Symbolic and Parametric Model Checking of Discrete-Time Markov Chains

Conrado Daws [*]

Nijmegen Institute for Computing and Information Sciences,
University of Nijmegen, The Netherlands
daws at cs.ru.nl

Abstract. We present a language-theoretic approach to symbolic model checking of PCTL over discrete-time Markov chains. The probability with which a path formula is satisfied is represented by a regular expression. A recursive evaluation of the regular expression yields an exact rational value when transition probabilities are rational, and rational functions when some probabilities are left unspecified as parameters of the system. This allows for parametric model checking by evaluating the regular expression for different parameter values, for instance, to study the influence of a lossy channel in the overall reliability of a randomized protocol.

1 Introduction

In recent years, the need to formally reason about probabilistic behaviour, exhibited, for instance, by randomized algorithms, or communication protocols and computer networks with unreliable or unpredictable behaviour, has triggered research in the area of formal methods for the specification and verification of probabilistic systems. The general approach has consisted in extending those models, logics and techniques, which have proved successful in the non-probabilistic setting, with probabilities . In particular, this has lead to the theory of probabilistic model checking [8,5] of PCTL [14,1] over discrete probabilistic systems, and, in the last few years, to tools implementing it [17,21].

Discrete probabilistic systems are typically modelled by an extension of transition systems with discrete probability distributions. In this model, a set of outgoing distributions on the set of states is associated with every state. Each such distribution gives the probability with which the source state can reach some target state in one step. Models with at most one distribution per state are said to be fully probabilistic, and usually referred to as a *discrete-time Markov chains* (DTMC). Models with both nondeterministic and probabilistic choice are usually referred to as a *Markov Decision Processes* (MDP).

[*] This work was originally carried out at the Formal Methods and Tools group of the University of Twente, supported by the European Community Project IST-2001-35304 AMETIST

The logic PCTL is a version of CTL where the existential and universal quantification over paths in CTL are replaced with a *probabilistic operator* $\mathcal{P}_{\sim\lambda}(\cdot)$, where $\sim \in \{\leq, <, >, \geq\}$, and $\lambda \in [0, 1]$ is the *probability threshold*, affording the specification of properties such as "a leader will eventually be elected with probability 1" or "the chance of shutdown occurring is at most 0.001".

Probabilistic model-checking of PCTL over discrete probabilistic systems is based on the computation in every state of the probability measure of the set of paths satisfying a (path) formula. These probabilities are computed numerically by solving a system of linear equations in the case of DTMCs [14], and by solving a linear optimization problem in the case of MDPs [5].

We present a new, language-theoretic, approach for probabilistic model checking of DTMCs. Within our approach, transition probabilities are considered *letters* of an alphabet of a finite state automaton. The probability measure of a set of paths satisfying a formula is computed symbolically as a *regular expression* on that alphabet, with the standard algorithms to obtain a regular expression from a finiste state automaton. The regular expression is then evaluated to its *exact rational value* when transition probabilities are rational. Moreover, the symbolic representation of probability measures as regular expressions allows us to leave transition probabilities unspecified as formal parameters. In this case, the evaluation of a regular expression is a quotient between two polynomials on the parameters, with rational coefficients.

In this way, we can perform *parametric model checking*, e.g., check whether a formula holds for different values of the parameters, for instance, to study the influence of a lossy channel on the reliabilty of a protocol, or to obtain algebraically the value of a parameter such that the system satisfies some property. However, parametric model-checking is applicable only for formulas without nested probabilistic operators, but this does not represent a strong restriction in practice because such formulas are not needed to specify properties of interest.

The remainder of the article is organized as follows. Section 2 is a short introduction to the theory behind probabilistic model checking of PCTL over discrete-time Markov chains. Section 3 introduces our technique for symbolic model-checking of DTMCs, and we extend it to the parametric case in Section 4. We illustrate its application with two small case studies in Section 5. Finally, Section 6 concludes our presentation with a discussion of related and future work.

2 Probabilistic Model Checking

We start with a short introduction to model checking of PCTL formulas for discrete-time Markov chains. Throughout this section, we consider a given set of atomic propositions AP.

2.1 Discrete time Markov chains

A *discrete-time Markov chain* is a tuple $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$ where

- $\mathcal{S}$ is a finite set of states
- $s_0 \in \mathcal{S}$ is an initial state
- $L : \mathcal{S} \mapsto 2^{\mathsf{AP}}$ is a labelling function which gives the atomic propositions that are true in a state.
- $\mathbf{P} : \mathcal{S} \times \mathcal{S} \mapsto [0,1] \cap \mathbb{Q}$ is a probability matrix with rational values such that for all $s \in \mathcal{S}$, $\sum_{t\in\mathcal{S}} \mathbf{P}(s,t) = 1$.

The function $\mathbf{P}(s,\cdot)$ is the distribution on $\mathcal{S}$ for state s. Notice that states with no outgoing distribution can be considered by adding a self-loop with probability 1, without changing the transient and limiting probabilities of the system. The matrix entry $\mathbf{P}(s,t)$ gives the probability of making a transition from s to t. The probability of following a finite path $s_0 s_1 \ldots s_n$ is $\mathbf{P}(s_0,s_1) \cdot \mathbf{P}(s_1,s_2) \cdot \ldots \cdot \mathbf{P}(s_{n-1},s_n)$. These probabilities for finite paths give rise to a unique probability measure $\Pr_s$ on the set Path_s of infinite paths starting in state s, defined on the sets of paths having a finite common prefixe, such that $\Pr_s(\{\omega | \omega = s s_1 \ldots s_n.\omega'\}) = \mathbf{P}(s,s_1) \cdot \mathbf{P}(s_1,s_2) \cdot \ldots \cdot \mathbf{P}(s_{n-1},s_n)$ [19].

2.2 The logic PCTL

The logic PCTL [14,5] is a version of CTL where the existential and universal quantification over paths in CTL are replaced by a *probabilistic operator* $\mathcal{P}_{\sim\lambda}(\cdot)$, with $\sim \in \{\leq,<,>,\geq\}$ and $\lambda \in [0,1]$ rational is the *probability threshold*, that can be applied to a path formula. The formal syntax of PCTL formulas over AP is given by the following grammar:

$$\begin{aligned}\phi &::= \mathit{true} \,|\, a \in \mathsf{AP} \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \mathcal{P}_{\sim\lambda}(\alpha)\\ \alpha &::= \mathsf{X}\phi \,|\, \phi \mathsf{U} \phi\end{aligned}$$

2.3 Semantics and model checking

The semantics of PCTL is the same as that of CTL for the fragment where they both coincide. The semantics of the probabilistic operator is:

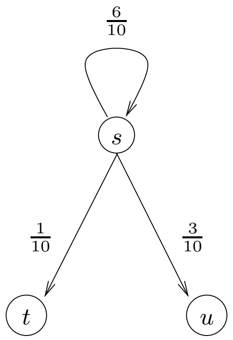
$$s \models \mathcal{P}_{\sim\lambda}(\alpha) \text{ iff } \Pr_s(\{\omega \in \mathit{Path}_s \,|\, \omega \models \alpha\}) \sim \lambda$$

meaning that the probability measure of the set of paths satisfying α is calculated and compared to the threshold λ, yielding true or false.

The model checking algorithm proceeds in the same way as for CTL, by induction on ϕ. The only difference is the evaluation of the probabilistic operator appearing in sub-formulas of the type $\mathcal{P}_{\sim\lambda}(\mathsf{X}\phi)$ and $\mathcal{P}_{\sim\lambda}(\phi_1 \mathsf{U} \phi_2)$. The example below shows the standard approach based on numerical solutions of a linear equation system. Section 3 presents our symbolic algorithm based on regular expressions.

2.4 A simple example

Let's consider the DTMC of Fig. 1 (left). The initial state s has a probabilistic branching to t with probability $\frac{1}{10}$, to u with probability $\frac{3}{10}$ and to itself with probability $\frac{6}{10}$. The probability with which t can be reached from s, denoted x_s, is the probability measure of the set of paths starting in s and satisfying the formula $\alpha = \mathit{true}\mathsf{U}t$. Its numerical value is obtained as the unique solution of the linear equation system of Fig. 1 (right), which is $x_s = \frac{1}{4}$. It follows that $s \models \mathcal{P}_{\leq \frac{1}{4}}(\alpha)$ and $s \models \mathcal{P}_{\geq \frac{1}{4}}(\alpha)$.



$$\begin{aligned} x_s &= \frac{6}{10}x_s + \frac{1}{10}x_t + \frac{3}{10}x_u \\ x_t &= 1 \\ x_u &= 0 \end{aligned}$$

Fig. 1. A simple DTMC and the corresponding linear equation system to compute the probabilities for $\mathit{true}\mathsf{U}t$ with solution $x_s = \frac{1}{4}, x_t = 1, x_u = 0$.

Tools like Prism [21] and Rapture [17] find the solution of the linear equation system using iterative methods (e.g. Jacobi, Gauss-Seidel), that numerically approximate the solution. It must be noticed that since these methods do not compute the exact solution, those tools might yield the wrong result when the solution is equal, or close enough, to the threshold of the formula, like in this example. Our symbolic approach does not suffer from this, but the same goal could be achieved using direct methods on rational numbers with arbitrary precision.

3 Symbolic model-checking of DTMCs

This section presents a language-theoretic approach to model checking of DTMCs. It is based on deriving from a DTMC a finite state automaton recognizing a language over an alphabet consisting of the *strictly positive* transition probabilities appearing in the matrix $\mathbf{P}$. The initial state of the FSA is the state on which the formula is to be checked, whilst the sets of final states and the transition function depend on the path formula under consideration. A path formula yields a regular expression that is evaluated recursively to the *exact rational value* of the probability measure of the set of paths satisfying it.

3.1 Derived FSA

We derive from a DTMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$ a finite state automaton $\mathcal{A} = (\mathcal{S}, \Sigma, \delta, \mathcal{S}_f)$ such that:

- $\mathcal{S}$ is the same set of states of $\mathcal{D}$
- $\mathcal{S}_f \subseteq \mathcal{S}$ is a subset of final states
- $\Sigma = \{(p/q) \mid \exists i, j \in \mathcal{S} \,.\, \mathbf{P}(i,j) = \frac{p}{q} > 0\}$ is the alphabet, consisting of the strictly positive entries of the probability matrix.
- $\delta : \mathcal{S} \times \Sigma \mapsto 2^{\mathcal{S}}$ is a transition function derived from $\mathbf{P}$ which associates with every pair of states and letters, a set of states such that if $\delta(s,a) = Q$ then for every $q \in Q$, $\mathbf{P}(s,q) = a$.

3.2 Evaluation of regular expressions

The set $\mathcal{R}(\Sigma)$ of regular expressions over the alphabet Σ, is the set of expressions containing the elements of Σ, and closed by union $(|)$, concatenation $(.)$ and star $(*)$. These expressions can be evaluated to a rational value, by replacing union by addition $(+)$, concatenation by multiplication $(\times)$ and star by the limit of a geometric series. Formally, the evaluation function val : $\mathcal{R}(\Sigma) \mapsto \mathbb{Q}$ is defined inductively by the rules of Table 1[1].

Table 1. Evaluation of regular expressions as rational numbers

$\mathrm{val}(p/q) = \dfrac{p}{q}$	$\mathrm{val}(x\|y) = \mathrm{val}(x) + \mathrm{val}(y)$
$\mathrm{val}(x^*) = \dfrac{1}{1 - \mathrm{val}(x)}$	$\mathrm{val}(x.y) = \mathrm{val}(x) \times \mathrm{val}(y)$

The regular language $\mathcal{L}(\mathcal{A}, s_i)$ recognized by $\mathcal{A}$ with initial state $s_i \in \mathcal{S}$, corresponds to the (possibly infinite) set Ω of finite paths from s_i to some final state in $\mathcal{S}_f$, following only transitions allowed by δ. Among all the regular expressions corresponding to this language, we consider a regular expression r computed with the *inductive* or the *state-elimination* algorithms of [15], without simplifying expressions of the type $a|a$ [2].

[1] Notice that the evaluation of x^* is not defined when x evaluates to 1, but this does not happen in the regular expressions we obtain because the final states of the FSAs we consider have no outgoing transition, thus, every cycle must be exited. The same remark applies for the parametric case.

[2] To be precise we should talk about regular expressions with multiplicities [4]. However, the regular expressions computed by the standard algorithms mentioned above without simplification, preserve the multiplicities of words, and, thus, our results hold.

The following proposition states that the evaluation of r is the probability measure in s_i of the set of paths with prefixes in Ω.

Proposition 1. *Let r be a regular expression computed for $\mathcal{L}(\mathcal{A}, s_i)$. Then,*

$$\begin{aligned}\mathrm{val}(r) = \mathrm{Pr}_{s_i}(\{\omega \in \mathit{Path}_{s_i} \mid\ & \exists\, k \geq 0 \,.\, \omega(k) \in \mathcal{S}_f, \textit{and} \\ & \forall\, l < k,\, \exists a \in \Sigma \,.\, \delta(\omega(l), a) \ni \omega(l+1)\})\end{aligned}$$

3.3 Model-checking

Let $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$ be a DTMC and $\mathcal{P}_{\sim\lambda}(\alpha)$ a PCTL formula. We charaterize the set of paths satisfying α as a regular expression on Σ. For the next operator the regular expression can be obtained directly from $\mathbf{P}$ and α. For an until formula, we derive from $\mathcal{D}$ and α a finite automaton $\mathcal{A}_\alpha$ generating the probability measure of paths satisfying α. The set of states satisfying a state formula ϕ is denoted by $[\![\phi]\!]$.

Next formulas. Let $\alpha = \mathsf{X}\phi$ be a next formula. A regular expression corresponding to the set of paths satisfying α is $|_j (p/q)_j$ such that $s_j \in [\![\phi]\!]$ and $\mathbf{P}(s_i, s_j) = \frac{p}{q}$.

Until formulas. Let $\alpha = \phi_1 \mathsf{U} \phi_2$ be an until formula. The derived finite automaton is such that the final states are those satisfying ϕ_2, and the transition function is restricted to those states satisfying $\phi_1 \wedge \neg\phi_2$. Formally:

- $\mathcal{S}_f = [\![\phi_2]\!]$
- $\delta(s, a) = \begin{cases} \emptyset & \text{if}\, s \notin [\![\phi_1]\!] \,\text{or}\, s \in [\![\phi_2]\!] \\ \{t \mid \mathbf{P}(s,t) = a\} \cap ([\![\phi_1]\!] \cup [\![\phi_2]\!]) & \text{otherwise} \end{cases}$

Model-checking. Let $\mathcal{A}_\alpha = (\mathcal{S}, \Sigma, \delta, \mathcal{S}_f)$ be the finite automaton derived from $\mathcal{D}$ and α. Then, the following proposition states that the model checking problem can be solved for a state s_i by evaluating a regular expression equivalent to the language recognized by $\mathcal{A}$ with initial state s_i.

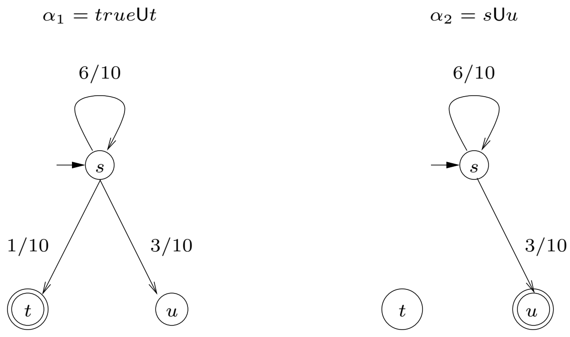
Proposition 2. *Let r be a regular expression computed for $\mathcal{L}(\mathcal{A}_{\alpha, s_i})$. Then,*

$$s_i \models \mathcal{P}_{\sim\lambda}(\alpha) \textit{ iff } \mathrm{val}(r) \sim \lambda$$

In order to model-check recursively formulas with nested probabilistic operators, we need to establish the validity of every probabilistic subformula in each state. In this case, the inductive algorithm for computing regular expressions which gives for every state a regular expression corresponding to the language it accepts, should be preferred. However, for efficiency reasons, the state-elimination algorithm is more appropriate to model-check simple formulas without nested probabilistic operators, like those usually considered in practice.

3.4 A simple symbolic example

Let's consider the DTMC of Figure 1, and the path formula $\alpha_1 = true\mathsf{U}t$ to be evaluated in state s. We derive the finite automaton depicted in Figure 2 (left) with alphabet $\Sigma = \{1/10, 3/10, 6/10\}$, initial state $s_i = s$, final states $\mathcal{S}_f = \{t\}$ and a transition function defined by $\delta(s, 6/10) = \{s\}$, $\delta(s, 1/10) = \{t\}$ and $\delta(s, 3/10) = \{u\}$.



$\alpha_1 = true\mathsf{U}t$

$\alpha_2 = s\mathsf{U}u$

Fig. 2. Finite automata for the verification of α_1 and α_2 in s.

The language recognized by this automaton corresponds to the set of paths reaching t from s. It can be described by the regular expression $r = (6/10)^*.(1/10)$ which is evaluated to $\text{val}(r) = \frac{1}{1-\frac{6}{10}} \times \frac{1}{10} = \frac{1}{4}$. It follows that $s \models \mathcal{P}_{\geq \frac{1}{4}}(\alpha_1)$ and $s \models \mathcal{P}_{\leq \frac{1}{4}}(\alpha_1)$, thus avoiding the problem arising with numerical computations. Figure 2 (right) also depicts the finite automaton derived for the evaluation of $\alpha_2 = s\mathsf{U}u$ in state s.

4 Model checking parametric DTMCs

Since regular expressions are computed formally, that is, probabilities are considered just symbols prior to evaluation, it is natural to extend our model checking technique to the case where probabilities are given as formal parameters. This makes possible to consider parametric models where some transition probabilities are left unspecified. The regular expression is in this case evaluated to a rational function, i.e., a quotient between two polynomials on the parameters, which can be manipulated algebraically for parametric analysis.

4.1 Parametric DTMCs

Let X be a set of formal parameters. A parametric DTMC is a DTMC where we extend the probability matrix to take values also in X. The formal parameters must satisfy the linear system corresponding to the stochastic condition of the probability matrix, i.e., for all $s \in \mathcal{S}$, $\sum_{t \in \mathcal{S}} \mathbf{P}(s,t) = 1$, and they must be *strictly positive* reflecting the fact that a transition between two states is present in the derived finite automaton only if it corresponds to a strictly positive probability.

A parametric DTMC gives rise to a family of DTMCs by instantiating the formal parameters to a value with an instantiation function $\kappa : \mathbb{Q}_+ \cup X \mapsto [0,1]$ such that for all $q \in \mathbb{Q}_+$, $\kappa(q) = q$, for all $x \in X$, $\kappa(x) > 0$, and for all $s \in \mathcal{S}$, $\sum_{t \in \mathcal{S}} \kappa(\mathbf{P}(s,t)) = 1$. For a parametric DTMC $\mathcal{D}_X$, and an instantiation function κ, $\kappa(\mathcal{D}_X)$ denotes the DTMC such that the probability matrix is obtained by instantiating the formal parameters.

4.2 Evaluation of the regular expression

The finite state automaton for a parameterized DTMC and a path formula is derived as in the non-parametric case. The regular expression is also evaluated recursively. In this case, the operators on regular expressions, union, concatenation and star, are replaced by the corresponding addition, multiplication and inversion for *rational functions*, that is, quotients $\frac{P(X)}{Q(X)}$ between two polynomials on X.

The evaluation function $\mathrm{val} : \mathcal{R}(\Sigma) \mapsto \langle \mathbb{Q} \rangle X \times \langle \mathbb{Q} \rangle X$ associates with a regular expression r, a pair (P_r, Q_r) of polynomials on X with coefficients in $\mathbb{Q}$, noted $\frac{P_r}{Q_r}$, defined by induction on the regular expression following the rules in Table 2.

Table 2. Evaluation of regular expressions as rational functions

$\mathrm{val}(p/q) = \dfrac{p}{q}$	$\mathrm{val}(r \mid s) = \dfrac{P_r Q_s + Q_r P_s}{Q_r Q_s}$	
$\mathrm{val}(x \in X) = x$	$\mathrm{val}(r.s) = \dfrac{P_r P_s}{Q_r Q_s}$	$\mathrm{val}(r^*) = \dfrac{Q_r}{Q_r - P_r}$

Let $\mathcal{D}_X$ be a parametric DTMC, $\mathcal{A}$ the derived FSA, and r a regular expression for its language computed with the inductive or the state-elimination algorithms. The following proposition states that the evaluation of r for any instantiation of the parameters κ, noted $\kappa(\mathrm{val}(r))$, is the probability measure in state s_i for $\kappa(\mathcal{D}_X)$, of the set of paths from s_i to some state in $\mathcal{S}_f$ following only transitions allowed by δ.

Proposition 3. *Let r be a regular expression computed for $\mathcal{L}(\mathcal{A}, s_i)$. Then,*

$$\kappa(\mathrm{val}(r)) = \mathrm{Pr}_{s_i, \kappa(\mathcal{D}_X)}(\{\omega \in \mathit{Path}_{s_i} \mid \exists\, k \geq 0 \,.\, \omega(k) \in \mathcal{S}_f, \textit{ and } \\ \forall\, l < k,\, \exists a \in \Sigma \,.\, \delta(\omega(l), a) \ni \omega(l+1)\})$$

4.3 Model checking simple PCTL formulas

Let $\mathcal{A}_\alpha = (\mathcal{S}, \Sigma, \delta, \mathcal{S}_f)$ be the finite automaton derived from $\mathcal{D}_X$ and a path formula α that does not contain nested probabilistic operators. The following proposition states that model-checking such path formulas for a state s_i in $\kappa(\mathcal{D}_X)$ consists in evaluating a regular expression equivalent to the language recognized by $\mathcal{A}_\alpha$ with initial state s_i, for the instantiation κ.

Proposition 4. *Let r be a regular expression computed for $\mathcal{L}(\mathcal{A}_{\alpha,s_i})$. Then,*

$$s_i \models_{\kappa(\mathcal{D}_X)} \mathcal{P}_{\sim\lambda}(\alpha) \textit{ iff } \kappa(\mathrm{val}(r)) \sim \lambda$$

Thus, by evaluating the corresponding regular expression, we obtain an algebraic expression of the probability measure of the sets of paths satisfying a path formula, as a rational function on the parameters. We can use the result to check whether the system satisfies a formula for different values of the parameters, without having to model check the system every time. Moreover, we can manipulate the algebraic expression in order to synthesize the values of certain parameters such that a formula is satisfied.

Parametric model-checking is however restricted to formulas without nested probabilistic operators, because a recursive evaluation of a formula is not possible in general, since the set of states satisfying a probabilistic formula is a parameterized set. This is not a strong restriction in our opinion, since in practice general formulas are not necessary to specify properties of interest. Moreover, such formulas are also problematic when using iterative numerical methods because of the propagation of the numerical error inherent to these methods.

4.4 A simple parametric example

Now let's consider that the transition probabilities of the DTMC of Figure 1 are not completely specified, and that we have the parametric DTMC depicted in Figure 3 (left), such that $\mathbf{P}(s,t) = p$, $\mathbf{P}(s,u) = 3p$ and $\mathbf{P}(s,s) = 1 - 4p$, for $0 < p < \frac{1}{4}$.

The finite state automaton derived for the verification of α_1 is depicted in Figure 3 (right). The regular expression for the language it accepts is $r = (1 - 4p)^*.p$, which is evaluated to $\mathrm{val}(r) = \frac{1}{1-(1-4p)} \times p = \frac{1}{4}$. That is, state s satisfies both $s \models \mathcal{P}_{\geq\frac{1}{4}}(\alpha_1)$ and $s \models \mathcal{P}_{\leq\frac{1}{4}}(\alpha_1)$ for any valid value of p. Notice that the evaluation of the regular expression is not defined for $p = 0$ but it is for $p = \frac{1}{4}$, hence we could relax the requirement that $\mathbf{P}(s,s)$ be strictly positive. Intuitively, this corresponds to removing the self-loop in s, which does not disconnect the graph.

5 Application

We apply our formal model checking approach to two small examples. We generate the regular expressions for the derived finite automata using the state-elimination algorithm implemented in JFLAP[13,18] and a simple script to evalute them.

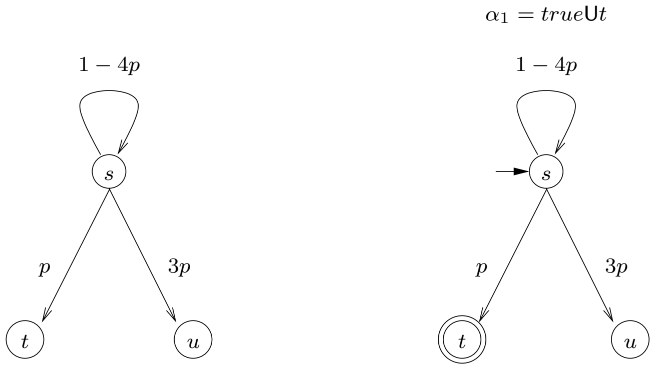


Fig. 3. Parametric DTMC and FSA for the verification of $true\mathsf{U}t$ in s.

5.1 Simulating a dice with a coin

We consider a probabilistic program due to Knuth and Yao [20], which models a fair dice using only fair coins, that has already been analyzed using a probability theory [16] for the theorem prover HOL [11,12], and the probabilistic symbolic model checker PRISM [22].

The DTMC of Figure 4 generates a uniform distribution on $\{1, \ldots, 6\}$ from a source of independent, unbiased, random bits, which can be seen as a model of a dice using a fair coin. Starting at state 0, the coin is tossed. Whenever heads appears, the system takes the upper branch and when tails appears, the lower branch. This continues until the value of the dice is decided.

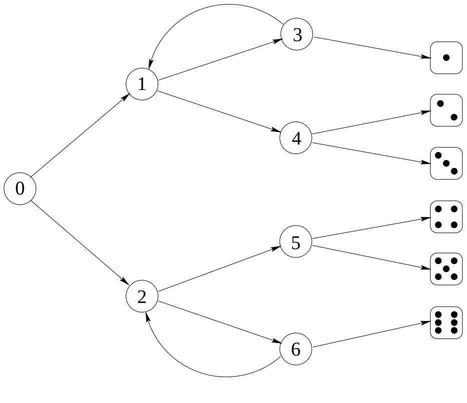


Fig. 4. Simulating a dice tossing a coin: upper branches correspond to tail, and lower branches to head.

The properties of interest of this example are that it terminates with probability 1, and that it generates the uniform distribution. Let $\boxed{i}$ be an atomic proposition characterizing the state where the value i was obtained. Let $\alpha_0 = \mathit{true}\mathsf{U}\bigvee_{i=1}^{i=6}\boxed{\mathrm{i}}$ and $\alpha_i = \mathit{true}\mathsf{U}\boxed{\mathrm{i}}$ for $i = 1 \ldots 6$. Then, the initial state s_0 must satisfy the following PCTL formulas, for $i = 1 \ldots 6$:

$$\mathcal{P}_{\geq 1}(\alpha_0), \quad \mathcal{P}_{\leq \frac{1}{6}}(\alpha_i), \quad \mathcal{P}_{\geq \frac{1}{6}}(\alpha_i)$$

Table 3 shows the results from applying our model checking approach to the dice model. For each path formula α_i, the second column gives the regular expression[3] corresponding to $\mathcal{L}(\mathcal{A}_{\alpha_i})$ and the third column gives its evaluation. The regular expression corresponding to $\mathcal{A}_{\alpha_0}$ is the union of the regular expressions for $\mathcal{A}_{\alpha_i}$, thus it is evaluated to 1. It follows that s_0 satisfies all above formulas.

Table 3. Regular expressions and evaluations for model checking the dice example.

α	$r = \mathcal{L}(\mathcal{A}_\alpha)$	val(r)
$\alpha_1, \alpha_2, \alpha_3, \alpha_6$	$(1/2).((1/2).(1/2))^*.(1/2).(1/2)$	$\frac{1}{6}$
α_4, α_5	$((1/2).(1/2)\vert(1/2).(1/2).((1/2).(1/2))^*.(1/2).(1/2)).(1/2)$	$\frac{1}{6}$

Now we show how to do parametric analysis on the dice model when we allow the use of biased coins. Let $0 < h_i < 1$ and $t_i = 1 - h_i$ be the probabilities of getting head or tail in state s_i. In order to obtain the uniform distribution, we must have $h_0 = h_4 = h_5 = \frac{1}{2}$ for symmetry reasons, hence, only states s_1, s_2, s_3 and s_6 can use biased coins. Table 4 shows the regular expressions obtained using the formal parameters h_i and t_i.

Table 4. Regular expressions for parametric analysis of the dice example.

α	$r = \mathcal{L}(\mathcal{A}_\alpha)$
α_1	$(1/2).(h_1.h_3)^*.h_1.(1 - h_3)$
α_2, α_3	$(1/2).(h_1.h_3)^*.(1 - h_1).(1/2)$
α_4, α_5	$(1/2).(1 - t_2).(1/2)\vert(1/2).t_2.(t_6.t_2)^*.t_6.(1 - t_2).(1/2)$
α_6	$(1/2).t_2.(t_6.t_2)^*.(1 - t_6)$

We will prove that the uniform distribution can not be obtained with a single biased coin. First, we must have $\mathrm{val}(r_{\alpha_1}) = \mathrm{val}(r_{\alpha_2})$. This means that $h_1(1 - h_3) = (1 - h_1)/2$ and, hence, $h_1 = 1/(3 - 2h_3)$. Thus, if s_1 and s_3 must

[3] Although the language is the same for every α_i, JFLAP can return different regular expressions.

use the same coin, we should also have $h_1 = h_3$ or $h_1 = 1 - h_3$. Both cases yield a second degree equation, with a unique solution in $]0, 1[$, $h_1 = h_3 = \frac{1}{2}$.

5.2 The IPv4 Zeroconf Protocol

We consider a simple probabilistic model of part of the IPv4 Zeroconf protocol, designed for the self-configuration of IP network interfaces. This part, modelled by the DTMC of figure 5, taken from [6], deals with the collision-avoiding mechanism of the protocol. When a fresh host joins the network, which we assume to have a fixed configuration, it selects uniformly a random IP address among the $K = 65024$ possible addresses. If there are m hosts in the network the probability of a collision is $q = m/K$.

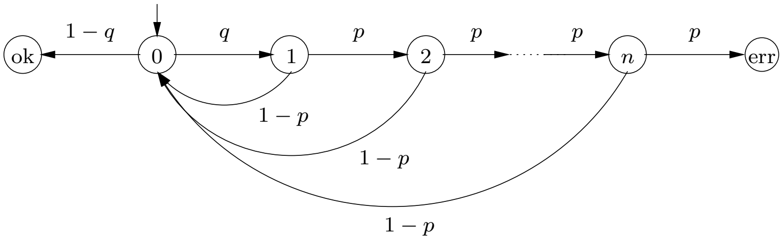


Fig. 5. Model of the zeroconf protocol

The host can select a new IP address with probability $1 - q$ and join the network. Otherwise, it tries to detect the collision by asking the other hosts whether they are using this address, and then waits for an answer. However, the new host might not receive an answer in time with probability p, in which case it repeats the question. If the answer is received in time, with probability $1-p$, then the new host can restart selecting a new address again. The protocol requires that n questions must be asked if no answer arrived. If after n tries the host didn't get an answer, then it will erroneously consider its IP as new.

We are interested in the probability with which a correct new address will be selected, that is the probability P_{ok} for reaching s_{ok} from s_0. In order to compute it, we consider the language recognized by the automaton with initial state s_0 and final state s_{ok}. The regular expression for it, is $r_{ok} = (q.(1 - p)(1|p|p.p|\ldots|p^n))^*.(1 - q)$. By evaluating this regular expression, and after a simple algebraic simplification, we obtain an analytic expression of P_{ok}:

$$P_{ok} = \frac{1-q}{1-q(1-p)\sum_{k=0}^{n} p^k} = \frac{1-q}{1-q(1-p)\frac{1-p^{n+1}}{1-p}} = \frac{1-q}{1-q(1-p^{n+1})}$$

We want the system to ensure that the new host will get a valid address with probability at least λ, i.e., that it satisfies $s_0 \models \mathcal{P}_{\geq\lambda}(true \mathsf{U} ok)$. This is equivalent to $P_{ok} \geq \lambda$. The table below (left) shows the results of parametric model checking for $\lambda = 0.999$, $n = 4$ and different values of parameters p and q. The graph below (right) plots the maximal value of q ensuring that the property holds, for $p = 0.3$ and $n = 2$, 3 and 4, in function of the probability threshold λ.

$p \setminus q$	0.1	0.2	0.4	0.7	0.8	0.9
0.1	✓	✓	✓	✓	✓	✓
0.2	✓	✓	✓	✓		
0.3	✓	✓				
0.5						
0.7						

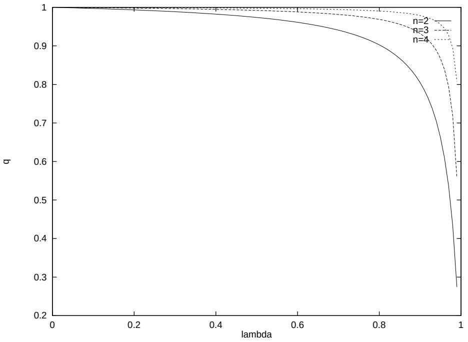


Fig. 6. Parametric analysis of the Zeroconf protocol

6 Conclusions

We presented a new language-theoretic approach to *symbolic probabilistic model checking* of PCTL over DTMCs. It is based on representing the probability measure of the set of paths satisfying a path formula as a regular expression, computed with the state elimination or inductive algorithms, for the language recognized by a finite-state automaton derived from a DTMC and a PCTL formula, where the alphabet is the set of strictly positive transition probabilities of the DTMC. When these are rational, the probability measure is evaluated to its exact rational value, whereas when transition probabilities are left unspecified as parameters, it yields a rational function on them, which can be used for parametric model checking of the system.

Although the symbolic approach cannot compete with advanced numerical methods in terms of efficiency, we believe that it has some important advantages. Besides allowing for parametric analysis as illustrated in the examples, our approach could be used to generate *"counter-examples"* violating a property, an important feature lacking in probabilistic model checking. For instance, any subterm of a regular expression whose evaluation is bigger than a threshold can be viewed as a counter-example for a property stating that the probability must be less than this threshold.

The only related result on symbolic model checking for parameterized DTMCs we are aware of is [1]. Their method consists in computing strongly connected components and then reduce a Markov chain to a DAG corresponding to its transient behaviour. Unfortunately, no algorithm is provided, and their description does not give any insight into how to obtain the probability matrix of the DAG, a step not trivial in our view. This missing step could boil down to something similar to our method, but we believe the latter to be more intuitive, precise and clear from an algorithmic point of view. Moreover, the technique of [1] cannot deal with irreducible Markov chains, that is Markov chains which are a strongly connected component.

We plan to implement our approach to model-check PCTL formulas without nested probabilistic operators for both the parametric and the non-parametric case, using the state-elimination algorithm for computing regular expressions. In our opinion, formulas with nested probabilistic operators are never or hardly necessary to specify probabilistic properties of practical interest. Moreover, these formulas are also problematic when using iterative numerical methods, since the numerical error introduced in a probabilistic subformula can yield that a path formula is satisfied with probability 1 when it is actually satisfied with probability 0, or vice-versa.

The state-elimination algorithm is the language-theoretic counterpart of Gaussian elimination for solving linear equation systems. Full PCTL could be considered in the non-parametric case, using the inductive algorithm, but since it consists in filling-in a matrix with new entries, we can expect it to have serious limitations to cope with large systems. It will be important to compare our approach based on regular expressions with symbolic methods to solve systems of linear equations based on matrix inversion or Gaussian elimination, as implemented in several computer algebra systems. In order to cope with large systems, reduction techniques based on simulation [9,10] can be applied, yielding a *symbolic* upper or lower bound for probabilistic reachability properties on a reduced state space.

As future work, we are interested in extending the method to Markov decission processes (MDPs), that is, probabilistic systems with non-determinism, necessary to model compositionally complex systems. One can see an MDP as a parametric DTMC where the non-deterministic choice is replaced by a parametric probabilistic one, such that all but one of these parameters are equal to zero, and then apply our technique. However, this will require a number of evaluations exponential in the number of non-deterministic choices in the worst case. Heuristics to reduce this number, or an alternative approach, are thus necessary. A possible solution could be to consider high-level specification languages like process algebras with iteration [2], parallel composition and communication, and to device a linearization algorithm of such specifications with respect to language equivalence (process algebras with iteration are strictly more expressive with respect to bisimulation in the presence of parallel composition [3]) without building the corresponding automata, for instance using the concept of derivatives of regular expressions [7].

Acknowledgments. We are grateful to Joost-Pieter Katoen for his comments on an early version of this paper, Wan Fokkink for his encouragement, and Ryszard Janicki for helping clarify our results.

References

1. A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, *7th International Conference On Computer Aided Verification*, volume 939 of *LNCS*, pages 155–165, Liege, Belgium, 1995. Springer Verlag.
2. J. Bergstra, W. Fokkink, and A. Ponse. Process algebra with recursive operations. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebrea*. Elsevier Science, 2001.
3. J. A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.
4. J. Berstel and C. Reutenauer. *Rational Series and Their Languages*. EATCS Monographs in Computer Science. Springer-Verlag, 1988.
5. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.
6. H. Bohnenkamp, P. van der Stok, H. Hermanns, and F. Vaandrager. Cost-optimization of the IPv4 zeroconf protocol. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN 2003)*, pages 531–540. IEEE Computer Society, June 2003.
7. J. Brzozowsky. Derivatives of regular expressions. *Journal of ACM*, 11(4), 1964.
8. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
9. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proceedings of Process Algebra and Probabilistic Methods. Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2001,* Aachen, Germany, volume 2165 of *Lecture Notes in Computer Science*, pages 29–56. Springer-Verlag, 2001.
10. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reduction and refinement strategies for probabilistic analysis. In H. Hermanns and R. Segala, editors, *Proceedings of Process Algebra and Probabilistic Methods. Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2002,* Copenhagen, Denmark, volume 2399 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
11. M. J. C. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer Academic Publishers, Boston, 1988.
12. M. J. C. Gordon and T. F. Melham. *Introduction to HOL (A theorem-proving environment for higher order logic)*. Cambridge University Press, 1993.
13. Gramond and Rodger. Using JFLAP to interact with theorems in automata theory. *SIGCSEB: SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education)*, 31, 1999.
14. H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
15. J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Longman, Reading, Massachusetts, 2000.

16. J. Hurd. *Formal Verification of Probabilistic Algorithms.* PhD thesis, University of Cambridge, 2002.
17. B. Jeannet, P. D'Argenio, and K. Larsen. Rapture: A tool for verifying Markov Decision Processes. In I. Cerna, editor, *Tools Day'02,* Brno, Czech Republic, Technical Report. Faculty of Informatics, Masaryk University Brno, 2002.
18. JFLAP (java formal languages and automata package) web page. `http://www.cs.duke.edu/~rodger/tools/jflap/`.
19. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains.* Graduate Texts in Mathematics. Springer, 2nd edition, 1976.
20. D. E. Knuth and A. C. Yao. The complexity of nonuniform random number generation. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results.* Academic Press, New York, 1976.
21. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In J. B. T. Field, P. Harrison and U. Harder, editors, *Proc. Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.
22. PRISM web page. `http://www.cs.bham.ac.uk/~dxp/prism/`.