiplications:

$$\underline{Prob}^{\mathcal{D}}(\Phi\ \mathtt{U}^{\leqslant k}\ \Psi) = (\mathbf{P}[\neg\Phi \vee \Psi])^k \cdot \underline{\Psi}\,.$$

This product is typically computed in an iterative fashion:

$$\mathbf{P}[\neg\Phi \vee \Psi] \cdot (\cdots(\mathbf{P}[\neg\Phi \vee \Psi] \cdot \underline{\Psi})\cdots)$$

which requires k matrix-vector multiplications. An alternative is to precompute the matrix power $(\mathbf{P}[\neg\Phi \vee \Psi])^k$ and then perform a single matrix-vector multiplication. In theory, this could be more efficient since the matrix power could be computed by repeatedly squaring $\mathbf{P}[\neg\Phi \vee \Psi]$, requiring approximately $log_2 k$, as opposed to k, multiplications. In practice, however, the matrix $\mathbf{P}[\neg\Phi \vee \Psi]$ is large and sparse, and therefore employing such an approach can dramatically increase the number of non-zero matrix elements, having serious implications for both time and memory usage.

Example 5. Let us return to the DTMC $\mathcal{D}_1$ in Fig. 1 and consider the PCTL formula $\mathtt{P}_{>0.98}[\Diamond^{\leqslant 2} succ]$. This is equivalent to the formula $\mathtt{P}_{>0.98}[\mathtt{true}\ \mathtt{U}^{\leqslant 2}\ succ]$. We have:

$$Sat(\mathtt{true}) = \{s_0, s_1, s_2, s_3\} \ \text{ and } \ Sat(succ) = \{s_3\}\,.$$

The matrix $\mathbf{P}_1[\neg\mathtt{true} \vee succ]$, is identical to $\mathbf{P}_1$, and we have that:

$$\begin{aligned}
&\underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 0}\ \Psi) = \underline{succ} = [0,0,0,1]\\
&\underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 1}\ \Psi) = \mathbf{P}_1[\neg\mathtt{true} \vee succ] \cdot \underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 0}\ \Psi) = [0, 0.98, 0, 1]\\
&\underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 2}\ \Psi) = \mathbf{P}_1[\neg\mathtt{true} \vee succ] \cdot \underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 1}\ \Psi) = [0.98, 0.9898, 0, 1]\,.
\end{aligned}$$

Hence, $Sat(\mathtt{P}_{>0.98}[\Diamond^{\leqslant 2} succ]) = \{s_1, s_3\}$.

Case when $k = \infty$. Note that, instead of $\mathtt{U}^{\leqslant\infty}$, we use the standard notation $\mathtt{U}$ for unbounded until. The probabilities $\underline{Prob}^{\mathcal{D}}(s, \Phi\ \mathtt{U}\ \Psi)$ can be computed as the least solution of the linear equation system:

$$Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}\ \Psi) = \begin{cases} 1 & \text{if } s \in Sat(\Psi)\\ 0 & \text{if } s \in Sat(\neg\Phi \wedge \neg\Psi)\\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot Prob^{\mathcal{D}}(s', \Phi\ \mathtt{U}\ \Psi) & \text{otherwise.}\end{cases}$$

PROB0($Sat(\Phi), Sat(\Psi)$)

1. $R := Sat(\Psi)$
2. $done :=$ **false**
3. **while** ($done =$ **false**)
4. $\quad R' := R \cup \{s \in Sat(\Phi) \mid \exists s' \in R \,.\, \mathbf{P}(s,s')>0\}$
5. $\quad$ **if** $(R' = R)$ **then** $done :=$ **true**
6. $\quad R := R'$
7. **endwhile**
8. **return** $S \backslash R$

PROB1($Sat(\Phi), Sat(\Psi), Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi])$)

1. $R := Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi])$
2. $done :=$ **false**
3. **while** ($done =$ **false**)
4. $\quad R' := R \cup \{s \in (Sat(\Phi) \backslash Sat(\Psi)) \mid \exists s' \in R \,.\, \mathbf{P}(s,s')>0\}$
5. $\quad$ **if** $(R' = R)$ **then** $done :=$ **true**
6. $\quad R := R'$
7. **endwhile**
8. **return** $S \backslash R$

Fig. 3. The PROB0 and PROB1 algorithm

To simplify the computation we transform this system of equations into one with a unique solution. This is achieved by first finding *all* the states s for which $Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)$ is exactly 0 or 1; more precisely, we compute the sets of states:

$$Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi]) = \{s \in S \mid Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)=0\}$$
$$Sat(\mathtt{P}_{\geqslant 1}[\Phi \mathtt{U} \Psi]) = \{s \in S \mid Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)=1\}\,.$$

These sets can be determined with the algorithms PROB0 and PROB1 which are described in Fig. 3:

$$Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi]) = \text{PROB0}(Sat(\Phi), Sat(\Psi))$$
$$Sat(\mathtt{P}_{\geqslant 1}[\Phi \mathtt{U} \Psi]) = \text{PROB1}(Sat(\Phi), Sat(\Psi), Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi]))\,.$$

PROB0 computes all the states from which it is possible, with *non-zero* probability, to reach a state satisfying Ψ without leaving states satisfying Φ. It then subtracts these from S to determine the states which have a *zero* probability. PROB1 first determines the set of states for which the probability is *less than* 1 of reaching a state satisfying Ψ without leaving states satisfying Φ. These are the states from which there is a non-zero probability of reaching a state in $Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi])$, passing only through states satisfying Φ but not Ψ. It then subtracts this set from S to produce $Sat(\mathtt{P}_{\geqslant 1}[\Phi \mathtt{U} \Psi])$. Note that both algorithms are based on the computation of a fixpoint operator, and hence require at most $|S|$ iterations.

The probabilities $Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)$ can then be computed as the unique solution of the following linear equation system:

$$Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi) = \begin{cases} 1 & \text{if } s \in Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi]) \\ 0 & \text{if } s \in Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi]) \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot Prob^{\mathcal{D}}(s', \Phi \,\mathtt{U}\, \Psi) & \text{otherwise.} \end{cases}$$

Since the probabilities for the sets of states $Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])$ and $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi])$ are known, i.e. 1 and 0 respectively, it is possible to solve the linear equation system over only the set of states $S^? = S \setminus (Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi]) \cup Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi]))$:

$$Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi) = \sum_{s' \in S^?} \mathbf{P}(s, s') \cdot Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi) + \sum_{s' \in Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])} \mathbf{P}(s, s')$$

reducing the number of unknowns from $|S|$ to $|S^?|$.

In either case, the linear equation system can be solved by any standard approach. These include direct methods, such as Gaussian elimination and L/U decomposition, or iterative methods, such as Jacobi and Gauss-Seidel. The algorithms Prob0 and Prob1 form the first part of the calculation of $Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi)$. For this reason, we refer to them as *precomputation algorithms*. For *qualitative* PCTL properties (i.e. where the bound p in the formula $\mathtt{P}_{\sim p}[\Phi \,\mathtt{U}\, \Psi]$ is either 0 or 1) or for cases where $Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi)$ happens to be either 0 or 1 for all states (i.e. $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi]) \cup Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi]) = S$), it suffices to use these precomputation algorithms alone. For *quantitative* properties with an arbitrary bound p, numerical computation is also usually required. The precomputation algorithms are still valuable in this case. Firstly, they can reduce the number of states for which numerical computation is required. Secondly, they determine the exact probability for the states in $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi])$ and $Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])$, whereas numerical computation typically computes an approximation and is subject to round-off errors.

Finally we note that, if desired, the Prob1 algorithm can be omitted and $Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])$ replaced by $Sat(\Psi)$. The set $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi])$, however, must be computed to ensure that the linear equation system has a unique solution.

Example 6. Consider again the DTMC $\mathcal{D}_1$ in Fig. 1 and the PCTL formula $\mathtt{P}_{>0.99}[try \,\mathtt{U}\, succ]$. We have $Sat(try) = \{s_1\}$ and $Sat(succ) = \{s_3\}$. Prob0 determines in two iterations that $Sat(\mathtt{P}_{\leqslant 0}[try \,\mathtt{U}\, succ]) = \{s_0, s_2\}$. Prob1 determines that $Sat(\mathtt{P}_{\geqslant 1}[try \,\mathtt{U}\, succ]) = \{s_3\}$. The resulting linear equation system is:

$$\begin{aligned} Prob^{\mathcal{D}_1}(s_0, try \,\mathtt{U}\, succ) &= 0 \\ Prob^{\mathcal{D}_1}(s_1, try \,\mathtt{U}\, succ) &= 0.01 \cdot Prob^{\mathcal{D}_1}(s_1, try \,\mathtt{U}\, succ) + \\ &\quad\; 0.01 \cdot Prob^{\mathcal{D}_1}(s_2, try \,\mathtt{U}\, succ) + \\ &\quad\; 0.98 \cdot Prob^{\mathcal{D}_1}(s_3, try \,\mathtt{U}\, succ) \\ Prob^{\mathcal{D}_1}(s_2, try \,\mathtt{U}\, succ) &= 0 \\ Prob^{\mathcal{D}_1}(s_3, try \,\mathtt{U}\, succ) &= 1\,. \end{aligned}$$

This yields the solution $\underline{Prob}^{\mathcal{D}}(try \,\mathtt{U}\, succ) = (0, \frac{98}{99}, 0, 1)$ and we see that the formula $\mathtt{P}_{>0.99}[try \,\mathtt{U}\, succ]$ is satisfied only in state s_3.

3.4 Extending DTMCs and PCTL with Rewards

In this section we enhance DTMCs with reward (or cost) structures and extend PCTL to allow for specifications over reward structures. For a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ a reward structure $(\underline{\rho}, \boldsymbol{\iota})$ allows one to specify two distinct types of rewards: *state* rewards, which are assigned to states by means of the reward function $\underline{\rho} : S \to \mathbb{R}_{\geqslant 0}$, and *transition* rewards, which are assigned to transitions by means of the reward function $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geqslant 0}$. The state reward $\underline{\rho}(s)$ is the reward acquired in state s per time-step, i.e. a reward of $\underline{\rho}(s)$ is incurred if the DTMC is in state s for 1 time-step and the transition reward $\boldsymbol{\iota}(s, s')$ is acquired each time a transition between states s and s' occurs.

A reward structure can be used to represent additional information about the system the DTMC represents, for example the power consumption, number of packets sent or the number of lost requests. Note that state rewards are sometimes called cumulative rewards while transition rewards are sometimes referred to as instantaneous or impulse rewards.

Example 7. Returning to Example 1 which describes the DTMC $\mathcal{D}_1$ of Fig. 1, consider the reward structure $(\underline{\rho}^{\mathcal{D}_1}, \mathbf{0})$, where $\underline{\rho}^{\mathcal{D}_1}(s) = 1$ if $s = s_1$ and equals 0 otherwise. This particular reward structure would be useful when we are interested in the number of time-steps spent in state s_1 or the chance that one is in state s_1 after a certain number of time-steps.

The logic PCTL is extended to allow for the reward properties by means of the following state formulae:

$$\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$$

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

Intuitively, a state s satisfies $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}]$ if, from state s, the expected reward *cumulated* after k time-steps satisfies $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$ is true if from state s the expected state reward at time-step k meets the bound $\sim r$; and $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$ is true if from state s the expected reward cumulated before a state satisfying Φ is reached meets the bound $\sim r$.

Formally, given a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$, the semantics of these formulae is defined as follows. For any $s \in S$, $k \in \mathbb{N}$, $r \in \mathbb{R}_{\geqslant 0}$ and PCTL formula Φ:

$$\begin{aligned} s \models \mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}] \quad &\Leftrightarrow \quad Exp^{\mathcal{D}}(s, X_{\mathtt{C}^{\leqslant k}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] \quad &\Leftrightarrow \quad Exp^{\mathcal{D}}(s, X_{\mathtt{I}^{=k}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi] \quad &\Leftrightarrow \quad Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi}) \sim r \end{aligned}$$

where $Exp^{\mathcal{D}}(s, X)$ denotes the expectation of the random variable $X : Path^{\mathcal{D}}(s) \to \mathbb{R}_{\geqslant 0}$ with respect to the probability measure Pr_s, and for any path $\omega = s_0 s_1 s_2 \cdots \in Path^{\mathcal{D}}(s)$:

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

$$X_{\mathtt{I}^{=k}}(\omega) \stackrel{\text{def}}{=} \underline{\rho}(s_k)$$

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

Example 8. Below are some typical examples of reward based specifications using these formulae:

- $\mathtt{R}_{\leqslant 5.5}[\mathtt{C}^{\leqslant 100}]$ - the expected power consumption within the first 100 time-steps of operation is less than or equal to 5.5;
- $\mathtt{R}_{\geqslant 4}[\mathtt{I}^{=10}]$ - the expected number of messages still to be delivered after 10 time-steps have passed is at least 4;
- $\mathtt{R}_{\geqslant 14}[\mathtt{F}\ \mathit{done}]$ - the expected number of correctly delivered messages is at least 14.

We now consider the computation of the expected values for each of the random variables introduced above.

The random variable $X_{\mathtt{C}^{\leqslant k}}$. In this case the computation of the expected values $\mathit{Exp}^{\mathcal{D}}(s, X_{\mathtt{C}^{\leqslant k}})$ for all $s \in S$ is based on the following set of equations:

$$\mathit{Exp}^{\mathcal{D}}(s, X_{\mathtt{C}^{\leqslant k}}) = \begin{cases} 0 & \text{if } k = 0 \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot \big(\boldsymbol{\iota}(s, s') + \mathit{Exp}^{\mathcal{D}}(s', X_{\mathtt{C}^{\leqslant k-1}})\big) & \text{otherwise.} \end{cases}$$

More precisely, one can iteratively compute the vector of expected values by means of the following matrix-vector operations:

$$\underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant k}}) = \begin{cases} \underline{0} & \text{if } k = 0 \\ \underline{\rho} + \mathbf{P} \cdot \big(\boldsymbol{\iota} \cdot \underline{1} + \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant k-1}})\big) & \text{otherwise} \end{cases}$$

where $\underline{1}$ denotes a vector with all entries equal to 1.

Example 9. Let us return to the DTMC $\mathcal{D}_1$ of Example 1 (see Fig. 1) and reward structure of Example 9. The PCTL formula $\mathtt{R}_{>1}[\mathtt{C}^{\leqslant k}]$ in this case states that, after k time steps, the expected number of time steps spent in state s_1 is greater than 1. Now from above:

$$\begin{aligned} \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 0}}) &= [0, 0, 0, 0] \\ \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 1}}) &= \underline{\rho} + \mathbf{P} \cdot \big(\boldsymbol{\iota} + \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 0}})\big) \\ &= [0, 1, 0, 0] + \mathbf{P} \cdot \big(\mathbf{0} \cdot \underline{1} + [0, 0, 0, 0]\big) \\ &= [0, 1, 0, 0] \\ \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 2}}) &= \underline{\rho} + \mathbf{P} \cdot \big(\boldsymbol{\iota} + \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 1}})\big) \\ &= [0, 1, 0, 0] + \mathbf{P} \cdot \big(\mathbf{0} \cdot \underline{1} + [0, 1, 0, 0]\big) \\ &= [1, 1.01, 0, 0] \end{aligned}$$

and hence $\mathit{Sat}(\mathtt{R}_{>1}[\mathtt{C}^{\leqslant 2}]) = \{s_1\}$.

The random variable $X_{\mathtt{I}=k}$. In this case the expected value can be computed iteratively through the following set of equations:

$$Exp^{\mathcal{D}}(s, X_{\mathtt{I}=k}) = \begin{cases} \underline{\rho}(s) & \text{if } k = 0 \\ \sum_{s' \in S} \mathbf{P} \cdot Exp^{\mathcal{D}}(s, X_{\mathtt{I}=k-1}) & \text{otherwise.} \end{cases}$$

Therefore, the vector $\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=k})$ can be computed by means of the following matrix-vector operations:

$$\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=k}) = \begin{cases} \underline{\rho} & \text{if } k = 0 \\ \mathbf{P} \cdot \underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=k-1}) & \text{otherwise.} \end{cases}$$

Example 10. Returning again to the DTMC $\mathcal{D}_1$ of Example 1 and reward structure of Example 9. In this case, the PCTL formula $\mathtt{R}_{>0}[\mathtt{I}^{=k}]$ specifies that, at time-step k, the expectation of being in state s_1 is greater than 0. We have:

$$\begin{aligned} &\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=0}) = [0, 1, 0, 0] \\ &\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=1}) = \mathbf{P} \cdot \underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=0}) = \mathbf{P} \cdot [0, 1, 0, 0] = [1, 0.01, 0, 0] \\ &\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=2}) = \mathbf{P} \cdot \underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=1}) = \mathbf{P} \cdot [1, 0.01, 0, 0] = [0.01, 0.0001, 1, 0]\,. \end{aligned}$$

Hence, the states s_0, s_1 and s_2 satisfy the formula $\mathtt{R}_{>0}[\mathtt{I}^{=2}]$.

The random variable $X_{\mathtt{F}\Phi}$. The expectations in this case are a solution of the following system of linear equations:

$$Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi}) = \begin{cases} 0 & \text{if } s \in Sat(\Phi) \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot \big(\boldsymbol{\iota}(s, s') + Exp^{\mathcal{D}}(s', X_{\mathtt{F}\Phi}) \big) & \text{otherwise.} \end{cases}$$

As above, to simplify the computation this system of equations is transformed into one for which the expectations $Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi})$ are the unique solution. To achieve this, we identify the sets of states for which $Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi})$ equals ∞. This set of states are simply the set of states for which the probability of reaching a Φ state is less than 1, that is, the set $Sat(\mathtt{P}_{<1}[\Diamond\, \Phi])$. We compute this set using the precomputation algorithms Prob1 and Prob0 described in the previous section and the equivalence $\mathtt{P}_{<1}[\Diamond\, \Phi] \equiv \neg \mathtt{P}_{\geqslant 1}[\Diamond\, \Phi]$. One can then compute $Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi})$ as the unique solution of the following linear equation system:

$$Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi}) = \begin{cases} 0 & \text{if } s \in Sat(\Phi) \\ \infty & \text{if } s \in Sat(\mathtt{P}_{<1}[\Diamond\, \Phi]) \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot \big(\boldsymbol{\iota}(s, s') + Exp^{\mathcal{D}}(s', X_{\mathtt{F}\Phi}) \big) & \text{otherwise.} \end{cases}$$

As for 'until' formulae, this can be solved using any standard direct or iterative method.

Example 11. Let us return to $\mathcal{D}_1$ of Example 1 and the reward structure in Example 9. The PCTL formula, $\mathtt{R}_{<1}[\mathtt{F}\ succ]$, in this case, asserts that the expected number of times state s_1 is entered before reaching a state satisfying *succ* is less than 1. Following the procedure outlined above, we compute:

$$\begin{aligned}
Sat(succ) &= \{s_3\} \\
Sat(\mathtt{P}_{<1}[\Diamond\ succ]) &= Sat(\neg \mathtt{P}_{\geqslant 1}[\Diamond\ succ]) \\
&= S \setminus Sat(\mathtt{P}_{\geqslant 1}[\Diamond\ succ]) \\
&= S \setminus \textsc{Prob1}(S, Sat(succ), Sat(\mathtt{P}_{\leqslant 0}[\Diamond\ succ])) \\
&= S \setminus \textsc{Prob1}(S, Sat(succ), \textsc{Prob0}(Sat(\mathtt{true}), Sat(succ))) \\
&= S \setminus \textsc{Prob1}(S, Sat(succ), \varnothing) \\
&= S \setminus \{s_0, s_1, s_2, s_3\} = \varnothing.
\end{aligned}$$

This leads to the linear equation system:

$$\begin{aligned}
Exp^{\mathcal{D}}(s_0, X_{\mathtt{F}succ}) &= 0 + 1.00 \cdot \big(0 + Exp^{\mathcal{D}}(s_1, X_{\mathtt{F}succ})\big) \\
Exp^{\mathcal{D}}(s_1, X_{\mathtt{F}succ}) &= 1 + 0.01 \cdot \big(0 + Exp^{\mathcal{D}}(s_1, X_{\mathtt{F}succ})\big) + 0.01 \cdot \big(0 + Exp^{\mathcal{D}}(s_2, X_{\mathtt{F}succ})\big) \\
Exp^{\mathcal{D}}(s_2, X_{\mathtt{F}succ}) &= 0 + 1.00 \cdot \big(0 + Exp^{\mathcal{D}}(s_0, X_{\mathtt{F}succ})\big) \\
Exp^{\mathcal{D}}(s_3, X_{\mathtt{F}succ}) &= 0
\end{aligned}$$

which has the solution $\underline{Exp}^{\mathcal{D}}(X_{\mathtt{F}succ}) = \big(\frac{100}{98}, \frac{100}{98}, \frac{100}{98}, 0\big)$, and hence it follows that $Sat(\mathtt{R}_{<1}[\mathtt{F}\ succ]) = \overline{\{s_3\}}$.

3.5 Complexity of PCTL Model Checking

The overall time complexity for model checking a PCTL formula Φ against a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ is linear in $|\Phi|$ and polynomial in $|S|$. The size $|\Phi|$ of a formula Φ is, as defined in [29], equal to the number of logical connectives and temporal operators in the formula plus the sum of the sizes of the temporal operators. Because of the recursive nature of the algorithm, we perform model checking for each of the $|\Phi|$ operators and each one is at worst polynomial in $|S|$. The most expensive of these are the operators $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}\ \Psi]$ and $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$, for which a system of linear equations of size at most $|S|$ must be solved. This can be done with Gaussian elimination, the complexity of which is cubic in the size of the system. Strictly speaking, the complexity of model checking is also linear in $k_{\max}$, the maximum value of k found in formulae of type $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leqslant k}\ \Psi]$, $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}]$ or $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$. In practice, k is usually much smaller than $|S|$.

4 Model Checking Continuous-Time Markov Chains

This section concerns model checking continuous-time Markov chains (CTMCs) against the logic Continuous Stochastic Logic (CSL). While each transition between states in a DTMC corresponds to a discrete time-step, in a CTMC transitions occur in real time. As for the case of DTMCs, we suppose that we have a fixed set of atomic propositions AP.

Definition 9. *A (labelled)* CTMC *is a tuple* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *where:*

- S *is a finite set of* states*;*
- $\bar{s} \in S$ *is the* initial state*;*
- $\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ *is the* transition rate matrix*;*
- $L : S \to 2^{AP}$ *is a* labelling *function which assigns to each state* $s \in S$ *the set* $L(s)$ *of atomic propositions that are valid in the state.*

The transition rate matrix $\mathbf{R}$ assigns rates to each pair of states in the CTMC, which are used as parameters of the exponential distribution. A transition can only occur between states s and s' if $\mathbf{R}(s, s')>0$ and, in this case, the probability of this transition being triggered within t time-units equals $1 - e^{-\mathbf{R}(s,s')\cdot t}$. Typically, in a state s, there is more than one state s' for which $\mathbf{R}(s, s')>0$. This is known as a *race condition*. The first transition to be triggered determines the next state of the CTMC. The time spent in state s, before any such transition occurs, is exponentially distributed with rate $E(s)$, where:

$$E(s) \stackrel{\text{def}}{=} \sum_{s' \in S} \mathbf{R}(s, s') \,.$$

$E(s)$ is known as the *exit rate* of state s. A state s is called *absorbing* if $E(s) = 0$, i.e. if it has no outgoing transitions. We can also determine the actual probability of each state s' being the next state to which a transition is made from state s, independent of the time at which this occurs. This is defined by the following DTMC.

Definition 10. *The* embedded DTMC *of a CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *is the DTMC* $emb(\mathcal{C}) = (S, \bar{s}, \mathbf{P}^{emb(\mathcal{C})}, L)$ *where for* $s, s' \in S$*:*

$$\mathbf{P}^{emb(\mathcal{C})}(s, s') = \begin{cases} \frac{\mathbf{R}(s,s')}{E(s)} & \text{if } E(s) \neq 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise.} \end{cases}$$

Using the above definitions, we can consider the behaviour of the CTMC in an alternative way. It will remain in a state s for a delay which is exponentially distributed with rate $E(s)$ and then make a transition. The probability that this transition is to state s' is given by $\mathbf{P}^{emb(\mathcal{C})}(s, s')$.

We also define the following matrix, which will be used when we perform analysis of the CTMC.

Definition 11. *The* infinitesimal generator matrix *for the CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *is the matrix* $\mathbf{Q} : S \times S \to \mathbb{R}$ *defined as:*

$$\mathbf{Q}(s, s') = \begin{cases} \mathbf{R}(s, s') & \text{if } s \neq s' \\ -\sum_{s'' \neq s} \mathbf{R}(s, s'') & \text{otherwise.} \end{cases}$$

Example 12. Fig. 4 shows a simple example of a CTMC $\mathcal{C}_1 = (S_1, \bar{s}_1, \mathbf{R}_1, L)$. The graphical notation is as for DTMCs, except that transitions are now labelled

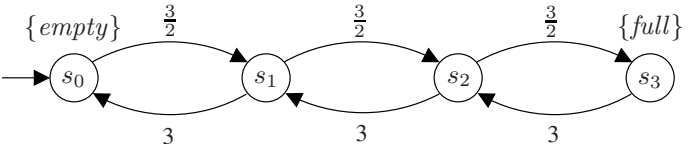


Fig. 4. The four state CTMC $\mathcal{C}_1$

with rates rather than probabilities. The CTMC models a queue of jobs: there are four states s_0, s_1, s_2 and s_3, where state s_i indicates that there are i jobs in the queue. Initially, the queue is empty ($\bar{s} = s_0$) and the maximum size is 3. Jobs arrive with rate $\frac{3}{2}$ and are removed from the queue with rate 3. The associated transition rate matrix $\mathbf{R}_1$, transition probability matrix $\mathbf{P}_1^{emb(\mathcal{C}_1)}$ for the embedded DTMC and infinitesimal generator matrix $\mathbf{Q}_1$ are as follows:

$$\mathbf{R}_1 = \begin{pmatrix} 0 & \frac{3}{2} & 0 & 0 \\ 3 & 0 & \frac{3}{2} & 0 \\ 0 & 3 & 0 & \frac{3}{2} \\ 0 & 0 & 3 & 0 \end{pmatrix} \quad \mathbf{P}_1^{emb(\mathcal{C}_1)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{3} & 0 \\ 0 & \frac{2}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \mathbf{Q}_1 = \begin{pmatrix} -\frac{3}{2} & \frac{3}{2} & 0 & 0 \\ 3 & -\frac{9}{2} & \frac{3}{2} & 0 \\ 0 & 3 & -\frac{9}{2} & \frac{3}{2} \\ 0 & 0 & 3 & -3 \end{pmatrix}.$$

We have labelled the state s_0, where the queue contains no jobs, with the atomic proposition *empty* and the state s_3, where it has the maximum number of jobs, with *full*. These are also illustrated in Fig. 4.

4.1 Paths and Probability Measures

An infinite path of a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ is a non-empty sequence $s_0 t_0 s_1 s_2 \ldots$ where $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$ for all $i \geqslant 0$. A finite path is a sequence $s_0 t_0 s_1 t_1 s_{k-1} t_{k-1} s_k$ such that s_k is absorbing. The value t_i represents the amount of time spent in the state s_i. As with DTMCs, we denote by $\omega(i)$ the ith state of a path ω, namely s_i. For an infinite path ω, we denote by $time(\omega, i)$ the amount of time spent in state s_i, namely t_i, and by $\omega@t$ the state occupied at time t, i.e. $\omega(j)$ where j is the smallest index for which $\sum_{i=0}^{j} t_i \geqslant t$. For a finite path $\omega = s_0 t_0 s_1 s_2 \ldots t_{k-1} s_k$, $time(\omega, i)$ is only defined for $i \leqslant k$: $time(\omega, j) = t_i$ for $i < k$ and $time(\omega, k) = \infty$. Furthermore, if $t \leqslant \sum_{i=1}^{k-1} t_i$, then $\omega@t$ is defined as for infinite paths, and otherwise $\omega@t = s_k$.

We denote by $Path^{\mathcal{C}}(s)$ the set of all (infinite and finite) paths of the CTMC $\mathcal{C}$ starting in state s. The probability measure Pr_s over $Path^{\mathcal{C}}(s)$, taken from [10], can be defined as follows. If the states $s_0, \ldots, s_n \in S$ satisfy $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $0 \leqslant i < n$ and $I_0, \ldots, I_{n-1}$ are non-empty intervals in $\mathbb{R}_{\geqslant 0}$, then the *cylinder set* $C(s_0, I_0, \ldots, I_{n-1}, s_n)$ is defined to be the set containing all paths $\omega \in Path^{\mathcal{C}}(s_0)$ such that $\omega(i) = s_i$ for all $i \leqslant n$ and $time(\omega, i) \in I_i$ for all $i < n$.

We then let $\Sigma_{Path^{\mathcal{C}}(s)}$ be the smallest σ-algebra on $Path^{\mathcal{C}}(s)$ which contains all the cylinder sets $C(s_0, I_0, \ldots, I_{n-1}, s_n)$, where $s_0, \ldots, s_n \in S$ range over all sequences of states with $s_0 = s$ and $\mathbf{R}(s_i, s_{i+1}) > 0$ for $0 \leqslant i < n$, and $I_0, \ldots, I_{n-1}$ range over all sequences of non-empty intervals in $\mathbb{R}_{\geqslant 0}$. Using Theorem 1, we

can then define the probability measure Pr_s on $\Sigma_{Path^{\mathcal{C}}(s)}$ as the unique measure such that $Pr_s(C(s)) = 1$ and for any cylinder $C(s, I, \ldots, I_{n-1}, s_n, I', s')$, $Pr_s(C(s, I, \ldots, I_{n-1}, s_n, I', s'))$ equals:

$$Pr_s(C(s, I, \ldots, I_{n-1}, s_n)) \cdot \mathbf{P}^{emb(\mathcal{C})}(s_n, s') \cdot \left(e^{-E(s_n)\cdot \inf I'} - e^{-E(s_n)\cdot \sup I'}\right) .$$

Example 13. Consider the CTMC $\mathcal{C}_1$ from Fig. 4 and the sequence of states and intervals $s_0, [0, 2], s_1$ (i.e. taking $I_0 = [0, 2]$ in the notation of the previous paragraph). Using the probability measure Pr_{s_0} over $(Path^{\mathcal{C}_1}(s_0), \Sigma_{Path^{\mathcal{C}}(s_0)})$, for the cylinder set $C(s_0, [0, 2], s_1)$, we have:

$$\begin{aligned} Pr_{s_0}(C(s_0, [0, 2], s_1)) &= Pr_{s_0}(C(s_0)) \cdot \mathbf{P}_1^{emb(\mathcal{C}_1)}(s_0, s_1) \cdot (e^{-E(s_0)\cdot 0} - e^{-E(s_0)\cdot 2}) \\ &= 1 \cdot 1 \cdot (e^0 - e^{-3}) \\ &= 1 - e^{-3} . \end{aligned}$$

Intuitively, this means that the probability of leaving the initial state s_0 and passing to state s_1 within the first 2 time units is $1 - e^{-3} \approx 0.950213$.

4.2 Steady-State and Transient Behaviour

In addition to path probabilities, we consider two more traditional properties of CTMCs: *transient* behaviour, which relates to the state of the model at a particular time instant; and *steady-state* behaviour, which describes the state of the CTMC in the long-run. For a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, the transient probability $\pi^{\mathcal{C}}_{s,t}(s')$ is defined as the probability, having started in state s, of being in state s' at time instant t. Using the definitions of the previous section:

$$\pi^{\mathcal{C}}_{s,t}(s') \stackrel{\text{def}}{=} Pr_s\{\omega \in Path^{\mathcal{C}}(s) \,|\, \omega@t = s'\} .$$

The steady-state probability $\pi^{\mathcal{C}}_s(s')$, i.e. the probability of, having started in state s, being in state s' in the long-run, is defined as:

$$\pi^{\mathcal{C}}_s(s') \stackrel{\text{def}}{=} \lim_{t\to\infty} \pi^{\mathcal{C}}_{s,t}(s') .$$

The steady-state probability distribution, i.e. the values $\pi^{\mathcal{C}}_s(s')$ for all $s' \in S$, can be used to infer the percentage of time, in the long-run, that the CTMC spends in each state. For the class of CTMCs which we consider here, i.e. those that are homogeneous and finite-state, the limit in the above definition always exists [59]. Furthermore, for CTMCs which are *irreducible* (strongly connected), that is, those for which there exists a finite path from each of its states to every other state, the steady-state probabilities $\pi^{\mathcal{C}}_s(s')$ are independent of the starting state s.

We now outline a standard technique, called uniformisation (also known as 'randomisation' or 'Jensen's method'), for computing transient probabilities of CTMCs as this will later be relied on in the model checking algorithms for CTMCs.

Uniformisation. For a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, we denote by $\boldsymbol{\Pi}_t^{\mathcal{C}}$ the matrix of all transient probabilities for time t, i.e. $\boldsymbol{\Pi}_t^{\mathcal{C}}(s, s') = \pi_{s,t}^{\mathcal{C}}(s')$. It can be shown (see for example [59]) that $\boldsymbol{\Pi}_t^{\mathcal{C}}$ can be expressed as a matrix exponential, and hence evaluated as a power series:

$$\boldsymbol{\Pi}_t^{\mathcal{C}} \;=\; e^{\mathbf{Q}\cdot t} \;=\; \sum_{i=0}^{\infty} \frac{(\mathbf{Q}\cdot t)^i}{i!}$$

where $\mathbf{Q}$ is the infinitesimal generator matrix of $\mathcal{C}$ (see Definition 11). Unfortunately, this computation tends to be unstable. As an alternative, the probabilities can be computed through the *uniformised* DTMC of $\mathcal{C}$.

Definition 12. *For any CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ with infinitesimal generator matrix $\mathbf{Q}$, the* uniformised *DTMC is given by $\mathit{unif}(\mathcal{C}) = (S, \bar{s}, \mathbf{P}^{\mathit{unif}(\mathcal{C})}, L)$ where $\mathbf{P}^{\mathit{unif}(\mathcal{C})} = \mathbf{I} + \mathbf{Q}/q$ and $q \geqslant \max\{E(s) \mid s \in S\}$.*

The *uniformisation rate* q is determined by the state with the shortest mean residence time. All (exponential) delays in the CTMC $\mathcal{C}$ are normalised with respect to q. That is, for each state $s \in S$ with $E(s) = q$, one epoch in $\mathit{unif}(\mathcal{C})$ corresponds to a single exponentially distributed delay with rate q, after which one of its successor states is selected probabilistically. As a result, no self-loop is added to such states in the DTMC $\mathit{unif}(\mathcal{C})$. If $E(s)<q$ – this state has on average a longer state residence time than $\frac{1}{q}$ – one epoch in $\mathit{unif}(\mathcal{C})$ might not be "long enough". Hence, in the next epoch these states might be revisited and, accordingly, are equipped with a self-loop with probability $1 - \frac{E(s)}{q}$. Note the difference between the embedded DTMC $\mathit{emb}(\mathcal{C})$ and the uniformised DTMC $\mathit{unif}(\mathcal{C})$: whereas the epochs in $\mathcal{C}$ and $\mathit{emb}(\mathcal{C})$ coincide and $\mathit{emb}(\mathcal{C})$ can be considered as the time less variant of $\mathcal{C}$, a single epoch in $\mathit{unif}(\mathcal{C})$ corresponds to a single exponentially distributed delay with rate q in $\mathcal{C}$. Now, using the uniformised DTMC the matrix of transient probabilities can be expressed as:

$$\boldsymbol{\Pi}_t^{\mathcal{C}} \;=\; \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{\mathit{unif}(\mathcal{C})}\right)^i \quad \text{where} \quad \gamma_{i,q\cdot t} = e^{-q\cdot t} \cdot \frac{(q\cdot t)^i}{i!}\,. \tag{2}$$

In fact, this reformulation has a fairly intuitive explanation. Each step of the uniformised DTMC corresponds to one exponentially distributed delay, with parameter q, in the CTMC. The matrix power $\left(\mathbf{P}^{\mathit{unif}(\mathcal{C})}\right)^i$ gives the probability of jumping between each pair of states in the DTMC in i steps and $\gamma_{i,q\cdot t}$ is the ith Poisson probability with parameter $q\cdot t$, the probability of i such steps occurring in time t, given the delay is exponentially distributed with rate q.

This approach has a number of important advantages. Firstly, unlike $\mathbf{Q}$, the matrix $\mathbf{P}^{\mathit{unif}(\mathcal{C})}$ is *stochastic*, meaning that all entries are in the range $[0, 1]$ and all rows sum to one. Computations using $\mathbf{P}^{\mathit{unif}(\mathcal{C})}$ are therefore more numerically stable. In particular, $\mathbf{Q}$ contains both positive and negative values which can cause severe round-off errors.

Secondly, the infinite sum is now easier to truncate. For example, the techniques of Fox and Glynn [27], which allow efficient computation of the Poisson

probabilities $\gamma_{i,q\cdot t}$, also produce an upper and lower bound $(L_\varepsilon, R_\varepsilon)$, for some desired precision ε, below and above which the probabilities are insignificant. Hence, the sum can be computed only over this range.

Lastly, the computation can be carried out efficiently using matrix-vector multiplications, rather than more costly matrix-matrix multiplications. Consider the problem of computing $\pi^{\mathcal{C}}_{s,t}(s')$ for a fixed state s. These values can be obtained by pre-multiplying the matrix $\boldsymbol{\Pi}^{\mathcal{C}}_t$ by the initial probability distribution, in this case the vector $\underline{\pi}^{\mathcal{C}}_{s,0}$ where $\pi^{\mathcal{C}}_{s,0}(s')$ is equal to 1 if $s' = s$ and 0 otherwise:

$$\underline{\pi}^{\mathcal{C}}_{s,t} = \underline{\pi}^{\mathcal{C}}_{s,0} \cdot \boldsymbol{\Pi}^{\mathcal{C}}_t = \underline{\pi}^{\mathcal{C}}_{s,0} \cdot \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i .$$

Rearranging, this can be expressed as a sum of vectors, rather than a sum of matrix powers:

$$\underline{\pi}^{\mathcal{C}}_{s,t} = \sum_{i=0}^{\infty} \left(\gamma_{i,q\cdot t} \cdot \underline{\pi}^{\mathcal{C}}_{s,0} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i \right)$$

where the vector required in each element of the summation is computed by a single matrix-vector multiplication, using the vector from the previous iteration:

$$\underline{\pi}^{\mathcal{C}}_{s,0} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i = \left(\underline{\pi}^{\mathcal{C}}_{s,0} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^{i-1} \right) \cdot \mathbf{P}^{unif(\mathcal{C})} .$$

Hence, the total work required is R_ε matrix-vector multiplications.

4.3 Continuous Stochastic Logic (CSL)

We write specifications of CTMCs using the logic CSL (Continuous Stochastic Logic), an extension of the temporal logic CTL.

Definition 13. *The syntax of CSL is as follows:*

$$\Phi ::= \mathtt{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{P}_{\sim p}[\phi] \mid \mathtt{S}_{\sim p}[\Phi]$$
$$\phi ::= \mathtt{X}\ \Phi \mid \Phi\ \mathtt{U}^I\ \Phi$$

where a is an atomic proposition, $\sim \in \{<, \leqslant, \geqslant, >\}$, $p \in [0,1]$ and I is an interval of $\mathbb{R}_{\geqslant 0}$.

As for PCTL, $\mathtt{P}_{\sim p}[\phi]$ indicates that the probability of the path formula ϕ being satisfied from a given state meets the bound $\sim p$. Path formulae are the same for CSL as for PCTL, except that the parameter of the 'until' operator is an interval I of the non-negative reals, rather than simply an integer upper bound. The path formula $\Phi\ \mathtt{U}^I\ \Psi$ holds if Ψ is satisfied at some time instant in the interval I and Φ holds at all preceding time instants. To avoid confusion, we will refer to this as the 'time-bounded until' operator. Similarly to PCTL, the standard 'unbounded until' operator can be derived by considering the interval $I = [0, \infty)$. The $\mathtt{S}$ operator describes the steady-state behaviour of the CTMC.

The formula $\mathtt{S}_{\sim p}[\Phi]$ asserts that the steady-state probability of being in a state satisfying Φ meets the bound $\sim p$.

As with PCTL, we write $s \models \Phi$ to indicate that a CSL formula Φ is satisfied in a state s and denote by $Sat(\Phi)$ the set $\{s \in S \mid s \models \Phi\}$. Similarly, for a path formula ϕ satisfied by path ω, we write $\omega \models \phi$. The semantics of CSL over CTMCs is defined as follows.

Definition 14. *Let $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ be a labelled CTMC. For any state $s \in S$ the relation $s \models \Phi$ is defined inductively by:*

$$\begin{array}{rcl} s \models \mathtt{true} & \textit{for all } s \in S & \\ s \models a & \Leftrightarrow & a \in L(s) \\ s \models \neg\Phi & \Leftrightarrow & s \not\models \Phi \\ s \models \Phi \wedge \Psi & \Leftrightarrow & s \models \Phi \wedge s \models \Psi \\ s \models \mathtt{P}_{\sim p}[\phi] & \Leftrightarrow & Prob^{\mathcal{C}}(s, \phi) \sim p \\ s \models \mathtt{S}_{\sim p}[\Phi] & \Leftrightarrow & \sum_{s' \models \Phi} \pi^{\mathcal{C}}_s(s') \sim p \end{array}$$

where:

$$Prob^{\mathcal{C}}(s, \phi) \stackrel{\text{def}}{=} Pr_s\{\omega \in Path^{\mathcal{C}}(s) \mid \omega \models \phi\}$$

and for any path $\omega \in Path^{\mathcal{C}}(s)$:

$$\begin{array}{l} \omega \models \mathtt{X}\ \Phi \quad \Leftrightarrow \omega(1) \textit{ is defined and } \omega(1) \models \Phi \\ \omega \models \Phi\ \mathtt{U}^I\ \Psi \Leftrightarrow \exists t \in I.\, (\omega@t \models \Psi \wedge \forall x \in [0, t).\, (\omega@x \models \Phi))\,. \end{array}$$

As discussed in [12], for any path formula Φ, the set $\{\omega \in Path^{\mathcal{C}}(s) \mid \omega \models \phi\}$ is a measurable set of $(Path^{\mathcal{C}}(s), \Sigma_{Path^{\mathcal{C}}(s)})$, and hence Pr_s is well defined over this set. In addition the steady-state probabilities $\pi^{\mathcal{C}}_s(s')$ always exists as $\mathcal{C}$ contains finitely many states [59].

As with PCTL, we can easily derive CSL operators for $\mathtt{false}$, $\vee$ and $\rightarrow$. Similarly, we can use the $\Diamond$ and $\Box$ temporal operators:

$$\begin{array}{l} \mathtt{P}_{\sim p}[\Diamond^I \Phi] \equiv \mathtt{P}_{\sim p}[\mathtt{true}\ \mathtt{U}^I\ \Phi] \\ \mathtt{P}_{\sim p}[\Box^I \Phi] \equiv \mathtt{P}_{\tilde{\sim} 1-p}[\Diamond^I \neg\Phi]\,. \end{array}$$

It is also worth noting that, despite the fact that CSL does not explicitly include operators to reason about transient probabilities, the following can be used to reason about the probability of satisfying a formula Φ at time instant t:

$$\mathtt{P}_{\sim p}[\Diamond^{[t,t]} \Phi]\,.$$

Example 14. Below are some typical examples of CSL formulae:

- $\mathtt{P}_{>0.9}[\Diamond^{[0,4.5]}\ \textit{served}]$ - the probability that a request is served within the first 4.5 seconds is greater than 0.9;
- $\mathtt{P}_{\leqslant 0.1}[\Diamond^{[10,\infty)}\ \textit{error}]$ - the probability that an error occurs after 10 seconds of operation is at most 0.1;

– $\mathit{down} \rightarrow \mathtt{P}_{>0.75}[\neg \mathit{fail}\ \mathtt{U}^{[1,2]}\ \mathit{up}\ ]$ - when a shutdown occurs, the probability of system recovery being completed in between 1 and 2 hours without further failures occurring is greater than 0.75;
– $\mathtt{S}_{<0.01}[\mathit{insufficient\_routers}]$ - in the long-run, the probability that an inadequate number of routers are operational is less than 0.01.

4.4 CSL Model Checking

In this section we consider a model checking algorithm for CSL over CTMCs. CSL model checking was shown to be decidable (for rational time bounds) in [4] and a model checking algorithm first presented in [12]. We use these techniques plus the subsequent improvements made in [10,40].

The inputs to the algorithm are a labelled CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ and a CSL formula Φ. The output is the set of states $\mathit{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$. As for DTMCs and PCTL, we first construct the parse tree of the formula Φ and, working upwards towards the root of the tree, we recursively compute the set of states satisfying each subformula. By the end, we have determined whether each state in the model satisfies Φ. The algorithm for CSL operators can be summarised as follows:

$$\begin{aligned}
\mathit{Sat}(\mathtt{true}) &= S \\
\mathit{Sat}(a) &= \{s \mid a \in L(s)\} \\
\mathit{Sat}(\neg \Phi) &= S \backslash \mathit{Sat}(\Phi) \\
\mathit{Sat}(\Phi \wedge \Psi) &= \mathit{Sat}(\Phi) \cap \mathit{Sat}(\Psi) \\
\mathit{Sat}(\mathtt{P}_{\sim p}[\phi]) &= \{s \in S \mid \mathit{Prob}^{\mathcal{C}}(s, \phi) \sim p\} \\
\mathit{Sat}(\mathtt{S}_{\sim p}[\Phi]) &= \{s \in S \mid \textstyle\sum_{s' \models \Phi} \pi^{\mathcal{C}}_s(s') \sim p\}\,.
\end{aligned}$$

Model checking for the majority of these operators is trivial to implement and is, in fact, the same as for the non-probabilistic logic CTL. The exceptions are the $\mathtt{P}_{\sim p}[\cdot]$ and $\mathtt{S}_{\sim p}[\cdot]$ operators which are considered below.

$\mathtt{P}_{\sim p}[\mathtt{X}\ \Phi]$ formulae. The CSL 'next' operator is defined as for the PCTL equivalent. Furthermore, the definition does not relate to any of the real-time aspects of CTMCs: it depends only on the probability of moving to the next immediate state, and hence this operator can actually be model checked by using the PCTL algorithms of Section 3 on the embedded DTMC $\mathit{emb}(\mathcal{C})$ (see Definition 10).

Example 15. Consider the CTMC $\mathcal{C}_1$ in Fig. 4 and the CSL formula $\mathtt{P}_{\geqslant 0.5}[\mathtt{X}\ \mathit{full}]$. Working with the embedded DTMC $\mathit{emb}(\mathcal{C}_1)$ and following the algorithm of Section 3, we multiply the matrix $\mathbf{P}^{\mathit{emb}(\mathcal{C}_1)}$, given in Example 12, by the vector $(0, 0, 0, 1)$, yielding the probabilities $\underline{\mathit{Prob}}^{\mathcal{C}_1}(\mathtt{X}\ \mathit{full}) = (0, 0, \frac{1}{3}, 0)$. Hence, the formula is not true in any state of the CTMC.

$\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^I\ \Psi]$ formulae. For this operator, we need to determine the probabilities $\mathit{Prob}^{\mathcal{C}}(s, \Phi\ \mathtt{U}^I\ \Psi)$ for all states s where I is an arbitrary interval of the non-negative real numbers. Noting that:

$$\mathit{Prob}^{\mathcal{C}}(s, \Phi\ \mathtt{U}^I\ \Psi) = \mathit{Prob}^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{cl(I)}\ \Psi)$$

where $cl(I)$ is the closure of the interval I, and that:

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,\infty)}\ \Psi) = Prob^{emb(\mathcal{C})}(s, \Phi\ \mathtt{U}^{\leqslant\infty}\ \Psi)$$

we are left with the following three cases for the interval I:

- $I = [0, t]$ for some $t \in \mathbb{R}_{\geqslant 0}$;
- $I = [t, t']$ for some $t, t' \in \mathbb{R}_{\geqslant 0}$ such that $t \leqslant t'$;
- $I = [t, \infty)$ for some $t \in \mathbb{R}_{\geqslant 0}$.

Following the method presented in [10], we will now show that the probabilities $Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{I}\ \Psi)$ for these cases can be computed using variants of uniformisation (see Section 4.2).

The case when $I = [0, t]$. Computing the probabilities in this case reduces to determining the least solution of the following set of integral equations: $Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t]}\ \Psi)$ equals 1 if $s \in Sat(\Psi)$, 0 if $s \in Sat(\neg\Phi \wedge \neg\Psi)$ and

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t]}\ \Psi) = \int_0^t \sum_{s' \in S} \mathbf{P}^{\mathrm{emb}(\mathcal{C})}(s, s') \cdot E(s) \cdot e^{-E(s)\cdot x} \cdot Prob^{\mathcal{C}}(s', \Phi\ \mathtt{U}^{[0,t-x]}\ \Psi)\, dx$$

otherwise. Here, $E(s) \cdot e^{-E(s)\cdot x}$ denotes the probability density of taking some outgoing transition from s at time x. Note the resemblance with equations (1) for the PCTL bounded until operator. Originally, [12] proposed to do this via approximate solution of Volterra integral equation systems. Experiments in [34] showed that this method was generally slow and, in [8], a simpler alternative was presented which reduces the problem to transient analysis. This approach is outlined below.

Definition 15. *For any CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *and CSL formula* Φ*, let CTMC* $\mathcal{C}[\Phi] = (S, \bar{s}, \mathbf{R}[\Phi], L)$ *with* $\mathbf{R}[\Phi](s, s') = \mathbf{R}(s, s')$ *if* $s \not\models \Phi$ *and 0 otherwise.*

Note that, using Definition 8, we have that $emb(\mathcal{C}[\Phi]) = emb(\mathcal{C})[\Phi]$.

Proposition 3 ([8]). *For a CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$*, CSL formulae* Φ *and* Ψ *and positive real* $t \in \mathbb{R}_{\geqslant 0}$*:*

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t]}\ \Psi) = \sum_{s' \models \Psi} \pi_{s,t}^{\mathcal{C}[\neg\Phi \vee \Psi]}(s')\,.$$

Consider the CTMC $\mathcal{C}[\neg\Phi \wedge \neg\Psi][\Psi] = \mathcal{C}[\neg\Phi \vee \Psi]$. Since a path in this CTMC cannot exit a state satisfying Ψ once it reaches one, and will never be able to reach a state satisfying Ψ if it enters one satisfying $\neg\Phi \wedge \neg\Psi$, the probability of the path formula $\Phi\ \mathtt{U}^{[0,t]}\ \Psi$ being satisfied in CTMC $\mathcal{C}$ is equivalent to the transient probability of being in a state satisfying Φ at time t in CTMC $\mathcal{C}[\neg\Phi \vee \Psi]$.

As shown in [40], uniformisation can be adapted to compute the vector of probabilities $\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[0,t]}\ \Psi)$ without having to resort to computing the probabilities for each state separately. More precisely, from Theorem 3:

$$\begin{aligned}\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[0,t]}\ \Psi) &= \boldsymbol{\Pi}_t^{\mathcal{C}[\neg\Phi\vee\Psi]}\cdot\underline{\Psi}\\ &= \left(\sum_{i=0}^{\infty}\gamma_{i,q\cdot t}\cdot\left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi\vee\Psi])}\right)^i\right)\cdot\underline{\Psi} \qquad \text{by (2)}\\ &= \sum_{i=0}^{\infty}\left(\gamma_{i,q\cdot t}\cdot\left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi\vee\Psi])}\right)^i\cdot\underline{\Psi}\right) \qquad \text{rearranging}\end{aligned}$$

Note that the inclusion of the vector $\underline{\Psi}$ within the brackets is vital since, like with uniformisation, it allows explicit computation of matrix powers to be avoided. Instead, each product is calculated as:

$$\left(\mathbf{P}^{unif(\mathcal{C})}\right)^0\cdot\underline{\Psi}=\boldsymbol{\Psi}\ \text{ and }\ \left(\mathbf{P}^{unif(\mathcal{C})}\right)^{i+1}\cdot\underline{\Psi}=\mathbf{P}^{unif(\mathcal{C})}\cdot\left(\left(\mathbf{P}^{unif(\mathcal{C})}\right)^i\cdot\underline{\Psi}\right),$$

reusing the computation from the previous iteration. As explained in Section 4.2, the infinite summation can be truncated using the techniques of Fox and Glynn [27]. In fact the summation can be truncated even sooner if the vector converges. As an additional optimisation, we can reuse the PROB0 algorithm, from Fig. 3 in Section 3, to initially identify all states s for which the probability $Prob^{\mathcal{C}}(s,\Phi\ \mathtt{U}^{[0,t]}\ \Psi)$ is 0.

Example 16. Consider the CTMC $\mathcal{C}_1$ in Fig. 4 and the CSL 'time-bounded until' formula $\mathtt{P}_{>0.65}[\mathtt{true}\ \mathtt{U}^{[0,7.5]}\ \mathit{full}]$. To compute the vector of probabilities $\underline{Prob}^{\mathcal{C}}(\mathtt{true}\ \mathtt{U}^{[0,7.5]}\ \mathit{full})$, i.e. the probability from each state that a state satisfying atomic proposition *full* is reached within 7.5 time units, we follow the procedure outlined above. First, observe that only state s_3 satisfies *full* and no states satisfy $\neg\mathtt{true}$. Hence, the only difference in the modified CTMC $\mathcal{C}_1[\neg\mathtt{true}\vee\mathit{full}]$ is that state s_3 made absorbing, i.e. the transition between states s_3 and s_2 is removed. Using the uniformisation rate $q = 4.5(=\max_{0\leqslant i\leqslant 3}E(s_i))$, the transition probability matrix for the uniformised DTMC of this modified CTMC $\mathcal{C}_1[\neg\mathtt{true}\vee\mathit{full}]$ is given by:

$$\mathbf{P}^{unif(\mathcal{C}_1[\neg\mathtt{true}\vee\mathit{full}])}=\begin{pmatrix}\frac{2}{3} & \frac{1}{3} & 0 & 0\\ \frac{2}{3} & 0 & \frac{1}{3} & 0\\ 0 & \frac{2}{3} & 0 & \frac{1}{3}\\ 0 & 0 & 0 & 1\end{pmatrix}.$$

Computing the summation of matrix-vector multiplications described above yields the solution:

$$\underline{Prob}^{\mathcal{C}_1}(\mathtt{true}\ \mathtt{U}^{[0,7.5]}\ \mathit{full})\approx(0.6482,0.6823,0.7811,1)$$

and we conclude that the CSL property is satisfied in states s_1, s_2 and s_3.

The case $I=[t,t']$. For this case, we split the computation into two parts. As shown in [8], we can consider separately the probabilities of: (a) staying in states satisfying Φ up until time t; (b) reaching a state satisfying Ψ, while remaining

in states satisfying Φ, within time $t' - t$. For the former, we use a similar idea to that used in the case when $I = [0, t]$, computing transient probabilities in a CTMC for which states satisfying $\neg\Phi$ have been made absorbing. We have:

$$\begin{aligned} Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[t,t']}\ \Psi) &= \sum_{s' \models \Phi} \pi^{\mathcal{C}[\neg\Phi]}_{s,t}(s') \cdot Prob^{\mathcal{C}}(s', \Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \\ &= \underline{\pi}^{\mathcal{C}[\neg\Phi]}_{s,t} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \end{aligned}$$

where $\underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi)$ is a vector with:

$$Prob^{\mathcal{C}}_{\Phi}(s, \Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) = \begin{cases} Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) & \text{if } s \models \Phi \\ 0 & \text{otherwise} \end{cases}$$

which can be computed using the method described above. The overall computation can be performed as a summation, in the style of uniformisation, to determine the probability for all states at once:

$$\begin{aligned} \underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[t,t']}\ \Psi) &= \boldsymbol{\Pi}^{\mathcal{C}[\neg\Phi]}_{t} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \\ &= \left(\sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])}\right)^{i}\right) \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \\ &= \sum_{i=0}^{\infty} \left(\gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])}\right)^{i} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi)\right) \end{aligned}$$

Again, this summation can be truncated and performed using only scalar and matrix-vector multiplication.

The case $I = [t, \infty)$. This case is, in fact, almost identical to the previous one. We again split the computation into two parts. Here, however, the second part is an unbounded, rather than time-bounded, 'until' formula, and hence the embedded DTMC can be used in this case. More precisely, we have:

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[t,\infty)}\ \Psi) = \underline{\pi}^{\mathcal{C}[\neg\Phi]}_{s,t} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}\ \Psi) = \underline{\pi}^{\mathcal{C}[\neg\Phi]}_{s,t} \cdot \underline{Prob}^{emb(\mathcal{C})}_{\Phi}(\Phi\ \mathtt{U}\ \Psi)\,.$$

Similarly to the above, this can be compute for all states:

$$\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[t,\infty)}\ \Psi) = \sum_{i=0}^{\infty} \left(\gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])}\right)^{i} \cdot \underline{Prob}^{emb(\mathcal{C})}_{\Phi}(\Phi\ \mathtt{U}\ \Psi)\right)\,.$$

$\mathtt{S}_{\sim p}[\Phi]$ **formulae.** A state s satisfies the formula $\mathtt{S}_{\sim p}[\Phi]$ if $\sum_{s' \models \Phi} \pi^{\mathcal{C}}_{s}(s') \sim p$. Therefore, to model check the formula $\mathtt{S}_{\sim p}[\Phi]$, we must compute the steady-state probabilities $\pi^{\mathcal{C}}_{s}(s')$ for all states s and s'. We first consider the simple case when $\mathcal{C}$ is irreducible.

The case when $\mathcal{C}$ is irreducible. As described in Section 4.2, the steady-state probabilities of $\mathcal{C}$ are independent of the starting state, and therefore we denote

by $\pi^{\mathcal{C}}(s)$ and $\underline{\pi}^{\mathcal{C}}$ the steady-state probability of being in the state s and the vector of all such probabilities, respectively. These probabilities can be computed as the unique solution of the linear equation system:

$$\underline{\pi}^{\mathcal{C}} \cdot \mathbf{Q} = \underline{0} \text{ and } \textstyle\sum_{s \in S} \pi^{\mathcal{C}}(s) = 1 \,. \tag{3}$$

This system can be solved by any standard approach, for example using direct methods, such as Gaussian elimination, or iterative methods, such as Jacobi and Gauss-Seidel. The satisfaction of the CSL formula, which in this case will be the same for all states, can be determined by summing the steady-state probabilities for all states satisfying Φ and comparing this result to the bound in the formula. More precisely, for any state $s \in S$:

$$s \models \mathtt{S}_{\sim p}[\Phi] \quad \Leftrightarrow \quad \textstyle\sum_{s' \models \Phi} \pi^{\mathcal{C}}(s') \sim p \,.$$

The case when $\mathcal{C}$ is reducible. In this case the procedure is more complex. First graph analysis is carried out to determine the set $bscc(\mathcal{C})$ of bottom strongly connected components (BSCCs) of $\mathcal{C}$, i.e. the set of strongly connect components of $\mathcal{C}$ that, once entered, cannot be left any more. Each individual BSCC $\mathcal{B} \in bscc(\mathcal{C})$ can be treated as an irreducible CTMC, and hence the steady-state probability distribution $\underline{\pi}^{\mathcal{B}}$ can be determined using the method described in the previous case.

Next, we calculate the probability of reaching each BSCC $\mathcal{B} \in bscc(\mathcal{C})$ from each state s of $\mathcal{C}$. In fact, this is simply $Prob^{emb(\mathcal{C})}(s, \Diamond\, a_{\mathcal{B}})$, where $a_{\mathcal{B}}$ is an atomic proposition true only in the states $s' \in \mathcal{B}$. Then, for states $s, s' \in S$, the steady-state probability $\pi^{\mathcal{C}}_s(s')$ can be computed as:

$$\pi^{\mathcal{C}}_s(s') = \begin{cases} Prob^{emb(\mathcal{C})}(s, \Diamond\, a_{\mathcal{B}}) \cdot \pi^{\mathcal{B}}(s') & \text{if } s' \in \mathcal{B} \text{ for some } \mathcal{B} \in bscc(\mathcal{C}) \\ 0 & \text{otherwise.} \end{cases}$$

Note that, since the steady-state probabilities $\pi^{\mathcal{B}}(s')$ are independent of s, the total work required to compute $\pi^{\mathcal{C}}_s(s')$ for all $s, s' \in S$ is the solution of two linear equation systems for each BSCC in the CTMC: one to obtain the vector of probabilities $\underline{\pi}^{\mathcal{B}}$ and another for the vector of probabilities $\underline{Prob}^{emb(\mathcal{C})}(\Diamond\, a_{\mathcal{B}})$. Computation of the BSCCs in the CTMC requires an analysis of its underlying graph structure and can be performed using classic algorithms based on depth-first search [60].

Example 17. Consider the CTMC $\mathcal{C}_1$ in Fig. 4 and the CSL 'steady-state' formula $\mathtt{S}_{<0.1}[\mathit{full}]$. From inspection, we see that the CTMC comprises a single BSCC containing all 4 states. Hence, the steady-state probabilities are computed by solving the linear equation system:

$$\begin{aligned}
-\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_0) + 3 \cdot \pi^{\mathcal{C}_1}(s_1) &= 0 \\
\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_0) - \tfrac{9}{2} \cdot \pi^{\mathcal{C}_1}(s_1) + 3 \cdot \pi^{\mathcal{C}_1}(s_2) &= 0 \\
\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_1) - \tfrac{9}{2} \cdot \pi^{\mathcal{C}_1}(s_2) + 3 \cdot \pi^{\mathcal{C}_1}(s_3) &= 0 \\
\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_2) - 3 \cdot \pi^{\mathcal{C}_1}(s_3) &= 0 \\
\pi^{\mathcal{C}_1}(s_0) + \pi^{\mathcal{C}_1}(s_1) + \pi^{\mathcal{C}_1}(s_2) + \pi^{\mathcal{C}_1}(s_3) &= 1
\end{aligned}$$

which has the solution $\underline{\pi}^{\mathcal{C}_1} = (\frac{8}{15}, \frac{4}{15}, \frac{2}{15}, \frac{1}{15})$. State s_3 is the only state satisfying atomic proposition *full*, and thus the CSL formula is true in all states.

4.5 Extending CTMCs and CSL with Rewards

As for DTMCs, given a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, we can enrich $\mathcal{C}$ with a reward structure $(\underline{\rho}, \boldsymbol{\iota})$. Recall that the state reward function $\underline{\rho} : S \to \mathbb{R}_{\geqslant 0}$ defines this as the rate at which reward is acquired in a state and the transition reward function $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geqslant 0}$ defines the reward acquired each time a transition occurs. Note that, since we are now in the continuous time setting, a reward of $t \cdot \underline{\rho}(s)$ will be acquired if the CTMC remains in state s for $t \in \mathbb{R}_{\geqslant 0}$ time units.

Example 18. Returning to the CTMC $\mathcal{C}_1$ of Example 12 (see Fig. 4), we consider two different reward structures:

– $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ where $\boldsymbol{\iota}^{\mathcal{C}_1}$ assigns 1 to the transitions corresponding to a request being served and 0 to all other transitions, that is:

$$\boldsymbol{\iota}^{\mathcal{C}_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

  Such a structure can be used for measures relating to the number of requests served within a given time interval or in the long-run.

– $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ where $\underline{\rho}^{\mathcal{C}_1}$ associates with each state the number of requests that are awaiting service:

$$\underline{\rho}^{\mathcal{C}_1} = (0, 1, 2, 3)\,.$$

  This structure is used when one is interested in the queue size at any time instant or in the long-run.

The construction of both the embedded DTMC $emb(\mathcal{C})$ (see Definition 10) and uniformised DTMC $unif(\mathcal{C})$ (see Definition 12) can be extended to incorporate the reward structure $(\boldsymbol{\iota}, \underline{\rho})$. In both constructions the transition reward function does not change, that is, $\boldsymbol{\iota}^{emb(\mathcal{C})} = \boldsymbol{\iota}^{unif(\mathcal{C})} = \boldsymbol{\iota}$. On the other hand, the constructed state reward function takes into account the expected time that the CTMC remains in each state. More precisely, if q is the uniformisation rate used in the construction of the uniformised DTMC, then for any $s \in S$:

$$\underline{\rho}^{emb(\mathcal{C})}(s) = E(s) \cdot \underline{\rho}(s) \;\text{ and }\; \underline{\rho}^{unif(\mathcal{C})}(s) = \tfrac{1}{q} \cdot \underline{\rho}(s)\,.$$

We extend the syntax of logic CSL to allow for specifications relating to rewards by introducing the following formulae:

$$\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{S}]$$

where $\sim \in \{<, \leqslant, \geqslant, >\}$, $r, t \in \mathbb{R}_{\geqslant 0}$ and Φ is a CSL formula. Intuitively, a state s satisfies $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}]$ if, from state s, the expected reward *cumulated* up until t time

units have elapsed satisfies $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{I}^{=t}]$ is true if, from state s, the expected state reward at time instant t meets the bound $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$ is true if, from state s, the expected reward cumulated before a state satisfying Φ is reached meets the bound $\sim r$; and $\mathtt{R}_{\sim r}[\mathtt{S}]$ is true if, from state s, the long-run average expected reward satisfies $\sim r$.

Formally, given a CTMC $\mathcal{D} = (S, \bar{s}, \mathbf{R}, L)$, the semantics of these formulae is defined as follows. For any $s \in S$, $r, t \in \mathbb{R}_{\geqslant 0}$ and PCTL formula Φ:

$$\begin{array}{rcl} s \models \mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}] & \Leftrightarrow & \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{C}^{\leqslant t}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] & \Leftrightarrow & \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{I}^{=t}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi] & \Leftrightarrow & \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{F}\Phi}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{S}] & \Leftrightarrow & \lim_{t\to\infty} \frac{1}{t} \cdot \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{C}^{\leqslant t}}) \sim r \end{array}$$

where $\mathit{Exp}^{\mathcal{C}}(s, X)$ denotes the expectation of the random variable X with respect to the probability measure Pr_s and for any path $\omega = s_0 t_0 s_1 t_1 s_2 \cdots \in \mathit{Path}^{\mathcal{C}}(s)$:

$$X_{\mathtt{C}^{\leqslant t}}(\omega) \stackrel{\text{def}}{=} \sum_{i=0}^{j_t-1} \left(t_i \cdot \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1})\right) + \left(t - \sum_{i=0}^{j_t-1} t_i\right) \cdot \underline{\rho}(s_{j_t})$$

$$X_{\mathtt{I}^{=t}}(\omega) \stackrel{\text{def}}{=} \underline{\rho}(\omega@t)$$

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

and $j_t = \min\{j \mid \sum_{i=0}^{j} t_i \geqslant t\}$.

Example 19. Below are some typical examples of reward based formulae:

- $\mathtt{R}_{>3.6}[\mathtt{C}^{\leqslant 4.5}]$ - the expected number of requests served within the first 4.5 seconds of operations is greater than 3.6;
- $\mathtt{R}_{<2}[\mathtt{I}^{=6.7}]$ - the expected size of the queue when 6.7 time units have elapsed is less than 2;
- $\mathtt{R}_{<10}[\mathtt{F}\ \mathit{full}]$ - the expected number of requests served before the queue becomes full is less than 10;
- $\mathtt{R}_{\geqslant 1.2}[\mathtt{S}]$ - the expected long-run queue size is at least 1.2.

We now consider the computation of the expected values of the different random variables defined above.

The random variable $X_{\mathtt{C}^{\leqslant t}}$. Based on the results in [43,44], we can use the uniformised DTMC to compute the expectation of this random variable. More precisely, we have have the following result.

Proposition 4 ([44]). *For a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, state $s \in S$ and positive real $t \in \mathbb{R}_{\geqslant 0}$:*

$$\mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{C}^{\leqslant t}}) \;=\; \sum_{i=0}^{\infty} \bar{\gamma}_{i,q\cdot t} \cdot \left(\mathbf{P}^{\mathit{unif}(\mathcal{C})}\right)^i \cdot \underline{f_q}(\underline{\rho}, \boldsymbol{\iota})$$

where

$$\bar{\gamma}_{i,q\cdot t} \stackrel{\text{def}}{=} \int_0^t \gamma_{i,q\cdot u}\, du \;=\; \frac{1}{q}\sum_{j=i+1}^{\infty} \gamma_{j,q\cdot t} \;=\; \frac{1}{q}\left(1 - \sum_{j=i}^{i} \gamma_{j,q\cdot t}\right),$$

$\underline{\pi}_{s,i}^{unif(\mathcal{C})}$ *denotes the probability distribution in* $unif(\mathcal{C})$ *after* i *steps when starting in* s*,* q *is the uniformisation rate and*

$$\underline{f_q}(\underline{\rho}, \boldsymbol{\iota}) = \underline{\rho} + q \cdot \left(\mathbf{P}^{unif(\mathcal{C})} \bullet \boldsymbol{\iota}\right) \cdot \underline{1}$$

with $\bullet$ *denoting the Schur or entry-wise multiplication of matrices and* $\underline{1}$ *a vector with all entries equal to 1.*

Similarly to computing the vector of probabilities $\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[0,t]}\ \Psi)$, we can both truncate the summation and use only scalar and matrix-vector multiplication in the computation. In this case, to compute the coefficients $\bar{\gamma}_{i,q\cdot t}$, we can employ the method (based on Fox and Glynn [27]) given in [42].

Example 20. Returning to the CTMC $\mathcal{C}_1$ of Example 12 and the reward structure $(\mathbf{0}, \underline{\rho}^{\mathcal{C}_1})$ of Example 18, the expected number of requests served after 5.5 time units have elapsed is given by:

$$\underline{Exp}^{\mathcal{C}_1}(X_{\mathtt{C}^{\leqslant 5.5}}) \approx (7.0690, 8.0022, 8.8020, 9.3350)$$

and hence only state s_3 satisfies $\mathtt{R}_{>9}[\mathtt{C}^{\leqslant 5.5}]$.

The random variable $X_{\mathtt{I}^{=t}}$. In this case, using the fact that:

$$Exp^{\mathcal{C}}(s, X_{\mathtt{I}^{=t}}) = \sum_{s' \in S} \underline{\rho}(s') \cdot \underline{\pi}_{s,t}^{\mathcal{C}}(s')$$

we can again use the uniformised DTMC $unif(\mathcal{C})$ to compute the expectation. More precisely, we can compute the vector $\underline{Exp}^{\mathcal{C}}(X_{\mathtt{I}^{=t}})$ through the following sum over vectors of coefficients:

$$\underline{Exp}^{\mathcal{C}}(X_{\mathtt{I}^{=t}}) = \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i \cdot \underline{\rho}$$

which again can be truncated and computed using only scalar and matrix-vector multiplications.

Example 21. Returning to the CTMC $\mathcal{C}_1$ in Example 12 and the reward structure $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ of Example 18, the expected size of the queue after 1 time unit has elapsed is given by:

$$\underline{Exp}^{\mathcal{C}_1}(X_{\mathtt{I}^{=1}}) \approx (0.5929, 0.7352, 1.0140, 1.2875)$$

and hence all states satisfy the formula $\mathtt{R}_{<2}[\mathtt{I}^{=1}]$.

The random variable $X_{\mathtt{F}\Phi}$. To compute the expectations in this case, we use the fact that:

$$Exp^{\mathcal{C}}(s, X_{\mathtt{F}\Phi}) = Exp^{emb(\mathcal{C})}(s, X_{\mathtt{F}\Phi})$$

that is, we compute the expectations by constructing the embedded DTMC $emb(\mathcal{C})$ and employing the algorithms for verifying DTMCs against PCTL given in Section 3.3.

Example 22. Consider the CTMC $\mathcal{C}_1$ of Example 12, the reward structure $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ of Example 18. The formula $\mathtt{R}_{<7}[\mathtt{F}\ \mathit{full}]$, in this case, states that the expected number of requests served before the queue becomes full is less than 7. Now, computing the expectations $Exp^{emb(\mathcal{C}_1)}(s, X_{\mathtt{F}\mathit{full}})$ according to Section 3.4:

$$\begin{aligned} \mathit{Sat}(\mathit{full}) &= \{s_3\} \\ \mathit{Sat}(\mathtt{P}_{<1}[\Diamond\ \mathit{full}]) &= S \setminus \textsc{Prob1}(S, \mathit{Sat}(\mathit{full}), \textsc{Prob0}(S, \mathit{Sat}(\mathit{full}))) \\ &= S \setminus \{s_0, s_1, s_2, s_3\} = \varnothing \end{aligned}$$

leading to the linear equation system:

$$\begin{aligned} Exp^{emb(\mathcal{C}_1)}(s_0, X_{\mathtt{F}\mathit{full}}) &= 1 \cdot Exp^{emb(\mathcal{C}_1)}(s_1, X_{\mathtt{F}\mathit{full}}) \\ Exp^{emb(\mathcal{C}_1)}(s_1, X_{\mathtt{F}\mathit{full}}) &= \tfrac{2}{3} \cdot \big(1 + Exp^{emb(\mathcal{C}_1)}(s_0, X_{\mathtt{F}\mathit{full}})\big) + \tfrac{1}{3} Exp^{emb(\mathcal{C}_1)}(s_2, X_{\mathtt{F}\mathit{full}}) \\ Exp^{emb(\mathcal{C}_1)}(s_2, X_{\mathtt{F}\mathit{full}}) &= \tfrac{2}{3} \cdot \big(1 + Exp^{emb(\mathcal{C}_1)}(s_1, X_{\mathtt{F}\mathit{full}})\big) \\ Exp^{emb(\mathcal{C}_1)}(s_3, X_{\mathtt{F}\mathit{full}}) &= 0\,. \end{aligned}$$

Solving this system of equations gives $\underline{Exp^{emb(\mathcal{C}_1)}}(X_{\mathtt{F}\mathit{full}}) = (8, 8, 6, 0)$, and therefore, since $Exp^{\mathcal{C}_1}(s, X_{\mathtt{F}\mathit{full}}) = Exp^{emb(\mathcal{C}_1)}(s, X_{\mathtt{F}\mathit{full}})$, only states s_2 and s_3 satisfy the formula $\mathtt{R}_{\leqslant 7}[\mathtt{F}\ \mathit{full}]$.

The random variable $X_{\mathtt{S}}$. As in the case of the operator $\mathtt{S}_{\sim p}[\cdot]$, we consider the cases when $\mathcal{C}$ is irreducible and reducible separately.

The case when $\mathcal{C}$ is irreducible. If $\underline{\pi}^{\mathcal{C}}$ is the vector of the steady-state probabilities (recall that when $\mathcal{C}$ is irreducible the steady-state probabilities are independent of the starting state), we have:

$$Exp^{\mathcal{C}}(s, X_{\mathtt{S}}) = \underline{\pi}^{\mathcal{C}} \cdot \underline{\rho} + \underline{\pi}^{\mathcal{C}} \cdot (\mathbf{R} \bullet \boldsymbol{\iota}) \cdot \underline{1}$$

with $\bullet$ again denoting the Schur or entry-wise multiplication of matrices and $\underline{1}$ a vector with all entries equal to 1. Note that since the expectation is independent of the starting state, we denote the expectation by $Exp^{\mathcal{C}}(X_{\mathtt{S}})$. The computation in this case therefore requires the computation of the steady-state probabilities of $\mathcal{C}$, which reduces to solving the linear equation system given in (3).

The case when $\mathcal{C}$ is reducible. Similarly, to the approach for checking formulae of the form $\mathtt{S}_{\sim p}[\Phi]$, first, through graph analysis, we determine the set $bscc(\mathcal{C})$ of BSCCs of $\mathcal{C}$. Next, treating each individual $\mathcal{B} \in bscc(\mathcal{C})$ as an irreducible

CTMC, we compute the expectations $Exp^{\mathcal{B}_i}(X_{\mathtt{S}})$ and determine the vector of probabilities $\underline{Prob}^{emb(\mathcal{C})}(\Diamond\, a_{\mathcal{B}})$ for each $\mathcal{B} \in bscc(\mathcal{C})$. Finally, for each state $s \in S$:

$$Exp^{\mathcal{C}}(s, \mathtt{S}) = \sum_{\mathcal{B} \in bscc(\mathcal{C})} Prob^{\mathrm{emb}(\mathcal{C})}(s, \Diamond\, a_{\mathcal{B}}) \cdot Exp^{\mathcal{B}}(\mathtt{S}) \; .$$

Example 23. Returning once again to the CTMC $\mathcal{C}_1$ in Example 12, using the steady-state probabilities computed earlier and the reward structure $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ of Example 18, the long-run average expected number of requests served is given by:

$$\begin{aligned}
\underline{\pi}^{\mathcal{C}} \cdot \underline{0} + \underline{\pi}^{\mathcal{C}} \cdot \left(\mathbf{R}_1 \bullet \boldsymbol{\iota}^{\mathcal{C}_1}\right) \cdot \underline{1} &= \left(\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}\right) \cdot \left(\begin{pmatrix} 0 & \frac{3}{2} & 0 & 0 \\ 3 & 0 & \frac{3}{2} & 0 \\ 0 & 3 & 0 & \frac{3}{2} \\ 0 & 0 & 3 & 0 \end{pmatrix} \bullet \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \right) \cdot \underline{1} \\
&= \left(\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}\right) \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \end{pmatrix} \cdot \underline{1} \\
&= \left(\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}\right) \cdot \begin{pmatrix} 0 \\ 3 \\ 3 \\ 3 \end{pmatrix} = \tfrac{7}{5}
\end{aligned}$$

and thus no states satisfy the formula $\mathtt{R}_{\geqslant 1.5}[\mathtt{S}]$ when the reward structure $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ is associated with the CTMC $\mathcal{C}_1$.

On the other hand, using the reward structure $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ of Example 18, the long-run average size of the queue is given by:

$$\underline{\pi}^{\mathcal{C}} \cdot \underline{c}_1 + \underline{\pi}^{\mathcal{C}} \cdot \left(\mathbf{R}_1 \bullet \mathbf{0}\right) \cdot \underline{1} = \left(\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}\right) \cdot \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} = \tfrac{11}{15}$$

and hence all states satisfy the formula $\mathtt{R}_{\leqslant 0.8}[\mathtt{S}]$ when the reward structure $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ is associated with the CTMC $\mathcal{C}_1$.

4.6 Complexity of CSL Model Checking

The overall time complexity for model checking a CSL formula Φ against a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ is linear in $|\Phi|$, polynomial in $|S|$ and linear in $q \cdot t_{\max}$, where $q = \max_{s \in S} |\mathbf{Q}(s, s)|$ and $t_{\max}$ is the maximum value found in the parameter of a 'time-bounded until' operator. For formulae of the form $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{[0,\infty)}\ \Psi]$, $\mathtt{S}_{\sim p}[\Phi]$, $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$ and $\mathtt{R}_{\sim r}[\mathtt{S}]$ a solution of a linear equation system of size $|S|$ is required. This can be done with Gaussian elimination, the complexity of which

is cubic in the size of the system. For formula of the form $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^I\ \Psi]$, $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}]$ and $\mathtt{R}_{\sim r}[\mathtt{I}^{=t}]$ we must perform at most two iterative summations, each step of which requires a matrix-vector multiplication. This operation is quadratic in the size of the matrix, i.e. $|S|$. The total number of iterations required is determined by the upper bound supplied by the algorithm of Fox and Glynn [27], which for large $q{\cdot}t$ is linear in $q{\cdot}t$.

5 Stochastic Model Checking in Practice

In this section we first give a high-level overview of the functionality of the stochastic model checker PRISM and then discuss three case studies employing stochastic model checking and PRISM.

5.1 The Probabilistic Model Checker PRISM

PRISM [36,53] is a probabilistic model checker developed at the University of Birmingham. It accepts probabilistic models described in its *modelling language*, a simple, high-level state-based language. Three types of probabilistic models are supported directly; these are discrete-time Markov chains (DTMCs), Markov decision processes (MDPs), and continuous-time Markov chains (CTMCs). Markov decision processes, not considered in this tutorial, extend DTMCs by allowing non-deterministic behaviour that is needed, for example, to model asynchronous parallel composition. For a detailed introduction to model checking of MDPs see, for example, [56]. Additionally, probabilistic timed automata (PTAs) are partially supported, with the subset of diagonal-free PTAs supported directly via *digital clocks* [47]. Properties are specified using PCTL for DTMCs and MDPs, and CSL for CTMCs. Probabilistic timed automata have a logic PTCTL, an extension of TCTL, a subset of which is supported via a connection to the timed automata model checking tool Kronos [24].

Tool Overview. PRISM first parses the model description and constructs an internal representation of the probabilistic model, computing the reachable state space of the model and discarding any unreachable states. This represents the set of all feasible configurations which can arise in the modelled system. Next, the specification is parsed and appropriate model checking algorithms are performed on the model by induction over syntax. In some cases, such as for properties which include a probability bound, PRISM will simply report a true/false outcome, indicating whether or not each property is satisfied by the current model. More often, however, properties return *quantitative* results and PRISM reports, for example, the actual probability of a certain event occurring in the model. Furthermore, PRISM supports the notion of *experiments*, which is a way of automating multiple instances of model checking. This allows the user to easily obtain the outcome of one or more properties as functions of model and property parameters. The resulting table of values can either be viewed directly,

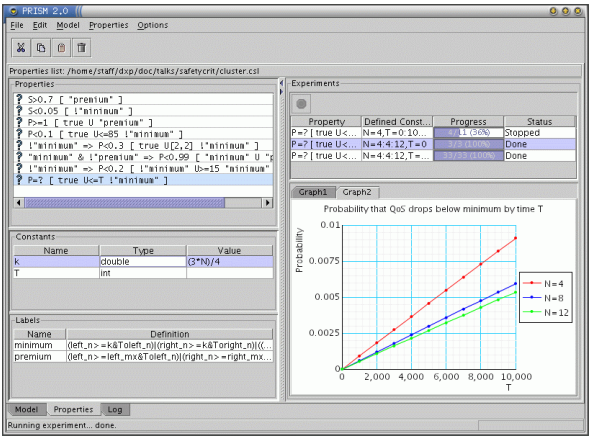


Fig. 5. A screenshot of the PRISM graphical user interface

exported for use in an external application such as a spreadsheet, or plotted as a graph. For the latter, PRISM incorporates substantial graph-plotting functionality. This is often a very useful way of identifying interesting patterns or trends in the behaviour of a system. The reader is invited to consult the 'Case Studies' section of the PRISM website [53] for many examples of this kind of analysis.

Fig. 5 shows a screenshot of the PRISM graphical user interface, illustrating the results of a model checking experiment being plotted on a graph. The tool also features a built-in text-editor for the PRISM language. Alternatively, all model checking functionality is also available in a command-line version of the tool. PRISM is a *free*, *open source* application. It presently operates on Linux, Unix, Windows and Macintosh operating systems. Both binary and source code versions can be downloaded from the website [53].

Implementation. One of the most notable features of PRISM is that it is a *symbolic* model checker, meaning that its implementation uses data structures based on binary decision diagrams (BDDs). These provide compact representations and efficient manipulation of large, structured probabilistic models by exploiting regularity that is often present in those models because they are described in a structured, high-level modelling language. More specifically, since we need to store numerical values, PRISM uses *multi-terminal* binary decision diagrams (MTBDDs) [21,7] and a number of variants [46,52,48] developed to improve the efficiency of probabilistic analysis, which involve combinations of *symbolic* data structures such as MTBDDs and conventional *explicit* storage schemes such as sparse matrices and arrays. Since its release in 2001, the model size capacity and tool efficiency has increased substantially (10^7 – 10^8 is feasible for CTMCs and higher for other types of models). PRISM employs and builds upon the Colorado University Decision Diagram package [58] by Fabio Somenzi which implements BDD/MTBDD operations.

The underlying computation in PRISM involves a combination of:

- *graph-theoretical algorithms*, for reachability analysis, conventional temporal logic model checking and *qualitative* probabilistic model checking;
- *numerical computation*, for *quantitative* probabilistic model checking, e.g. solution of linear equation systems (for DTMCs and CTMCs) and linear optimisation problems for (MDPs).

Graph-theoretical algorithms are comparable to the operation of a conventional, non-probabilistic model checker and are always performed in PRISM using BDDs. For numerical computation, PRISM uses iterative methods rather than direct methods due to the size of the models that need to be handled. For solution of linear equation systems, it supports a range of well-known techniques, including the Jacobi, Gauss-Seidel and SOR (successive over-relaxation) methods. For the linear optimisation problems which arise in the analysis of MDPs, PRISM uses dynamic programming techniques, in particular, value iteration. Finally, for transient analysis of CTMCs, PRISM incorporates another iterative numerical method, uniformisation (see Section 4.4).

In fact, for numerical computation, the tool actually provides three distinct numerical *engines*. The first is implemented purely in MTBDDs (and BDDs); the second uses sparse matrices; and the third is a hybrid, using a combination of the two. Performance (time and space) of the tool may vary depending on the choice of the engine. Typically the sparse engine is quicker than its MTBDD counterpart, but requires more memory. The hybrid engine aims to provide a compromise, providing faster computation than pure MTBDDs but using less memory than sparse matrices (see [46,52]).

The PRISM modelling language. The PRISM modelling language is a simple, state-based language based on the Reactive Modules formalism of Alur and Henzinger [1]. In this section, we give a brief outline of the language. For a full definition of the language and its semantics, see [45]. In addition a wide range of examples can be found both in the 'Case Studies' section of the PRISM website [53] and in the distribution of the tool itself.

The fundamental components of the PRISM language are *modules* and *variables*. Variables are typed (integers, reals and booleans are supported) and can be local or global. A model is composed of *modules* which can interact with each other. A module contains a number of local *variables*. The values of these variables at any given time constitute the state of the module. The *global state* of the whole model is determined by the *local state* of all modules, together with the values of the global variables. The behaviour of each module is described by a set of *commands*. A command takes the form:

$$[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n\,;$$

The *guard* g is a predicate over all the variables in the model (including those belonging to other modules). Each *update* u_i describes a transition which the module can make if the guard is true. A transition is specified by giving the new

```
dtmc

module D1

    x : [0..3] init 0;

    [] x=0 → (x'=1);
    [] x=1 → 0.01 : (x'=1)
           + 0.01 : (x'=2)
           + 0.98 : (x'=3);
    [] x=2 → (x'=0);
    [] x=3 → (x'=3);

endmodule
```

```
ctmc

module C1

    y : [0..3] init 0;

    [] y<3 → 1.5 : (y'=y+1);
    [serve] y>0 → 3 : (y'=y−1);

endmodule
```

Fig. 6. The PRISM Language: Specification of $\mathcal{D}_1$ and $\mathcal{C}_1$

values of the variables in the module, possibly as an *expression* formed from other variables or constants. The expressions λ_i are used to assign probabilistic information to the transitions.

In Fig. 6 we present the specification of the DTMC $\mathcal{D}_1$ (see Example 1 and Fig. 1) and the CTMC $\mathcal{C}_1$ (see Example 12 and Fig. 4). For both these models there is a single initial state, but PRISM allows the specification of a set of initial states, see [45]. The labelling '*serve*' of the second command in the specification of $\mathcal{C}_1$ will be used below to specify a reward structure for this model.

In general the probabilistic model corresponding to a PRISM language description is constructed as the parallel composition of its modules. In every state of the model, there is a set of commands (belonging to any of the modules) which are enabled, i.e. whose guards are satisfied in that state. The choice between which command is performed (i.e. the scheduling) depends on the model type. For a DTMC, the choice is *probabilistic*, with each enabled command selected with equal probability and for CTMCs it is modelled as a *race condition*. PRISM also supports multi-way *synchronisation* in the style of process algebras. For synchronisation to take effect, commands are labelled with *actions* that are placed between the square brackets.

Reward Structures. PRISM includes support for the specification and analysis of properties based on *reward* (and cost) structures. Reward structures are associated with models using the `rewards` "*reward_name*" ... `endrewards` construct and are specified using multiple reward items of the form:

$$g \ : \ r; \quad \text{or} \quad [a] \ g \ : \ r;$$

depending on whether a state or transition rewards are being specified, where g is a predicate (over all the variables of the model), a is a action label appearing in the commands of the model and r is a real-valued expression (containing any variables, constants, etc. from the model). A single reward item can assign

different rewards to different states or transitions, depending on the values of model variables in each one. Any states/transitions which do not satisfy the guard of a reward item will have no reward assigned to them. For states/transitions which satisfy multiple guards, the reward assigned is the sum of the rewards for all the corresponding reward items.

For example, the two reward structures of the CTMC $\mathcal{C}_1$ given in Example 18 can be specified as:

```
rewards "reward1"
    true : y;
endrewards
```

```
rewards "reward2"
    [serve] true : 1;
endrewards
```

To further illustrate how reward structures are specified in PRISM consider the reward structure given below, which assigns a state reward of 100 to states satisfying x=1 or y=1 and 200 to states that satisfy both x=1 and y=1, and a transition reward of $2 \cdot x$ to transitions labelled by a from states satisfying x>0 and x<5.

```
rewards "reward_name"
    x=1 : 100;
    y=1 : 100;
    [a] x>0 & x<5 : 2 * x;
endrewards
```

Property specifications. Properties of PRISM models are expressed in PCTL for DTMCs and CSL for CTMCs. The operators $\mathtt{P}_{\sim p}[\cdot]$, $\mathtt{S}_{\sim p}[\cdot]$ and $\mathtt{R}_{\sim r}[\cdot]$ by default include the probability bound $\sim p$ or reward bound $\sim r$. However, in PRISM, we can also directly specify properties which evaluate to a *numerical value* by replacing the bounds in the P, S and R operators with =?, as illustrated in the following PRISM specifications:

- P=? [! *proc2_terminate* U *proc1_terminate*] - the probability that process 1 terminates before process 2 completes;
- S=? [(*queue_size*/*max_size*)>0.75] - the long-run probability that the queue is more than 75% full;
- R=? [C ⩽ 24] - the expected power consumption during the first 24 hours of operation;
- R=? [I = 100] - after 100 time units, the expected number of packets awaiting delivery;
- R=? [F *elected*] - the expected number of steps required for the leader election algorithm to complete;
- R=? [S] - the long-run expected queue-size.

Note that the meaning ascribed to these properties is, of course, dependent on the definitions of the atomic propositions and reward structures.

By default, the result for properties of this kind is the probability for the initial state of the model. It is also possible, however, to obtain the probability

for an arbitrary state or more generally either the minimum or maximum probability for a particular class of states, as demonstrated in the following PRISM specifications:

- P=? [*queue_size*⩽5 U *queue_size*<5 {*queue_size*=5}] - the probability, from the state where the queue contains 5 jobs, of the queue processing at least one job before another arrives;
- P=? [!*proc2_terminate* U *proc1_terminate* {*init*}{min}] - the minimum probability, over all possible initial configurations, that process 1 terminates before process 2 does.

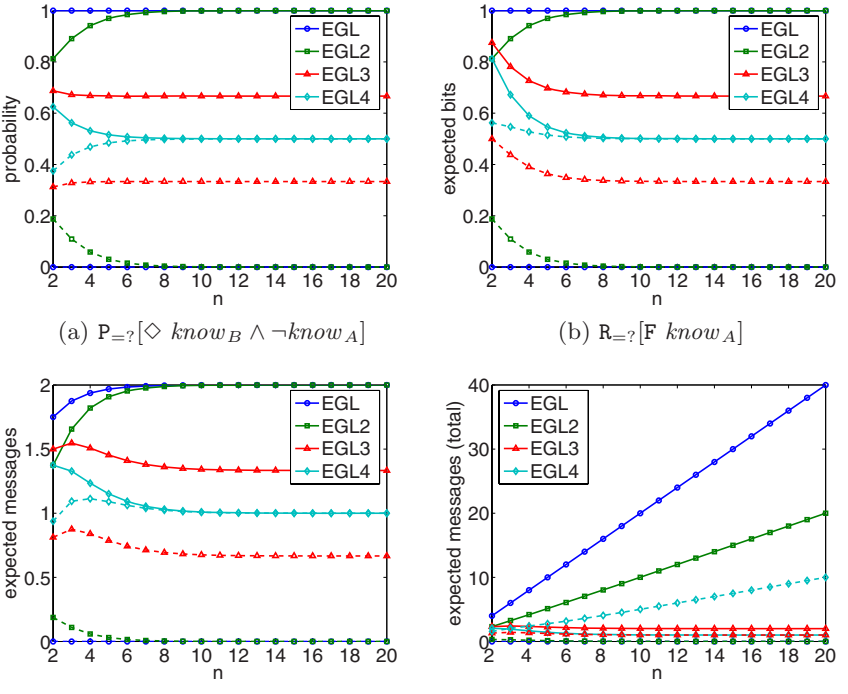
5.2 Case Study 1: Probabilistic Contract Signing

This case study, taken from [51], concerns the probabilistic contract signing protocol of Even, Goldreich and Lempel [25]. The protocol is designed to allow two parties, A and B, to exchange commitments to a contract. In an asynchronous setting, it is difficult to perform this task in a way that is fair to both parties, i.e. such that if B has obtained A's commitment, then A will always be able to obtain B's. In the Even, Goldreich and Lempel (EGL) protocol, the parties A and B each generate a set of pairs of secrets which are then revealed to the other party in a probabilistic fashion. A is committed to the contract once B knows both parts of one of A's pairs of secrets (and vice versa).

PRISM was used to identify a weakness of the protocol [51,53], showing that, by quitting the protocol early, one of the two parties (the one which did not initiate the protocol) can be at an advantage by being in possession of a complete pair of secrets while the other party knows no complete pairs. Various modifications to the basic EGL protocol were proposed [51,53] and PRISM was used to quantify the fairness of each.

The model is constructed as a DTMC and below we list the range of PCTL properties relating to party A that have been studied with PRISM (the dual properties for party B have also been studied). For each property we also state any modification to the model or reward structure required and explain the relevance of the property to the performance of the protocol.

- $\mathtt{P}_{=?}[\Diamond\ know_B \wedge \neg know_A]$ – the probability of reaching a state where A does not know a pair while B does know a pair. This measure can be interpreted as the "chance" that the protocol is unfair towards either party.
- $\mathtt{R}_{=?}[\mathtt{F}\ done]$ - the expected number of bits that A needs to know a pair once B knows a pair. In this case the model of the protocol was modified by adding a transition to the final state *done* as soon as B knows a pair and assigning to this transition a reward equal to the number of bits that A requires to know a pair. This property is a quantification of how unfair the protocol is with respect to either party.
- $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ - once B knows a pair, the expected number of messages from B that A needs to know a pair. The reward structure in this case associates a reward of 1 to all transitions which correspond to B sending a message to A



(a) $\mathtt{P}_{=?}[\Diamond\ know_B \wedge \neg know_A]$ (b) $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$

(c) $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ (d) $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ (total messages)

Fig. 7. Model checking results for the EGL contract signing protocol

from a state where B already knows a pair. This measure can be interpreted as an indication of how much influence a corrupted party has on the fairness of the protocol, since a corrupted party can try and delay these messages in order to gain an advantage.

– $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ - once B knows a pair, the expected total number of messages that need to be sent (by either party) before A knows a pair. In this case we assign a reward of 1 to any transition which corresponds to either B sending a message to A or A sending a message to B in a state where B already knows a pair. This measure can be interpreted as representing the "duration" of unfairness, that is, the time that one of the parties has an advantage.

Fig. 7 shows plots of these values for both the basic protocol (EGL) and three modifications (EGL2, EGL3 and EGL4). The solid lines and dashed lines represent the values for parties A and B, respectively (where process B initiated the protocol). The data is computed for a range of values of n: the number of pairs of secrets which each party generates.

The results show EGL4 is the 'fairest' protocol except for the 'duration of fairness measure' (expected messages that need to be sent for a party to know a

pair once the other party knows a pair). For this measure, the value is larger for B than for A and, in fact, as n increases, this measure increases for B but decreases for A. In [51] a solution is proposed and analysed which merges sequences of bits into a single message. For further details on this case study see [51] and the PRISM website [53].

5.3 Case Study 2: Dynamic Power Management

Dynamic Power Management(DPM) is a technique for saving energy in devices which can be turned on and off under operating system control. DPM has gained considerable attention over the last few years, both in the research literature and in the industrial setting, with schemes such as OnNow and ACPI becoming prevalent. One of the main reasons for this interest is the continuing growth in the use of mobile, hand-held and embedded devices, for which minimisation of power consumption is a key issue.

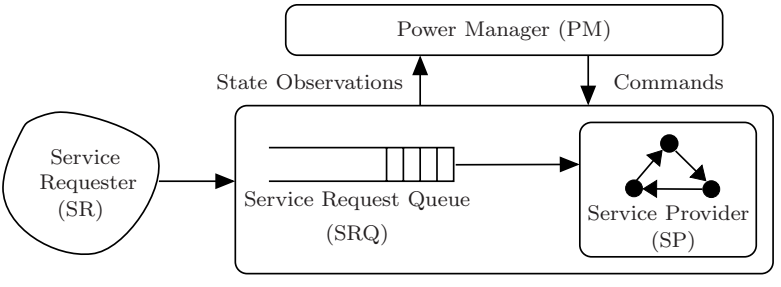


Fig. 8. The DPM System Model

DPM-enabled devices typically have several *power states* with different power consumption rates. A DPM *policy* is used to decide when commands to transition between these states should be issued, based on the current state of the system. In this case study we consider only simple policies, so called N-policies, which 'switch on' when the queue of requests awaiting service is greater than or equal to N, and 'switch off' when the queue becomes empty.

The basic structure of the DPM model can be seen in Fig. 8. The model consists of: a Service Provider (SP), which represents the device under power management control; a Service Requester (SR), which issues requests to the device; a Service Request Queue (SRQ), which stores requests that are not serviced immediately; and the Power Manager (PM), which issues commands to the SP, based on observations of the system and a stochastic DPM policy.

This case study is based on a CTMC model of a Fujitsu disk drive [54]. The SP has three power states: *sleep*, *idle* and *busy*. In *sleep* the SP is inactive and no requests can be served. In *idle* and *busy* the SP is active; the difference is that *idle* corresponds to the case when the SP is not working on any requests (the SRQ is empty) and *busy* it is actively working on requests (the SRQ is not empty). Transitions between *sleep* and *idle* are controlled by the PM (that is,

	sleep	*idle*	*busy*
sleep	0	1.6	-
idle	0.67	0	0
busy	-	0	0

(a) Transition time

	sleep	*idle*	*busy*
sleep	0	7	-
idle	0.067	0	0
busy	-	0	0

(b) Energy consumed

	sleep	*idle*	*busy*
av. power	0.13	0.95	2.15
av. service	0	0	0.008

(c) Power and service times

Fig. 9. Transition times, energy and power consumption and service times for the SP

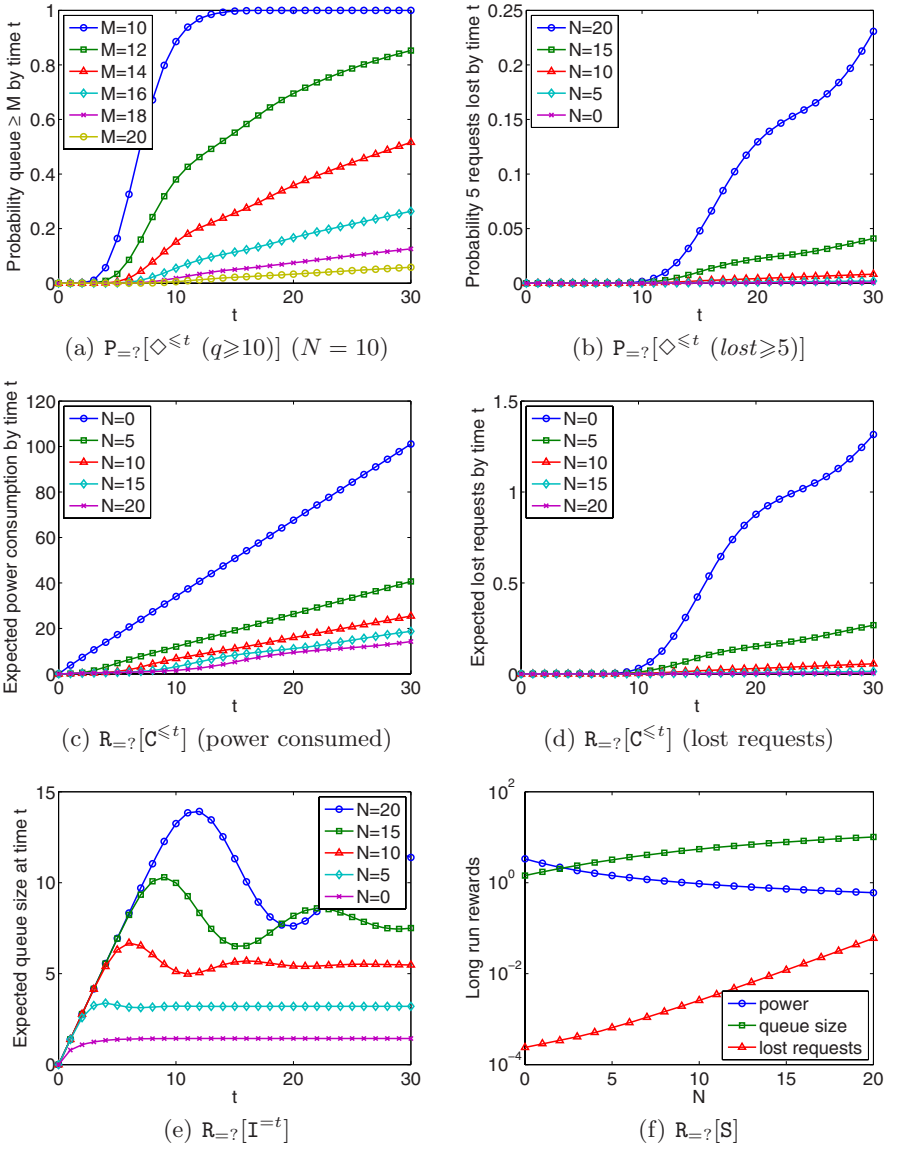
by the DPM policy), while transitions between *idle* and *busy* are controlled by the state of the SRQ. Fig. 9(a) shows the average times for transitions between power states, Fig. 9(b) show the energy used for these transitions and Fig. 9(c) the average power consumption and service times for each state. The SR models the inter-arrival distribution of requests given by exponential distribution with rate 100/72 and the SRQ models a service request queue which has a maximum size of 20. Note that, if a request arrives from the SR and the queue is full (20 requests are already awaiting service), then they are presumed lost.

The three reward structures constructed for this case study which are outlined below.

1. The first reward structure, used to investigate the power consumption of the system, is defined using the energy and power consumption of the SP given in Fig. 9. More precisely, the state rewards equal the average power consumption of the SP in that state and the transition reward for transitions in which the SP changes state is assigned the energy consumed by the corresponding state change.
2. The second reward structure, used for analysing the size of the service request queue, is obtained by setting the reward in each state to the size of the SRQ in that state (there are no transition based rewards);
3. The third reward structure, used when calculating the number of lost requests, assigns a reward of 1 to any transition representing the arrival of a request in a state where the queue is full (there are no state rewards in this case).

Below we list a range of CSL properties that have been studied for this case study in PRISM.

- $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (q \geqslant M)]$ – the probability that the queue size becomes greater than or equal to M by t;
- $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (\mathit{lost} \geqslant M)]$ – the probability that at least M requests get lost by t;
- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ – the expected power consumption by t or the expected number of lost customers by time t (depending on whether the first or third reward structure is used);
- $\mathtt{R}_{=?}[\mathtt{I}^{=t}]$ – the expected queue size at t (using the second reward structure);
- $\mathtt{R}_{=?}[\mathtt{S}]$ – the long run average power consumption, long run average queue size or long run average number of requests lost per unit time (depending on which reward structure is used).



(a) $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (q{\geqslant}10)]$ ($N = 10$)

(b) $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (\mathit{lost}{\geqslant}5)]$

(c) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ (power consumed)

(d) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ (lost requests)

(e) $\mathtt{R}_{=?}[\mathtt{I}^{=t}]$

(f) $\mathtt{R}_{=?}[\mathtt{S}]$

Fig. 10. Range of results for the DPM case study obtained with PRISM

Fig. 10 presents a range of the results obtained with PRISM for this case study. The results demonstrate, as expected, that increasing N decreases the power consumption, while increasing both the queue size and the number of lost requests. For further details about DPM see, for example, [14,55] and for probabilistic model checking of DPM [50].

5.4 Case Study 3: Fibroblast Growth Factors

The final case study concerns a biological pathway for Fibroblast Growth Factors taken from [32]. Fibroblast Growth Factors (FGF) are a family of proteins which play a key role in the process of cell signalling in a variety of contexts, for example wound healing. The model is a CTMC and it incorporates protein-protein interactions (including competition for partners), phosphorylation and dephosphorylation, protein complex relocation and protein complex degradation (via ubiquitin-mediated proteolysis). Fig. 11 illustrates the different components in the pathway and their possible bindings.

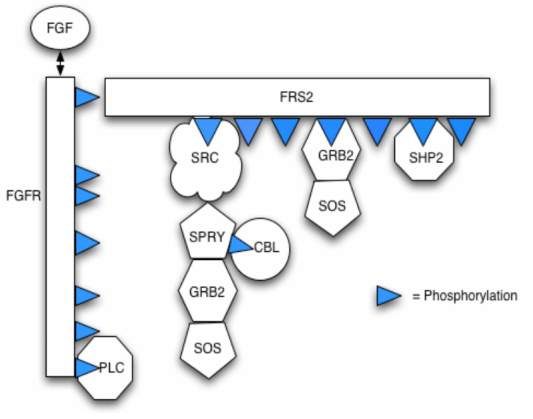
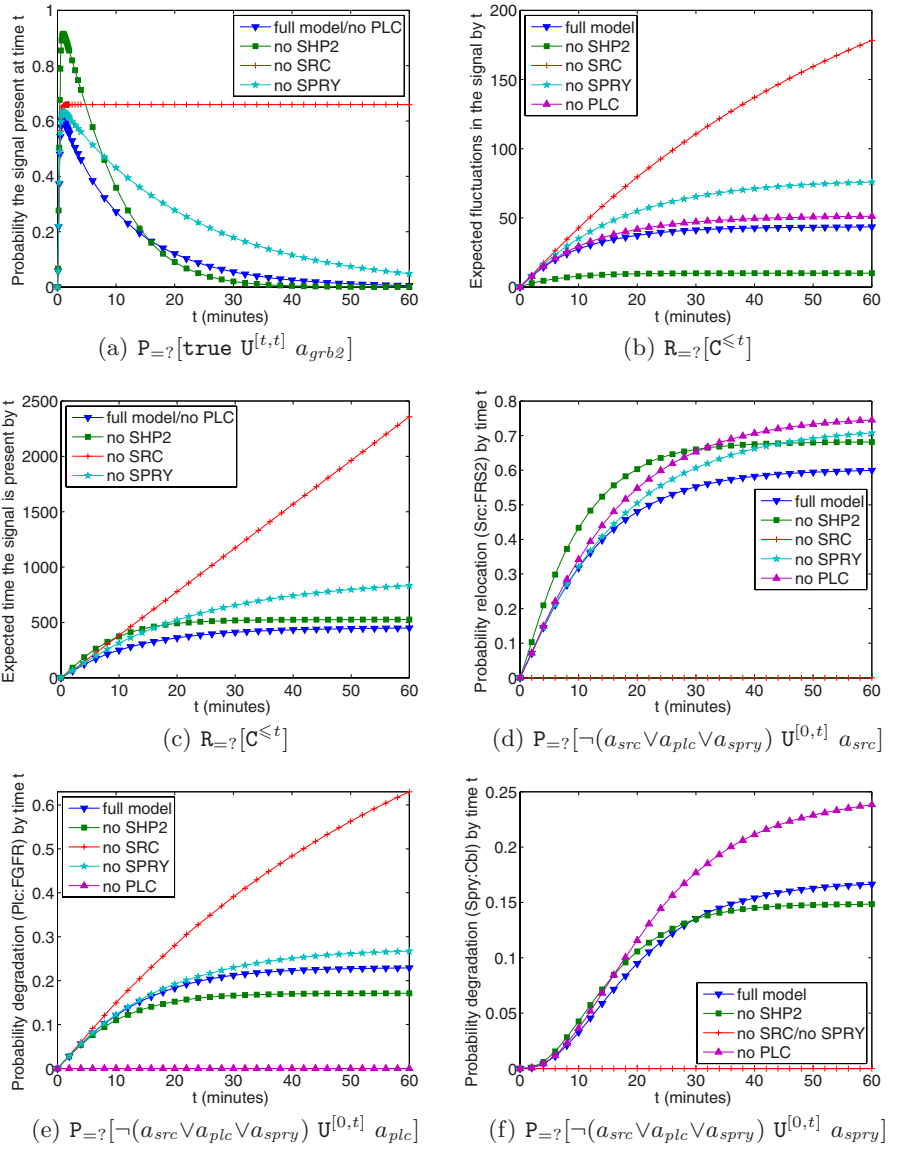


Fig. 11. Diagram showing the different possible bindings in the pathway

In [32] a base model, representing the full system, was developed. Subsequently, a series of 'in silico genetics' experiments on the model designed to investigate the roles of the various components of the activated receptor complex in controlling signalling dynamics. This involves deriving a series of modified models of the pathway where certain components are omitted (Shp2, Src, Spry or Plc), and is easily achieved in a PRISM model by just changing the initial value of the component under study. Below, we present a selection of the various CSL properties of the model that were analysed including, for properties relating to rewards, an explanation of the corresponding reward structure.

- $\mathtt{P}_{=?}[\Diamond^{[t,t]}\ a_{grb2}]$ - the probability that Grb2 is bound to FRS2 at the time instant t.
- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ - the expected number of times that Grb2 binds to FRS2 by time t. In this case, the only non-zero rewards are associated with transitions involving Grb2 binding to FRS2 which have a reward 1.
- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ – the expected time that Grb2 spends bound to FRS2 within the first T time units. The reward structure for this property assigns a reward of 1 to all states where Grb2 is bound to FRS2 and 0 to all other states and transitions.
- $\mathtt{S}_{=?}[a_{grb2}]$ - the long-run probability that Grb2 is bound to FRS2.



(a) $\mathtt{P}_{=?}[\mathtt{true}\ \mathtt{U}^{[t,t]}\ a_{grb2}]$ (b) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$

(c) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ (d) $\mathtt{P}_{=?}[\neg(a_{src}\vee a_{plc}\vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{src}]$

(e) $\mathtt{P}_{=?}[\neg(a_{src}\vee a_{plc}\vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{plc}]$ (f) $\mathtt{P}_{=?}[\neg(a_{src}\vee a_{plc}\vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{spry}]$

Fig. 12. Transient numerical results

- $\mathtt{R}_{=?}[\mathtt{F}\ (a_{src}\vee a_{plc}\vee a_{spry})]$ - the expected number of times Grb2 binds to FRS2 before degradation or relocation occurs. As in the second property, transitions involving Grb2 binding to FRS2 are assigned reward 1.
- $\mathtt{R}_{=?}[\mathtt{F}\ (a_{src}\vee a_{plc}\vee a_{spry})]$ - the expected time Grb2 spends bound to FRS2 before degradation or relocation occurs As for the third property, all states where Grb2 is bound to FRS2 have a reward of 1.

Table 1. Long run and expected reachability properties for the signal

	$\mathtt{S}_{=?}[a_{grb2}]$	$\mathtt{R}_{=?}[\mathtt{F}\ (a_{src} \vee a_{plc} \vee a_{spry})]$	
		bindings	time (min)
full model	7.54e-7	43.1027	6.27042
no Shp2	3.29e-9	10.0510	7.78927
no Src	0.659460	283.233	39.6102
no Spry	4.6e-6	78.3314	10.8791
no Plc	0.0	51.5475	7.56241

Table 2. Probability and expected time until degradation/relocation in the long run

	$\mathtt{P}_{=?}[\neg(a_{src} \vee a_{plc} \vee a_{spry})\ \mathtt{U}\ a_{xxx}]$			$\mathtt{R}_{=?}[\mathtt{F}\ (a_{src} \vee a_{plc} \vee a_{spry})]$
	$xxx = src$	$xxx = plc$	$xxx = spry$	(min)
full model	0.602356	0.229107	0.168536	14.0258
no Shp2	0.679102	0.176693	0.149742	10.5418
no Src	-	1.0	0.0	60.3719
no Spry	0.724590	0.275410	-	16.8096
no Plc	0.756113	-	0.243887	17.5277

- $\mathtt{P}_{=?}[\neg(a_{src} \vee a_{plc} \vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{src}]$ - the probability that degradation or relocation occurs by by time t and Src is the cause.
- $\mathtt{P}_{=?}[\neg(a_{src} \vee a_{plc} \vee a_{spry})\ \mathtt{U}\ a_{plc}]$ - the probability that Plc is the first cause of degradation or relocation.
- $\mathtt{R}_{=?}[\mathtt{F}\ (a_{src} \vee a_{plc} \vee a_{spry})]$ - the expected time until degradation or relocation occurs in the pathway. For this property all states are assigned reward 1 (and all transitions are assigned reward 0).

Fig. 12 presents results relating to the transient properties, while Tables 1 and 2 consider long-run properties. Note that the results of Table 1 and Table 2 can be regarded as the values of Fig. 12(a)–(c) and Fig. 12(d)–(f) in "the limit", i.e. as t tends to infinity. For further details on the case study see [32] and the PRISM website [53].

6 Conclusions

In this tutorial we have presented an overview of stochastic model checking, covering both the theory and practical aspects for two important types of probabilistic models, discrete- and continuous-time Markov chains. Algorithms were given for verifying these models against probabilistic temporal logics PCTL and CSL and their extensions with the reward operator. The probabilistic model checker PRISM, which implements these algorithms, was used to analyse three real-world case studies also described here. However, there are many other aspects of stochastic model checking not covered in this tutorial and below we attempt to give brief pointers to related and further work.

More expressive logics than PCTL have been proposed, including LTL and PCTL* [6,15]. For the corresponding model checking algorithms see [62,22,6,15,13]. We also mention the alternative reward extension of PCTL given in [2]. With regards to CTMCs, a number of extensions of CSL have been proposed in the literature, along with associated model checking algorithms. For example, [35] proposes an action based version of CSL; [31,9] introduce the logics CRL and CSRL which added support for reward-based properties [42]; and [44] augment CSL with random time-bounded until and random expected-time operators, respectively.

This tutorial concentrated on stochastic model checking. Related topic include: probabilistic generalisations of bisimulation and simulation relations for DTMCs [49,57] and for CTMCs [17,11]; and approximate methods for stochastic model checking based on discrete event simulation [33,63]. Stochastic model checkers SMART [19], $\mathsf{E}\vdash\mathsf{MC}^2$[34] and MRMC [39] have similarities with the PRISM model checker described here. Finally, we mention a challenging direction of research is into the verification of models which allow more general probability distributions. While the restriction to exponential distributions imposed by CTMCs is important for the tractability of their model checking, it may prove too simplistic for some modelling applications. See [28] for an introduction to this area.

References

1. R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
2. S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In K. Larsen and P. Niebert, editors, *Proc. Formal Methods for Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 88–104. Springer, 2003.
3. J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, 1990.
4. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.
5. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.
6. A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, *Proc. 7th Int. Conf. Computer Aided Verification (CAV'95)*, volume 939 of *LNCS*, pages 155–165. Springer, 1995.
7. I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.
8. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In A. Emerson and A. Sistla, editors, *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 358–372. Springer, 2000.

9. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In U. Montanari, J. Rolim, and E. Welzl, editors, *Proc. 27th Int. Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.
10. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.
11. C. Baier, J. Katoen, H. Hermanns, and B. Haverkort. Simulation for continuous-time Markov chains. In L. Brim, P. Jancar, M. Kretinzki, and A. Kucera, editors, *Proc. Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 338–354. Springer, 2002.
12. 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 Int. Conf. Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.
13. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.
14. L. Benini, A. Bogliolo, G. Paleologo, and G. D. Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(3):299–316, 2000.
15. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
16. P. Billingsley. *Probability and Measure*. Wiley, 1995.
17. P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.
18. L. Cheung. Randomized wait-free consensus using an atomicity assumption. In *Proc. 9th International Conference on Principles of Distributed Systems (OPODIS'05)*, 2005.
19. G. Ciardo, R. Jones, A. Miner, and R. Siminiceanu. Logic and stochastic modeling with smart. *Performance Evaluation*, 63(6):578–608, 2006.
20. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logics. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
21. E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10((2/3):149–169, 1997.
22. C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE Computer Society Press, 1988.
23. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
24. C. Daws, M. Kwiatkowska, and G. Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. *Int. Journal on Software Tools for Technology Transfer*, 5(2–3):221–236, 2004.
25. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
26. W. Fokkink and J. Pang. Variations on itai-rodeh leader election for anonymous rings and their analysis in prism. *Journal of Universal Computer Science*, 12(8):981–1006, 2006.

27. B. Fox and P. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.
28. R. German. *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.
29. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
30. B. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, 1988.
31. B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *Proc. Int. Conf. Dependable Systems and Networks (DSN'02)*. IEEE Computer Society Press, 2002.
32. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In C. Priami, editor, *Proc. Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Bioinformatics*, pages 32–47. Springer, 2006.
33. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In B. Steffen and G. Levi, editors, *Proc. Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 73–84. Springer, 2004.
34. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach, editors, *Proc. 6th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 347–362. Springer, 2000.
35. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In W. Grieskamp and T. Santen, editors, *Proc. Integrated Formal Method (IFM 2000)*, volume 1945 of *LNCS*, pages 420–439. Springer, 2000.
36. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
37. IEEE standard for a high performance serial bus. IEEE Computer Society, IEEE Std 1394-1995.
38. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.
39. J.-P. Katoen, M. Khattri, and I. Zapreev. A Markov reward model checker. In *Proc. Second Int. Conf. Quantitative Evaluation of Systems (QEST 05)*, pages 243–244. IEEE Computer Society Press, 2005.
40. J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint Int. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 23–38. Springer, 2001.
41. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer, 2nd edition, 1976.
42. M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking CSL until formulae with random time bounds. In H. Hermanns and R. Segala, editors, *Proc. 2nd Joint Int. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 152–168. Springer, 2002.

43. M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. In *Proc. 2nd Euro-Japanese Workshop on Stochastic Risk Modelling for Finance, Insurance, Production and Reliability*, 2002.
44. M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. *Computers & Mathematics with Applications*, 51(2):305–316, 2006.
45. M. Kwiatkowska, G. Norman, and D. Parker. PRISM users' guide. Available from www.cs.bham.ac.uk/~dxp/prism.
46. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *Int. Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.
47. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.
48. M. Kwiatkowska, D. Parker, Y. Zhang, and R. Mehmood. Dual-processor parallelisation of symbolic probabilistic model checking. In D. DeGroot and P. Harrison, editors, *Proc. 12th Int. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, pages 123–130. IEEE Computer Society Press, 2004.
49. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
50. G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. *Formal Aspects of Computing*, 17(2):160–176, 2005.
51. G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.
52. D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems.* PhD thesis, University of Birmingham, 2002.
53. PRISM web site. www.cs.bham.ac.uk/~dxp/prism.
54. Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: Construction and optimization. In *Proc. Int. Symposium on Low Power Electronics and Design*, 1999.
55. Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: construction and optimization. *IEEE Transactions on Computer Aided Design*, 20(10):1200–1217, 2001.
56. J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), volume 23 of *CRM Monograph Series.* American Mathematical Society, 2004.
57. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *Proc. 5th Int. Conf. Concurrency Theory (CONCUR'94)*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.
58. F. Somenzi. CUDD: Colorado University decision diagram package. Public software, Colorado Univeristy, Boulder, http://vlsi.colorado.edu/~fabio/, 1997.
59. W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton, 1994.
60. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
61. K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley & Sons, 2001.

62. M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.
63. H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *Int. Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006.