Discrete-time rewards model-checked

Suzana Andova[a], Holger Hermanns[a,b], and Joost-Pieter Katoen[a]

[a] *Formal Methods and Tools Group, Department of Computer Science*
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

[b] *Department of Computer Science*
Saarland University, D-66123 Saarbrücken, Germany

Abstract. This paper presents a model-checking approach for analyzing discrete-time Markov reward models. For this purpose, the temporal logic probabilistic CTL is extended with reward constraints. This allows to formulate complex measures – involving expected as well as accumulated rewards – in a precise and succinct way. Algorithms to efficiently analyze such formulae are introduced. The approach is illustrated by model-checking a probabilistic cost model of the IPv4 zeroconf protocol for distributed address assignment in ad-hoc networks.

1 Introduction

Modelling techniques such as queueing networks and probabilistic variants of Petri nets, automata networks and process algebra are convenient means to describe performance and dependability models. Based on a high-level specification of the system under investigation, the underlying model – albeit a continuous time or a discrete time Markov chain (CTMC or DTMC) – is automatically obtained and can be analyzed with well-studied means to obtain transient and stationary measures. Most of these techniques have been extended to CTMCs (or DTMCs) augmented with costs, or dually bonuses (rewards); approaches using stochastic reward nets [7], reward-based variants of process algebra [4] extensions of automata [15], logic-based approaches [8] and so on, have been proposed. These formalisms provide adequate means to specify performance and dependability *models*.

It is fair to say that the specification of performance or dependability *measures* in a high-level manner has received far less attention. In recent works, we have proposed to use appropriate extensions of temporal logic – as typically used to reason about the functional correctness of systems – for specifying constraints over such measures [2, 3]. This technique allows to specify standard (e.g., transient and stationary) and complex measures in a precise, unambiguous and lucid manner. Even more importantly, this specification technique is complemented by powerful means to automatically check constraints on measures over finite Markovian models using a light-weight extension of model checking [9]. This hides specialized algorithms from the performance engineer, supports automated measure-driven model adaptation, and allows for the checking of quantitative as well as functional properties (such as absence of deadlocks) in a single integrated framework.

The model-checking algorithms for CTMCs rely on well-developed standard numerical algorithms. Therefore even the more intricated measures – beyond standard stationary and transient measures – can be checked rather efficiently. Further work in this area has focussed on CTMCs decorated with rewards. We have introduced a logic to specify measures over such so-called continuous-time Markov reward models (CMRMs) [2, 12]. The logic allows one to express a rich spectrum of measures. For instance, when rewards are interpreted as costs, this logic can express a constraint on the probability that, given a start state, a certain goal can be reached within t time units while deliberately avoiding to visit certain intermediate states, and with a total cost (i.e., accumulated reward) below a given threshold. Such path-based measures are, however, computationally expensive as they are based on determining transient reward distributions, a measure that has not been widely addressed in the literature and for which the rarely available algorithms are highly time- and/or space-consuming [12].

In this paper, we aim to avoid this inefficiency by considering discrete-time Markov chains instead, decorated with (possibly multiple) state rewards. This paper introduces a logic and model-checking algorithms for discrete time Markov reward models (DMRMs). In particular, we extend probabilistic CTL [11] with operators to reason about long-run average, and more importantly, by operators that allow to specify constraints on (i) the expected reward rate at a time instant, (ii) the long-run expected reward rate per time unit, (iii) the cumulated reward rate at a time instant – all for a specified set of states – and (iv) the cumulated reward over a time interval. The proposed logic allows to specify non-trivial, though interesting, constraints such as *"the probability to reach one of the goal states (via indicated allowed states) within n steps while having earned an accumulated reward that does not exceed r is larger than 0.92"*. We present model-checking algorithms that verify such properties in an efficient manner, and show how these can be extended to multiple rewards in a straightforward way. The approach is illustrated by checking some properties of the IPv4 zeroconf protocol for distributed address assignment in ad-hoc networks.

2 Discrete-time Markov reward models

This section presents the basic concepts of discrete-time Markov reward models that are needed for the rest of the paper. For more details we refer to [14].

DMRMs. In order to enable the logical specification of measures-of-interest over performability models we consider a slight extension of traditional Markov models where states are equipped with elementary properties, the so-called atomic propositions. Let AP be a fixed, finite set of atomic propositions.

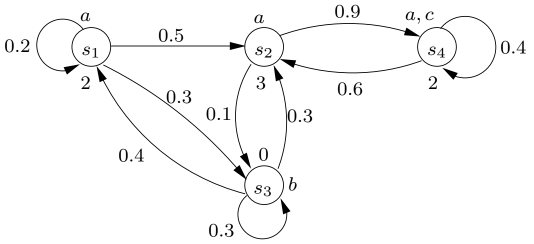
Definition 1. *A (labelled) DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, L)$ where S is a finite set of* states*, $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a* probability matrix *such that $\sum_{s' \in S} \mathbf{P}(s, s') \in \{0, 1\}$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a* labelling *function that assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in s.*

Definition 2. *A discrete-time reward model (DMRM) $\mathcal{M}$ is a pair (D, ρ) with DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ and $\rho : S \rightarrow I\!R_{\geqslant 0}$ a* reward *assignment function that associates a real reward (or: cost) to any state in S. Real number $\rho(s)$ denotes the reward earned on leaving state s.* [1]

Paths. Let $\mathcal{M}$ be a DMRM with underlying DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ and reward function ρ. An infinite path σ is a sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ where $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for $i \geqslant 0$. For $i \in I\!N$ let $\sigma[i] = s_i$, the state occupied after i transitions. A finite path σ with length n is a sequence $s_0 \rightarrow \ldots \rightarrow s_n$ with $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for $0 \leqslant i < n$. *Path*(s) denotes the set of (finite and infinite) paths starting in s. Let $\Pr_s$ denote the unique probability measure on sets of paths that start in state s [3]. The *cumulative reward* along finite path σ with length n is defined as $\rho(\sigma) = \sum_{i \geqslant 0}^{n-1} \rho(s_i)$. Note that rewards are considered on leaving a state, i.e., $\rho(s_n)$ is not considered in the cumulative reward of σ.

Example 1. Consider the DMRM $\mathcal{M}$ depicted below with $S = \{ s_1, s_2, s_3, s_4 \}$, $L(s_1) = L(s_2) = \{ a \}$, $L(s_3) = \{ b \}$ and $L(s_4) = \{ a, c \}$, reward structure ρ defined by $\rho(s_1) = 2, \rho(s_2) = 3, \rho(s_3) = 0, \rho(s_4) = 2$, and the probability matrix defined by:

$$\mathbf{P} = \begin{bmatrix} 0.2 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0.1 & 0.9 \\ 0.4 & 0.3 & 0.3 & 0 \\ 0 & 0.6 & 0 & 0.4 \end{bmatrix}$$



An example finite path is $\sigma = s_1\, s_2\, s_3\, s_2\, s_4$; we have $\sigma[0] = s_1$ and $\sigma[3] = s_2$, and the cumulative reward $\rho(\sigma)$ equals 8. The probability of σ, $\Pr_{s_1}(\sigma) = \frac{1}{2} \cdot \frac{1}{10} \cdot \frac{3}{10} \cdot \frac{9}{10}$.

Transient and limiting behavior. Transient analysis studies the system at a certain time instant n. Let $\pi(s, s', n)$ denote the probability that the system is in state s' after n steps given that the system started in state s. These transition probabilities can be calculated using the Chapman-Kolmogorov equations:

$$\pi(s, s', n) = \sum\nolimits_{t \in S} \pi(s, t, i) \cdot \pi(t, s', n{-}i) \text{ for } 0 \leqslant i \leqslant n, \qquad (1)$$

where $\pi(s, s', 0) = 0$ if $s' \neq s$ and $\pi(s, s, 0) = 1$. When n tends to infinity, one considers the limiting (i.e., long-run) behaviour of DTMCs. The limiting behaviour of a DTMC strongly depends on the structure of the considered chain, more specifically, on the capacity of states reaching each other within finitely many steps. It is well known that in case of an irreducible finite aperiodic DTMC the limit $\lim_{n \rightsquigarrow \infty} \pi(s, s', n)$ exists and precisely characterises the limiting probabilities $\pi(s, s')$, also called steady-state probabilities, of the DTMC [13]. If the considered DTMC is irreducible and periodic then this limit does not exist. In that case, one considers the long-run fraction of time that the system spends in state s' when starting in state s:

[1] Here we consider rewards to be constant, but there do exist variants in which rewards are random variables.

$$\pi(s,s') = \lim_{n\rightsquigarrow\infty} \frac{1}{n+1} \sum\nolimits_{i=0}^{n} \pi(s,s',i) \tag{2}$$

The probabilities $\pi(s,s')$ can (in both cases) be characterised as the unique solution of the following system of linear equations:

$$\pi(s) = \sum\nolimits_{s' \in S} \pi(s') \cdot \mathbf{P}(s,s') \text{ such that } \sum\nolimits_{s \in S} \pi(s) = 1. \tag{3}$$

For irreducible aperiodic DTMCs, $\pi(s,s')$ coincides with $\lim_{n\rightsquigarrow\infty} \pi(s,s',n)$, see e.g., [14]. Although the initial state does not have any influence on the value of $\pi(s,s')$, we keep this notation because in the case of reducible DTMCs the initial state has influence on the limiting behaviour. Let $\pi(s,S')$ denote $\sum_{s' \in S'} \pi(s,s')$ for $S' \subseteq S$.

Reward measures. For DMRMs the following reward measures are considered, see also [18]. Assume that the system starts in state s.

- The expected reward rate per time-unit up to time instant n, denoted $g(s,n)$, and its limiting counterpart, the long-run expected reward rate per time-unit, denoted $g(s)$. They are defined as follows:
$$g(s,n) = \frac{1}{n+1} \sum\nolimits_{i=1}^{n} E(\rho(\sigma_s[i])) \quad \text{and} \quad g(s) = \lim_{n\rightsquigarrow\infty} g(s,n)$$
where (the random variable) σ_s ranges over $\mathit{Path}(s)$.
- The instantaneous reward at time instant n: $\rho(s,s',n) = \pi(s,s',n) \cdot \rho(s')$. For $S' \subseteq S$ let $\rho(s,S',n) = \sum_{s' \in S'} \rho(s,s',n)$, i.e., $\rho(s,S',n)$ is the instantaneous reward at time instant n in the set S'.
- The expected accumulated reward until the n-th transition is defined as follows: $y(s,n) = \sum_{i=0}^{n-1} \rho(s,S,i)$. According to the definition of path reward, the sum goes up to $n-1$, i.e., the reward of the last state of the path is ignored. An alternative characterisation of this reward measure is: $y(s,n) = \sum_{\sigma \in \mathit{Path}(s,s',n)} \rho(\sigma) \cdot \Pr_s(\sigma)$, where $\mathit{Path}(s,s',n)$ denotes the set of finite paths of length n that start in s and end in s'.

Multiple rewards. If various measures-of-interest are to be determined for a Markov model, typically several different reward structures are imposed, see e.g., [18, Section II]. For $k > 0$ reward structures, a DMRM is a $(k{+}1)$-tuple $(\mathcal{D}, \rho_1, \ldots, \rho_k)$ with $\mathcal{D}$ a DTMC and ρ_j a reward assignment function, for $0 < j \leqslant k$. The reward measures defined above can now be all considered for each of the different reward assignments ρ_j in a fairly straightforward manner. Let $\rho_j(\sigma)$ be the accumulated j-th reward along finite path σ, i.e., for $\sigma = s_0 \rightarrow \ldots \rightarrow s_n$ we have $\rho_j(\sigma) = \sum_{i \geqslant 0}^{n-1} \rho_j(s_i)$. The instantaneous j-th reward at time instant n is defined by: $\rho_j(s,s',n) = \pi(s,s',n) \cdot \rho_j(s')$. The other reward measures are generalised in a similar manner.

3 Probabilistic reward CTL

This section introduces the logic Probabilistic Reward CTL (PRCTL) that is aimed at the specification of performability measures over discrete-time Markov reward models. To simplify the presentation we first recall Probabilistic CTL (PCTL) by Hansson and Jonsson [11], and extend it by a long-run average operator.

PCTL with long-run average. Let $a \in AP$, $p \in [0,1]$, n be a natural (or ∞) and binary comparison operator $\unlhd \in \{\leqslant, <, \geqslant, >\}$. The syntax of PCTL is:

$$\Phi ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\Phi \;\Big|\; \mathcal{L}_{\unlhd p}(\Phi) \;\Big|\; \mathcal{P}_{\unlhd p}(\Phi\, \mathcal{U}^{\leqslant n}\, \Phi)$$

The other boolean connectives are derived in the usual way. For the sake of simplicity, we do not consider the next state operator in this paper. The standard (i.e. unbounded) until formula is obtained by taking n equal to ∞, i.e., $\Phi\,\mathcal{U}\,\Psi = \Phi\,\mathcal{U}^{\leqslant\infty}\,\Psi$. Temporal operators like $\Diamond$, $\Box$ and their timed variants $\Diamond^{\leqslant n}$ or $\Box^{\leqslant n}$ can be derived, e.g., $\mathcal{P}_{\unlhd p}(\Diamond^{\leqslant n}\,\Phi) = \mathcal{P}_{\unlhd p}(\mathrm{tt}\,\mathcal{U}^{\leqslant n}\,\Phi)$ and $\mathcal{P}_{\geqslant p}(\Box\,\Phi) = \mathcal{P}_{\leqslant 1-p}(\Diamond\,\neg\Phi)$. Let $Sat(\Phi) = \{\, s \mid s \models \Phi \,\}$ be the set of states that satisfy Φ. The semantics of PCTL is defined by [11]:

$$\begin{array}{llll}
s \models \mathrm{tt} & \text{for all } s \in S & s \models \Phi \wedge \Psi & \text{iff } s \models \Phi \wedge s \models \Psi \\
s \models a & \text{iff } a \in L(s) & s \models \mathcal{L}_{\unlhd p}(\Phi) & \text{iff } \pi(s, Sat(\Phi)) \unlhd p \\
s \models \neg\Phi & \text{iff } s \not\models \Phi & s \models \mathcal{P}_{\unlhd p}(\Phi\,\mathcal{U}^{\leqslant k}\,\Psi) & \text{iff } Prob(s, \Phi\,\mathcal{U}^{\leqslant k}\,\Psi) \unlhd p
\end{array}$$

$\mathcal{P}_{\unlhd p}(\Phi\,\mathcal{U}^{\leqslant k}\,\Psi)$ asserts that the probability measure of the paths that start in s and that satisfy $\Phi\,\mathcal{U}^{\leqslant k}\,\Psi$ meets the bound $\unlhd p$. The state formula $\mathcal{L}_{\unlhd p}(\Phi)$ asserts that the long-run average fraction of time for the set of Φ-states meets the bound $\unlhd p$. Here, $Prob(s, \Phi\,\mathcal{U}^{\leqslant k}\,\Psi) = \Pr_s\{\, \sigma \in Path(s) \mid \sigma \models \Phi\,\mathcal{U}^{\leqslant k}\,\Psi \,\}$. Formula $\Phi\,\mathcal{U}^{\leqslant n}\,\Psi$ asserts that Ψ will be satisfied within n steps and that all preceding states satisfy Φ, i.e.:

$$\sigma \models \Phi\,\mathcal{U}^{\leqslant n}\,\Psi \;\text{ iff }\; \exists j \leqslant n.\ (\sigma[j] \models \Psi \wedge \forall i < j.\, \sigma[i] \models \Phi)$$

Some example properties that can be expressed in PCTL for our running example are $\mathcal{P}_{\geqslant 0.3}(\Diamond b)$ (a b-state can be reached with probability at least 0.3), $\mathcal{P}_{\geqslant 0.3}(a\,\mathcal{U}^{\leqslant 3}\,b)$ (a b-state can be reached with probability at least 0.3 by at most 3 hops along a-states), and $\mathcal{L}_{\leqslant 0.5}(a)$ (the long-run average fraction of time spent in a-states is at most 0.5).

Syntax of PRCTL. We now extend the logic PCTL with ample means to specify properties that do not only address probabilistic aspects but in addition allow to specify constraints over reward measures. Some of the new operators are inspired by Baier *et al.* [3] who introduced a performability logic for continuous-time Markov reward models (with state rewards).

Let $J \subseteq \mathbb{R}_{\geqslant 0}$ be an interval on the real line, n a natural number, $p \in [0,1]$ and $N \subseteq \mathbb{N} \cup \{\infty\}$ an interval of natural numbers (or infinity). The syntax of PRCTL is defined by the following syntax clauses:

$$\begin{array}{c}
\Phi ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\Phi \;\Big|\; \mathcal{L}_{\unlhd p}(\Phi) \;\Big|\; \mathcal{P}_{\unlhd p}(\Phi\; \mathcal{U}_J^N\; \Phi) \;\Big| \\
\mathcal{E}_J^n(\Phi) \;\Big|\; \mathcal{E}_J(\Phi) \;\Big|\; \mathcal{C}_J^n(\Phi) \;\Big|\; \mathcal{Y}_J^n(\Phi)
\end{array}$$

The intuitive interpretation of these operators is as follows. Formula $\mathcal{E}_J^n(\Phi)$ asserts that the expected reward rate in Φ-states up to n transitions – reached at the n-th epoch – lies within the interval J. Formula $\mathcal{E}_J(\Phi)$ expresses that the long-run expected reward rate per time-unit for Φ-states meets the bounds of J. The formula $\mathcal{C}_J^n(\Phi)$ asserts that the instantaneous reward in Φ-states at the n-th epoch meets the bounds of J. Formula $\mathcal{Y}_J^n(\Phi)$ asserts that the expected accumulated reward in Φ-states until the n-th transition meets the bounds of J.

Semantics of PRCTL. The semantics of the state-formulas of PRCTL that are common with PCTL is identical to the semantics for PCTL as presented above. The semantics of the new operators is defined by:

$$\begin{array}{llll} s \models \mathcal{E}^n_J(\Phi) & \text{iff } g(s, Sat(\Phi), n) \in J & s \models \mathcal{C}^n_J(\Phi) & \text{iff } \rho(s, Sat(\Phi), n) \in J \\ s \models \mathcal{E}_J(\Phi) & \text{iff } g(s, Sat(\Phi)) \in J & s \models \mathcal{Y}^n_J(\Phi) & \text{iff } y(s, Sat(\Phi), n) \in J \end{array}$$

where we have that for $S' \subseteq S$:

$$g(s, S', n) = \frac{1}{n+1} \sum\nolimits_{i=0, \sigma_s[i] \in S'}^{n} E(\rho(\sigma_s[i])) \quad \text{and} \quad g(s, S') = \lim_{n \rightsquigarrow \infty} g(s, S', n).$$

Note that $g(s, n)$ as defined earlier coincides with $g(s, S, n)$. Stated in words, $g(s, S', n)$ denotes the expected reward rate up to the n-th epoch given that we are only interested in states belonging to the set S'. The expected accumulated reward for states in S' until the n-th transition is defined by: $y(s, S', n) = \sum_{i=0}^{n-1} \rho(s, S', i)$. Note that $y(s, n)$ as defined earlier coincides with $y(s, S, n)$.

Formula $\Phi \, \mathcal{U}^N_J \, \Psi$ asserts that Ψ will be satisfied within $j \in N$ steps, that all preceding states satisfy Φ, and that the accumulated reward until reaching the Ψ-state lies in the interval J. Formally:

$$\sigma \models \Phi \, \mathcal{U}^N_J \, \Psi \ \text{ iff } \exists j \in N. \left(\sigma[j] \models \Psi \wedge \forall i < j. \, \sigma[i] \models \Phi \ \wedge \ \sum\nolimits_{i=0}^{j-1} \rho(\sigma[i]) \in J \right)$$

Example 2. Some example properties that can be expressed in PRCTL for our running example are, $\mathcal{P}_{\geqslant 0.3}(a \mathcal{U}^{\leqslant 3}_{(23,\infty)} b)$ (a b-state can be reached with probability at least 0.3 by at most 3 hops along a-states accumulating costs of more than 23), and $\mathcal{Y}^3_{[3,5]}(a)$ (the accumulated costs expected within 3 hops is at least 3 and at most 5).

Multiple rewards. The logic PRCTL can easily be enhanced such that properties over models equipped with multiple reward structures can be treated. Suppose $\mathcal{M} = (\mathcal{D}, \rho_1, \ldots, \rho_k)$ is a DMRM with $k > 0$ reward structures, and let $0 < j \leqslant k$. The reward operators of PRCTL can be generalised in a straightforward manner such that constraints on all k reward structures can be expressed in a single formula. For instance, the formula $\mathcal{E}_{J_1,\ldots,J_k}(\Phi)$ expresses that the long-run expected reward rate per time-unit for Φ-states meets the bounds of J_1 for reward structure ρ_1, …, the bounds of J_k for reward structure ρ_k. Its semantics is defined by: $s \models \mathcal{E}_{J_1,\ldots,J_k}(\Phi)$ if and only if $g_j(s, Sat(\Phi)) \in J_j$ for all $0 < j \leqslant k$. The other operators can be generalised in a similar manner.

If extending to multiple rewards, it is actually possible to encode the time constraint N (in $\mathcal{U}^N_J$) into a reward constraint over a simple auxiliary reward structure.

4 Model-checking algorithms

Given a state s of DMRM $\mathcal{M}$ and a PRCTL-formula Φ, the question to be addressed is how to check whether or not Φ holds for state s, i.e., whether $s \models \Phi$ or $s \not\models \Phi$. The basic procedure is the same as for model-checking CTL [9]: the set $Sat(\Phi)$ of all states satisfying Φ is computed recursively and we have that $s \models \Phi$ if and only if $s \in Sat(\Phi)$. The recursive computation basically boils down to a

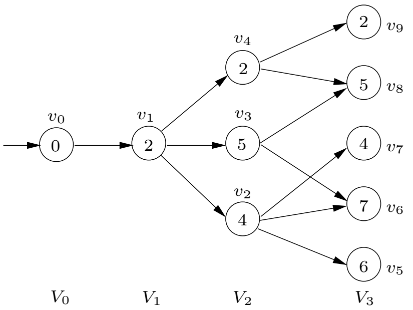
```
$V_0 := \{ 0 \}$;                              // only nodes with cumulative reward zero
$S_0 := \{ (s_0, 1) \}$;                  // state $s_0$ can be reached with probability one
for ($i := 0; i < n; i{+}{+}$)                            // $i$ is current level number
  foreach $m \in V_i$                              // check all rewards at level $i$
    foreach $(s, p) \in S_m$
        $m' := m + \rho(s)$;                               // new cumulative reward
        foreach $s'$ with $\mathbf{P}(s, s') > 0$          // all direct successors of $s$
          if $m' \notin V_{i+1}$                  // encountering fresh reward $m'$
          then $V_{i+1} := V_{i+1} \cup \{ m' \}$;                  // add new vertex
                 $S_{m'} := \{ (s', p{\cdot}\mathbf{P}(s, s')) \}$;
           elseif $(s', p') \in S_{m'}$          // shared node encountered?
                 then $p' := p' + p{\cdot}\mathbf{P}(s, s')$;
           else $S_{m'} := S_{m'} \cup \{ (s', p{\cdot}\mathbf{P}(s, s')) \}$;
           endif;
        endforeach;
    endforeach;
  endforeach;
endfor;
```

Fig. 1. Path graph generation algorithm

bottom-up traversal of the parse tree of the formula Φ. For the propositional fragment of PRCTL this goes along the lines of CTL. For determining $Sat(\mathcal{L}_{\trianglelefteq p}(\Phi))$ we use the method of [3]. Model-checking time-bounded until-formulae is based on the path graph generation.

Path graph generation. The basic concept of the algorithm is to compute the "unfolding" of the DMRM under consideration while keeping track of the accumulated reward so far. Nodes in the tree that have the same accumulated rewards are grouped together in a single vertex. The resulting directed acyclic graph (V, E, v_0) with finite (non-empty) set of vertices V, set of edges E and a distinguished initial vertex v_0, is called *path graph.* We have $V \subseteq \mathbb{N} \times \mathcal{P}(S \times (0, 1])$. The initial vertex v_0 equals $(0, \{ (s_0, 1) \})$ where s_0 is the state of the DMRM to be investigated, 0 is the accumulated reward so far, and 1 denotes that the probability to be in state s_0 equals one (when starting in s_0). In general, vertex $v = (k, S_k)$ with $S_k = \{ (s_1, p_1), \ldots, (s_m, p_m) \}$ denotes that starting from state s_0 each state s_i $(0 < i \leqslant m)$ can be reached with probability $p_i > 0$ (possibly via more than one path) while having earned a cumulative reward k. A path graph is basically an unfolding of the DMRM – while keeping track of the accumulated reward – and thus may be infinite. Since we are interested in paths of a certain length, viz. n, we "cut off" the unfolding at depth n. Formally, we consider the sets $V_0, \ldots, V_n$, where $V_i \subseteq V$ is the set of all vertices (of the above form) that can be reached in exactly i steps. Thus, for $v = (k, S_k) \in V_n$ with $S_k = \{ (s_1, p_1), \ldots, (s_m, p_m) \}$ we have that $\sum_i p_i$ equals the probability to gain k reward in n transitions when starting in state s_0. Figure 1 presents the pseudo-code of a variation of the path graph generation algorithm [16]. For the sake of simplicity, we let V_i be a set of naturals such that if $m \in V_i$ then there is a vertex $v = (m, S_m) \in V_i$.

Example 3. The path graph for our running example DMRM for three steps ($n = 3$), starting from state s_1 is:



V_0	S_0	$(s_1, 1)$
V_1	S_2	$(s_1, 0.2), (s_2, 0.5), (s_3, 0.3)$
V_2	S_4	$(s_1, 0.04), (s_2, 0.1), (s_3, 0.06)$
	S_5	$(s_4, 0.45), (s_3, 0.05)$
	S_2	$(s_1, 0.12), (s_2, 0.09), (s_3, 0.09)$
V_3	S_6	$(s_1, 0.008), (s_2, 0.02), (s_3, 0.012)$
	S_7	$(s_4, 0.27), (s_3, 0.01), (s_2, 0.27)$
	S_4	$(s_3, 0.054), (s_2, 0.078), (s_1, 0.048)$
	S_5	$(s_3, 0.024), (s_2, 0.015), (s_1, 0.02),$ $(s_4, 0.081)$
	S_2	$(s_1, 0.036), (s_2, 0.027), (s_3, 0.027)$

Time-bounded transient rewards: fix-point characterization. Verification algorithms for until-formulae (in CTL and PCTL) are inspired by a fix-point characterization [9, 11]. Checking the bounded until-operator in PRCTL amounts to computing the least solution of the following set of equations: $\mathit{Prob}(s, \Phi\, \mathcal{U}_J^N\, \Psi)$ equals 1 if $s \in \mathit{Sat}(\Psi)$ and $0 \in N$ and $0 \in J$,

$$\mathit{Prob}(s, \Phi\, \mathcal{U}_J^N\, \Psi) = \sum\nolimits_{s' \in S} \mathbf{P}(s, s') \cdot \mathit{Prob}(s', \Phi\, \mathcal{U}_{J \ominus \rho(s)}^{N \ominus 1}\, \Psi) \qquad (4)$$

if $s \in \mathit{Sat}(\Phi)$, $\sup N > 0$, and $\rho(s) \geqslant \sup J > 0$, and equals 0 otherwise. Here, $N \ominus n = \{\, m{-}n \mid m \in N, m \geqslant n \,\}$ and $J \ominus r = \{\, j{-}r \mid j \in J, j \geqslant r \,\}$ for some $r \in \mathrm{I\!R}$. Stated in words, the probability to reach a Ψ-state from s in $n \in N$ steps by earning a reward $r \in J$ equals the probability to move to a direct successor s' of s multiplied by the probability to reach a Ψ-state from s' in $N \ominus 1$ transitions by earning $J \ominus \rho(s)$ reward. Model-checking the until-operator in PRCTL thus amounts to determining the least solution of this set of linear equations.

Time-bounded transient rewards: algorithm. The algorithm to check time- and (possibly) reward-bounded until-formulas is based on the path graph generation algorithm presented above. We discuss our algorithm for the case that N is a singleton set, say $N = \{\, n \,\}$, and later discuss how this can be adapted to arbitrary sets. Suppose we have to check whether $s_0 \models \mathcal{P}_{\trianglelefteq q}(\Phi \mathcal{U}_J^n\, \Psi)$ assuming $s_0 \models \Phi$. In order to do so, the following adaptations are made to the algorithm of Figure 1:

- in selecting the successor states of s (line 8 in Figure 1) we only consider Φ-states if $i < n{-}1$ and only Ψ-states if $i = n{-}1$ (i.e., in the last step); this guarantees that all paths considered satisfy $\Phi \mathcal{U}^n\, \Psi$ and all other paths are ignored;
- it is checked whether the reward bound $\sup J$ is exceeded (line 6);
- in order to decide whether $s_0 \models \mathcal{P}_{\trianglelefteq q}(\Phi \mathcal{U}_J^n\, \Psi)$ we check (after finishing the outermost iteration) whether the total probability to end up in states with an accumulated reward in J meets the bound $\trianglelefteq q$, i.e., whether

$$\sum\nolimits_{(s,p) \in S_m \wedge m \in (V_n \cap J)} p \ \trianglelefteq\ q,$$

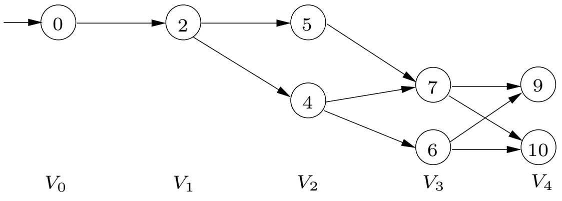
This requires an iteration over all vertices in $V_n \cap J$.

```
$V_0 := \{0\}$;                          // only nodes with cumulative reward zero
$S_0 := \{(s_0, 1)\}$;               // state $s_0$ can be reached with probability one
for ($i := 0; i < n; i{+}{+}$)
  foreach $m \in V_i$                               // check all rewards at level $i$
    foreach $(s, p) \in S_m$
      if ($m + \rho(s) \leqslant \sup J$)                // reward bound not exceeded?
      then $m' := m + \rho(s)$;                           // new cumulative reward
        foreach $s'$ with $\mathbf{P}(s, s') > 0$        // all direct successors of $s$
          if ($i < n{-}1 \wedge s' \in Sat(\Phi)$) $\vee$ ($i = n{-}1 \wedge s' \in Sat(\Psi)$)
            if $m' \notin V_{i+1}$            // encountering fresh reward $m'$
            then $V_{i+1} := V_{i+1} \cup \{m'\}$;          // add new vertex
              $S_{m'} := \{(s', p{\cdot}\mathbf{P}(s, s'))\}$;
            elseif $(s', p') \in S_{m'}$      // shared node encountered?
              then $p' := p' + p{\cdot}\mathbf{P}(s, s')$;
            else $S_{m'} := S_{m'} \cup \{(s', p{\cdot}\mathbf{P}(s, s'))\}$;
            endif;
          endif;
        endforeach;
      endif;
    endforeach;
  endforeach;
endfor;
$pr := 0$;
foreach $m \in (V_n \cap J)$
  foreach $(s, p) \in S_m$ $pr := pr + p$; endforeach;
endforeach;
return $pr \trianglelefteq q$.
```

Fig. 2. Checking whether $s_0 \models \mathcal{P}_{\trianglelefteq q}(\Phi\,\mathcal{U}^n_J\,\Psi)$

The resulting algorithm is presented in Figure 2. Note that the original path graph generation algorithm by Qureshi and Sanders [16] is obtained by checking the PRCTL formula $\mathrm{tt}\,\mathcal{U}^n_k\,\mathrm{tt}$. The following optimization can be made. For checking properties with lowerbounds on the required probabilities (i.e., $\trianglelefteq \in \{\geqslant >\}$), the computation can be terminated as soon as the total probability of all vertices at level i (with $i < n$) is less than q (or at most q, respectively). In that case, it is certain that the PRCTL-formula is refuted – as the total probability will not further increase by going from level i to $i{+}1$.

Example 4. Consider the formula $\mathcal{P}_{\geqslant 0.3}(a\,\mathcal{U}^4_{[6,10]}\,c)$ for state s_1. Stated in words, we want to check whether the probability to reach a c-state via an a-path (a path only consisting of a-states) in exactly 4 steps while earning a reward between 6 and 10 exceeds 0.3. Note that the path graph for $n = 4$ is the extension of the path graph of the previous example with the level V_4. The pruned path graph that is obtained after running our adapted path graph generation algorithm is:



V_0	S_0	$(s_1, 1)$
V_1	S_2	$(s_1, 0.2), (s_2, 0.5)$
V_2	S_4	$(s_1, 0.04), (s_2, 0.1)$
	S_5	$(s_4, 0.45)$
V_3	S_6	$(s_1, 0.008), (s_2, 0.02)$
	S_7	$(s_4, 0.36), (s_2, 0.18)$
V_4	S_9	$(s_4, 0.234)$
	S_{10}	$(s_4, 0.162)$

As the sum of the probabilities in the vertices of V_n (0.234+0.162) exceeds 0.3 we conclude that $s_1 \models \mathcal{P}_{\geqslant 0.3}(a\,\mathcal{U}^4_{[6,10]}\ c)$.

Time intervals. The next question is how we can adapt the algorithm if the number of transitions is not fixed (like n above) but an interval, i.e., $N = [n, n']$ with $0 \leqslant n \leqslant n'$. Obviously, in the worst case we have to generate all levels of the path graph from 0 to n'. Since the above algorithm does not keep track of probabilities achieved in earlier levels (but only in the last two levels), a new variable is introduced to accumulate these probabilities from level n to n'. Furthermore, we cut-off all transitions emanating from a node for which the formula under consideration becomes true. Thus, if during model-checking $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^N_J\ \Psi)$ we encounter that $\Phi\,\mathcal{U}^n_J\ \Psi$ is valid for a generated path from s_0 to s, no further investigation of the sub-tree starting in s is done as all such paths have the path $s_0, \ldots, s$ as prefix.

Unbounded time case. For model checking an until-formula with an unbounded time (or reward) interval, the algorithm in Fig. 2 is not always terminating. To solve this problem we do the following. If the DMRM contains a strongly connected component (SCC) B with only Φ-states having reward 0, we transform (as a preprocessing step) the DMRM into an equivalent DMRM that does not have such SCCs. This basically amounts to compute $\mathit{Prob}(s, \Diamond\neg B)$ for each possible entrance state s of B. This amounts to solving a system of linear equations [14].

Multiple rewards. The previous algorithm can easily be extended in order to deal with DMRMs with more than one reward structure. Suppose we have k reward structures and we are about to check whether $s_0 \models \mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^N_{J_1,\ldots,J_k}\ \Psi)$. In this case, vertices in the path graph are tuples $(l_1, \ldots, l_k, S_l)$ with $S_l = \{\,(s_1, p_1), \ldots (s_m, p_m)\,\}$ as before that are to be interpreted as follows: starting from state s_0 each state s_i $(0 < i \leqslant m)$ can be reached with probability p_i while having earned reward l_j according to reward structure ρ_j $(0 < j \leqslant k)$. The algorithm is now obtained by interpreting m and m' as k-dimensional vectors and interpreting the statement in which these variables occur accordingly. For instance, $m + \rho(s) \leqslant \sup J$ should now be read as $\forall 0 < j \leqslant k.\, m_j + \rho_j(s) \leqslant \sup J_j$. Note that the time complexity of the algorithm is increased by a factor k.

Reward measures. The reward-operators $\mathcal{C}$, $\mathcal{E}$, and $\mathcal{Y}$ are verified as follows. In order to decide whether $s \in \mathit{Sat}(\mathcal{C}^n_J(\Phi))$ we first determine the set $\mathit{Sat}(\Phi)$, i.e., the states that fulfill Φ, and then sum the instantaneous rewards in these states at epoch n (when starting in s) – which boils down to a transient analysis of the underlying DTMC – and check whether this sum lies in J. To check whether

$s \in Sat(\mathcal{E}_J(\varPhi))$, recursively $Sat(\varPhi)$ is computed. A slight generalisation of [14, Theorem 3.23] now yields that

$$s \in Sat(\mathcal{E}_J(\varPhi)) \text{ if and only if } \sum_{s' \in Sat(\varPhi)} \pi(s,s')\cdot\rho(s') \in J$$

This thus boils down to solving a system of linear equations. For $\mathcal{E}_J^n(\varPhi))$ we have:

$$s \in Sat(\mathcal{E}_J^n(\varPhi)) \text{ if and only if } \sum_{s' \in Sat(\varPhi)} \pi^*(s,s',n)\cdot\rho(s') \in J$$

where $\pi^*(s,s',n) = \frac{1}{n+1}\sum_{i=0}^{n} \pi(s,s',i)$. Finally, checking the $\mathcal{Y}$-operator amounts to solving a system of linear equations (again). The quantity $y(s, Sat(\varPhi), n)$ is characterized as the smallest solution of the following system of linear equations:

$$E(s,n) = \begin{cases} 0 & \text{if } n = 0 \\ \rho(s) + \sum_{s' \in S} \mathbf{P}(s,s')\cdot E(s',n-1) & \text{if } s \in Sat(\varPhi) \ \wedge \ n > 0 \\ \sum_{s' \in S} \mathbf{P}(s,s')\cdot E(s',n-1) & \text{if } s \notin Sat(\varPhi) \ \wedge \ n > 0 \end{cases}$$

Complexity analysis. If real numbers are permitted as rewards, the time complexity of the algorithm (cf. Fig. 2) is exponential in $|\, S\,|$ due to the fact that all paths (of some length) may have distinct accumulated rewards. If, however, we restrict to naturals or rationals – which mostly suffices – as rewards, checking a time- and reward-bounded until-formula has a time complexity in $\mathcal{O}(n \cdot \sup J \cdot |S|^3)$.

5 Case study: the IPv4 zeroconf protocol

As a case study, we consider the IPv4 zeroconf protocol, a simple protocol proposed by the IETF [6], aimed at the self-configuration of IP network interfaces in ad-hoc networks. The probababilistic behaviour of this protocol modeled as an DMRM is adopted from [5].

The IPv4 zeroconf protocol. The IPv4 zeroconf protocol is designed for a home local network of appliances (microwave oven, laptop, VCR, DVD-player etc.) each of which supplied with a network interface to enable mutual communication. Such ad-hoc networks must be hot-pluggable and self-configuring. Among others, this means that when a new appliance (interface) is connected to the network, it must be configured with a *unique* IP address automatically. The zeroconf protocol solves this task in the following way. A host that needs to be configured randomly selects an IP address, U say, out of the 65024 available addresses and broadcasts a message (called *probe*) saying “Who is using the address U?”. If the probe is received by a host that is already using the address U, it replies by a message indicating that U is in use. After receiving this message the host to be configured will re-start: it randomly selects a new address, broadcasts a probe, etc.

Due to message loss or a busy host, a probe or a reply message may not arrive at some (or all) other hosts. In order to increase the reliability of the protocol, a host is required to send n probes, each followed by a listening period of r time units. Therefore, the host can start using the selected IP address only after n probes have been sent and no reply has been received during $n{\cdot}r$ time units. Note that after running the protocol a host may still end up using an IP address already in use by another host, e.g., because all probes were lost. This situation, called *address collision*, is highly undesirable since it may force a host to kill active TCP/IP connections.

Modeling the protocol. The protocol behaviour of a single host is modeled by a DTMC that is adapted from [5]. The DTMC consists of $n{+}5$ states (cf. Figure 3 for $n = 4$) where n is the maximal number of probes needed (as above). The initial state is s_0 (labeled *start*). In state s_{n+4} (labeled *ok*) the host finally ends up with an unused IP address; in state s_{n+2} (labeled *error*) it ends up with an address that is already in use, i.e., an address collision. State s_i ($0 < i \leqslant n$) is reached after issuing the i-th probe. In state s_0 the host randomly chooses an IP address. With probability $q = m/65024$, where m is the number of hosts in the network when connecting the host to the network, this address is already in use. With probability $1{-}q$ the host chooses an unused address and ends up in state s_{n+3}. Then it issues $n{-}1$ probes and waits $n{\cdot}r$ time units before using this address. If the chosen IP address is already in use, state s_1 is reached. Now two situations are possible. With probability p, no reply is received during r time units (as either the probe or its reply has been lost), and a next probe is sent, resulting in state s_2. If, however, a reply has arrived in time, the host returns to the initial state and re-starts the protocol. The behaviour in state s_i ($2 \leqslant i < n$) is similar. If in state n, however, no reply has received within r time units after sending the n-th probe, an address collision occurs.

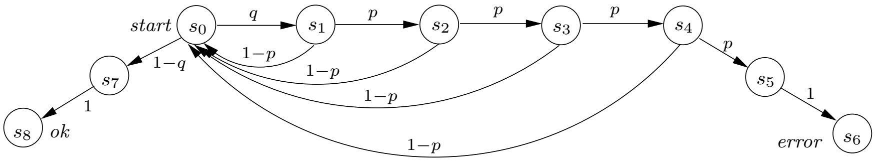


Fig. 3. DMRM model of the IPv4 zeroconf protocol (for n=4 probes).

We consider three reward structures for this model:

- The first reward assignment (denoted ρ_1) represents waiting times and is defined by: $\rho_1(s_i) = r$ for $0 < i \leqslant n$, $\rho_1(s_0) = 0$ assuming that the host randomly selects an address promptly, $\rho_1(s_{n+3}) = n{\cdot}r$, $\rho_1(s_{n+2}) = \rho_1(s_{n+4}) = 0$, and $\rho_1(s_{n+1}) = E$, where E denotes some large number that represents the highly undesirable situation of an address collision.
- The second reward assignment (denoted ρ_2) is used to keep track of the number of probes that are sent in total. It is defined by: $\rho_2(s_i) = 1$ for $0 < i \leqslant n$, $\rho_2(s_{n+3}) = n$ and 0 otherwise.
- Finally, the third reward assignment (denoted ρ_3) is used to keep track of the number of failed attempts to acquire an unused address. It is defined by: $\rho_3(s_0) = 1$ and 0 otherwise.

Properties of interest. The reward-based operators are subscripted with three reward intervals that refer to the three reward structures defined above. Intervals equal to $[0, \infty)$ are represented by a small line; if all reward intervals equal $[0, \infty)$ then the subscript is omitted. For instance, $\Diamond_{-[4,10]-}$error asserts that the protocol eventually ends up with an address collision (state *error*) where between 4 and 10 probes have been sent; no constraints are given on the number of collisions and the required time. We consider the following properties that are of interest for the IPv4 zeroconf protocol and provide their formal specification in PRCTL.

(i) "The probability to end up with an unused address is at least p'": $\mathcal{P}_{\geqslant p'}(\Diamond ok)$. As state *ok* is one of the BSCCs of the DTMC an alternative formulation would be $\mathcal{L}_{\geqslant p'}(ok)$. (Note that, in general, the formulae $\mathcal{P}(\Diamond \Phi)$ and $\mathcal{L}(\Phi)$ are not equivalent.)

(ii) "The probability to end up with an unused address within time t exceeds p'":
$$\mathcal{P}_{>p'}(\Diamond_{[0,t]--}\ ok)$$

(iii) "The probability to end up with an unused address after at most k probes exceeds p'":
$$\mathcal{P}_{>p'}(\Diamond_{-[0,k]-}\ ok)$$

(iv) "The probability to end up with an unused address within time t while having sent at most k probes exceeds p'":
$$\mathcal{P}_{>p'}(\Diamond_{[0,t][0,k]-}\ ok)$$

(v) "The probability to select more than k times a used address during the execution of the protocol is at most p'":
$$\mathcal{P}_{\leqslant p'}(\Diamond_{--[k+1,\infty)}\ start)$$
Here we use the fact that on selecting a used address the host returns to the *start* state. As the host also starts the protocol in that state, the lowerbound of the reward bound equals $k{+}1$ (rather than k).

Note that the first property does not refer to any reward and is in fact PCTL-formula that can be verified using any model checker for this logic.

Verification results. We use the following settings for the parameters: $n = 7$, $m = 10^4$, $p = 10^{-3}$, and only present results concerning the two formulas (ii) and (iv), both requiring the application of the main algorithm (Figure 2). On the top of Figure 4 different plots are shown for varying values of the bound t in formula (ii), $\mathcal{P}_{>p'}(\Diamond_{[0,t]--}\ ok)$. We display the border probability p' where the truth value of the formula turns from false to true. These boundary probabilities are very close to 1. Therefore we use a semi-logarithmic scale, and plot $1 - p'$ instead of p'. The value $1 - p'$ corresponds to the probability of not obtaining an unused address in time. As expected, increasing the waiting time r decreases the likelihood to obtain an unused address in time; but small changes to r may not induce a change of the likelihood. This phenomenon has to do with the fact that the state prior to the *ok*-state (s_7 in Figure 3) has a reward $n{\cdot}r$. For a fixed upper bound on the reward (here time t), a small increase of r does not necessarily induce less opportunities (i.e. paths) to reach the *ok*-state, explaining the displayed discontinuities. We further note that increasing the time bound t also decreases the probability for a successful address assignment. On

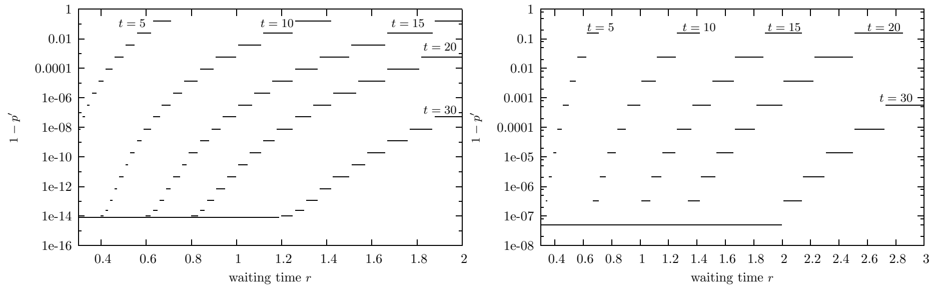


Fig. 4. Boundary probabilities for formula (ii) (left) and (iv) (right)

the bottom of Figure 4 we depict the corresponding border probabilities for formula (iv), $\mathcal{P}_{>p'}(\Diamond_{[0,t][0,k]-}\ ok)$, with the same variations on t, while $k = 15$ remains constant. We observe that the shape of the displayed plots is similar to the corresponding ones for formula (ii), but the computed probabilities p' are lower by several orders of magnitude. This is a result of the constraining effect of the second reward interval $[0, k]$, which induces that far less paths to the *ok*-state satisfy all constraints.

6 Related work

A temporal logic with accompanying efficient model-checking algorithms for CTMCs has been introduced by Baier *et al.* [3] and was later extended to reward models [2, 12]. Another notable approach to continuous time reward analysis is based on path automata [15]. Basic analysis algorithms for continuous reward models are discussed in [16] which served also as a basis for our discrete time model checking algorithms.

In the discrete time context, we are aware of the work of Voeten [20], who describes a rich assembly language for discrete reward measures. Instead of state-space based analysis algorithms, discrete event simulation is proposed to compute these measures. De Alfaro [10] also describes analysis algoritms for DMRM-like models, focussing on long-run average behaviours rather than on finite-horizon properties such as bounded until while allowing for nondeterminism. DMRM models have recently become somewhat *en vogue* as models for power-aware systems. Sokolsky *et al.* [19] have proposed a process algebra to specify power-aware systems as discrete time Markov models with nondeterminism, and have proposed model checking a μ-calculus-like logic for their analysis.

7 Conclusion

This paper developed logics and algorithms for model checking discrete time Markov reward models, providing means to formulate and efficiently check complex measures constraints – involving expected as well as accumulated rewards. We illustrated the approach by model-checking a probabilistic cost model of the IPv4 zeroconf protocol developed for ad-hoc network address assignement.

Based on the work presented here, we are further investigating efficient algorithms for continuous time reward model checking.

Acknowledgements. This work has taken place in the context of the SPACE project (SPecification-bAsed Performability ChEcking) that is supported by the Netherlands Organization for Scientific Research (NWO). We thank William H. Sanders for fruitful discussions at an early stage of this work. Henrik Bohnenkamp is thanked for discussions concerning the zeroconf protocol. Part of this work has been achieved during the Dagstuhl Seminar 03201 "Probabilistic Methods in Verification and Planning".

References

1. A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In *Computer-Aided Verification*, LNCS 1102, pp. 269–276, 1996.
2. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. On the logical characterisation of performability properties. In *Automata, Languages, and Programming*, LNCS 1853, pp. 780–792, 2000.
3. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, **29**(6):524–545, 2003.
4. M. Bernardo. An algebra-based method to associate rewards with EMPA terms. In *Automata, Languages and Programming (ICALP)*, LNCS 1256, pp. 358–368, 1997.
5. H. Bohnenkamp, P. van der Stok, H. Hermanns, and F.W. Vaandrager. Cost optimisation of the IPv4 zeroconf protocol. In *Intl. Conf. on Dependable Systems and Networks*. IEEE CS Press. 2003. (to appear).
6. S. Cheshire, B. Adoba and E. Guttman. Dynamic configuration of IPv4 link-local addresses. 2002 (draft). (available at `www.ietf.org/internet-drafts`).
7. G. Ciardo, J. Muppala and K.S. Trivedi. SPNP: Stochastic Petri Net Package. In *Proc. 3rd Int'l Workshop on Petri Nets and Performance Models*: 142–151, 1989.
8. G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. In *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601, pp. 211–227, 1999.
9. E. Clarke, O. Grumberg and D. Peled. *Model Checking*. MIT Press, 1999.
10. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *IEEE Symp. on Logic in Computer Science*, pp. 454-465, 1998.
11. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.* **6**: 512–535, 1994.
12. B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen and C. Baier. Model checking performability properties. In *Intl. Conf. on Dependable Systems and Networks*, pp. 103–113, 2002.
13. J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Van Nostrand, 1960.
14. V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.
15. W.D. Obal and W.H. Sanders. State-space support for path-based reward variables. *Performance Evaluation*, **35**(3-4): 233–251, 1999.
16. M.A. Qureshi and W.H. Sanders. Reward model solution method with impulse and rate rewards: An algorithm and numerical results. *Performance Evaluation*, **20**: 413–436, 1994.
17. A. Reibman and K. Trivedi. Transient analysis of cumulative measures of Markov model behavior. *Commun. Statist.Stochastic Models*, **5**(4):683–710, 1989.
18. R.M. Smith, K.S. Trivedi and A.V. Ramesh. Performability analysis: measures, an algorithm and a case study. *IEEE Tr. on Computers*, **37**(4):406–417, 1988.
19. O. Sokolsky, A. Philippou I. Lee, and K. Christou. Modeling and analysis of power-aware systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2619, pp. 409–425, 2003.
20. J.P.M. Voeten. Performance evaluation with temporal rewards. *Performance Evaluation*, **50**:189–218, 2002.