A Recursive Probabilistic Temporal Logic

Pablo F. Castro[1,2(✉)], Cecilia Kilmurray[1,2], and Nir Piterman[3]

[1] Departamento de Computación,
FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Argentina
{pcastro,ckilmurray}@dc.exa.unrc.edu.ar
[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET),
Río Cuarto, Argentina
[3] Department of Computer Science, University of Leicester, Leicester, UK
nir.piterman@leicester.ac.uk

Abstract. In this paper we introduce recursive probabilistic computation-tree logic as a restriction of μPCTL. We introduce the logic in detail and show its usefulness for verifying systems. We illustrate this by means of some examples. Roughly speaking, we include recursive operators within PCTL, which enable one to identify repeating patterns of probability. This new feature seems in particular useful for expressing properties regarding stability of system executions; such properties are usual, for instance, in those scenarios where one is interested to verify whether the system under verification stays in, or revisits, a subset of safe states. Also, the logic makes it possible to reason about set of executions with zero measure; something no possible in related logics.

1 Introduction

The increasing role of computing systems in critical activities has led to the use of mathematical formalisms for producing error-free software as well as reducing the occurrence of faults during the execution of systems. In the case of verification of complex and large systems, automated techniques based on mathematical formalisms have been proved to be essential. One of these techniques that has received an increasing amount of attention in the last decades is *model checking*. Model checking establishes whether a system satisfies a certain property in an automated way. For that, a representation of a system M called *model* is constructed and contrasted with a property φ in temporal logic. Temporal logic can be used to express properties about concurrent and reactive systems [1].

Tools that implement model checking algorithms for various temporal logics have been applied to verify *hardware components*, *software programs*, and *network protocols* among others. Many examples of applications are reported in the literature, and we refer to [1,2] for in depth introduction to model checking.

This work was partially supported by FP7-PEOPLE-IRESES-2011 MEALS project, EPSRC EP/L007177/1 project, PICT 2013-0080 project and PICT 2012-1298 project.

M. Butler et al. (Eds.): ICFEM 2015, LNCS 9407, pp. 336–348, 2015.
DOI: 10.1007/978-3-319-25423-4_22

In the last years, several types of temporal logics incorporating probabilities into the picture have been proposed. These formalisms provide the basis to perform model checking in scenarios where probabilities are needed. This is the case, for instance, of randomized algorithms and distributed protocols. Such logics include, for example, PCTL, the probabilistic counterpart of CTL, and PCTL*, the probabilistic counterpart of CTL*, to name a few. Tools such as PRISM [3] and LiQuor [4] support probabilistic model checking. In particular, they allow to check the validity of PCTL and PCTL* formulas over Markov chain models.

A few years ago, there was a major effort that led to standardization of temporal hardware specification languages (cf. [5,6]). This effort was preceded by much research about the constructs that should (and should not) be included in such languages (e.g., [7,8]). Much care has been given to find the right balance between ease of specification, expressive power, and complexity of model checking. In this paper we would like to start a similar process for temporal logics intended for reasoning about probabilistic systems. The standard language for reasoning about such systems is PCTL, whose expressive power is very limited. Much effort has been recently dedicated to considering probabilistic μ-calculi [9,10] and automata [11]. These are very expressive, however, neglect the issues of usability and tractability.

We present a logic called RPCTL, which, as mentioned, extends PCTL with recursive calls. This logic is a fragment of μPCTL presented in [10]. The recursive operator is, in fact, a greatest fixpoint. However, introducing it through recursion takes advantage of the familiarity of the recursion concept to computer scientists. By not allowing nesting of different recursion schemes (least and greatest fixpoints) we bound the complexity of the logic. At the same time, our logic extends PCTL in expressive power and allows it to characterize repetition in the probabilistic system.[1]

We cast expressiveness results from [10] in the context of RPCTL and show that RPCTL is more expressive than PCTL. We show that the complexity of model checking matches that of PCTL and is polynomial in the underlying Markov chain. In fact, the algorithm repeatedly calls PCTL model checking.

We believe that a main application of RPCTL is the verification of properties related to *fault-tolerance*. A system is said *fault-tolerant* when it is able to continue working in an acceptable way even under the occurrence of faults. The grade of tolerance exhibited by a given system can be characterized by using collections of safe states. For instance, a system is said *fail-safe* if it stays in a set of safe states under the occurrence of faults [12], and it is classified as *non-masking* tolerant when it revisits infinitely often a set of safe or desirable states [12]. In the case of probabilistic systems (and probabilistic temporal logics), the characterization of such properties cannot be achieved in a direct way. This is mainly because the probability of a system to stay in a set of safe states is 0 when the occurrence of faults has a positive probability (i.e., the system will eventually escape from this set of "good" states with probability 1). We illustrate this

[1] We note that this extension is orthogonal to the power added by PCTL*, or other mechanisms for describing regular path properties.

point with some examples in Sect. 4. Instead of using the standard quantifier for greatest fixed point we use a syntactic sugar pointing out its recursive character, we believe this improves its usability when specifying and verifying systems, we illustrate this with two examples.

The paper is structured as follows. In Sect. 2 we introduce the basic concepts needed to tackle the ideas described in this paper. In Sect. 3 we describe the logic in detail and show how it compares to CTL and PCTL. We include two examples and show the motivation for using this logic in Sect. 4. Then, we describe the model checking algorithm in Sect. 5. Finally, we discuss some conclusions and future work.

2 Preliminaries

In this section we briefly introduce some basic concepts. A *Kripke structure* over a set AP of atomic propositions is a tuple $\langle S, \rightarrow, L, s_0 \rangle$, where S is a (finite) set of locations, $\rightarrow \subseteq S \times S$ is a relation, $L : S \rightarrow 2^{AP}$ is a labeling function and $s_0 \in S$ is an initial location. A *Markov chain* over a set AP of atomic letters is a tuple $\langle S, P, L, s_0 \rangle$, where S is a (finite) set of locations, $P : S \times S \rightarrow [0, 1]$ is a stochastic matrix, $L : S \rightarrow 2^{AP}$ is a labeling function and $s_0 \in S$ is an initial location. For a location $s \in S$ we denote by M_s the Markov chain obtained from M by setting s to the initial location.

PCTL formulas over a set AP are defined as follows:

$$\begin{aligned} J &::= \{>, \geq\} \times [0, 1] \\ \Phi &::= \top \mid \bot \mid p_i \mid \neg p_i \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}_J(\Psi) \\ \Psi &::= \mathsf{X}\Phi \mid \Phi \,\mathcal{U}\, \Phi \mid \Phi \,\mathcal{W}\, \Phi \end{aligned}$$

As usual we introduce the abbreviations F and G. The semantics and intuitions of PCTL formulas are as usual, see [2].

The logic μPCTL extends PCTL with the inclusion of fixpoint variables and least and greatest fixpoint operators [10]. We now describe the syntax and semantics of μPCTL. Let AP be a set $\{p_0, p_1, \dots\}$ of atomic propositions and let $\mathcal{V} = \{V_0, V_1, V_2, \dots\}$ be an enumerable set of variables; the sets Φ and Ψ of location and path formulas, respectively, are mutually recursively defined as follows:

$$\begin{aligned} J &::= \{>, \geq\} \times [0, 1] \\ \Phi &::= \top \mid \bot \mid p_i \mid \neg p_i \mid V_i \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}_J(\Psi) \mid \nu V_i.\Phi \mid \mu V_i.\Phi \\ \Psi &::= \mathsf{X}\Phi \mid \Phi \,\mathcal{U}\, \Phi \mid \Phi \,\mathcal{W}\, \Phi \end{aligned} \tag{1}$$

We assume that in every formula there is no repetition of bound variables; it is straightforward to see that every formula can be rewritten to satisfy this requirement. In general, we are interested in formulas in which all variables are bound.

A μPCTL formula characterizes the set of states of a Markov chain in which it holds. Consider a Markov chain $M = \langle S, P, L, s_0 \rangle$. The semantics of subformulas may depend on a valuation associating a set of states with every variable

appearing in it. Formally, a valuation is a function $\tau : \mathcal{V} \to 2^S$. We denote by $\tau[S'/V]$ the valuation such that $\tau(V) = S'$ and for every $V' \neq V$ we have $\tau[S'/V](V') = \tau(V)$.

The semantics of a formula φ, denoted $[\varphi]^M_\tau$ is defined as follows:

$$\begin{aligned}
[p_i]^M_\tau &= L(p_i) \\
[\neg p_i]^M_\tau &= S \setminus L(p_i) \\
[V_i]^M_\tau &= \tau(V_i) \\
[\varphi_1 \wedge \varphi_2]^M_\tau &= [\varphi_1]^M_\tau \cap [\varphi_2]^M_\tau \\
[\varphi_1 \vee \varphi_2]^M_\tau &= [\varphi_1]^M_\tau \cup [\varphi_2]^M_\tau \\
[\mathcal{P}_J(\Psi)]^M_\tau &= \{s \in S \mid measure_M(s, \Psi) J\} \\
[\nu V_i.\Phi]^M_\tau &= \mathsf{gfp}\{S' \subseteq S \mid S' = [\Phi]^M_{\tau[S'/V_i]}\} \\
[\mu V_i.\Phi]^M_\tau &= \mathsf{lfp}\{S' \subseteq S \mid S' = [\Phi]^M_{\tau[S'/V_i]}\}
\end{aligned}$$

We notice that 2^S is a lattice and that all operators are monotonic. It follows from the Knaster-Tarski Theorem that the greatest and least fixpoint indeed exist.

3 RPCTL

In this section we present an extension of probabilistic computation tree logic with recursive statements. We provide the fixed point operators that allow writing recursive formulas. We allow a formula to contain a recursive call by using two novel operators rec and call which are syntactic sugar for the greatest fixed point. Technically speaking, this introduces greatest fixed points in the logic, effectively making it a subset of μPCTL.

Let us start presenting the syntax of RPCTL. Let AP be a set $\{p_0, p_1, \dots\}$ of atomic propositions; the sets Φ and Ψ of location and path formulas, respectively, are mutually recursively defined as follows:

$$\begin{aligned}
J &::= \{>, \geq\} \times [0, 1] \\
\Phi &::= \top \mid \bot \mid p_i \mid \neg p_i \mid \mathsf{call}_i \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}_J(\Psi) \mid \mathsf{rec}_i.\Phi \\
\Psi &::= \mathsf{X}\Phi \mid \Phi \,\mathcal{U}\, \Phi \mid \Phi \,\mathcal{W}\, \Phi
\end{aligned}$$

In general, we are interested in formulas in which all variables are bound. Note the indexes appearing in rec and call statements, they serve mainly two purposes; firstly, they provide an enumerable collection of variables for recursion ($\mathsf{call}_0, \mathsf{call}_1, \mathsf{call}_2, \dots$); and secondly, they indicate which quantifiers bind which variables, that is, call_i is bound by rec_i, for every i.

We now describe the semantics of RPCTL. An RPCTL formula characterizes the set of states of a Markov chain in which it holds. Consider a Markov chain $M = \langle S, P, L, s_0 \rangle$. The semantics of subformulas may depend on a valuation associating a set of states with every call statement appearing in it. Formally,

a valuation is $\tau : \{\mathsf{call}_0, \mathsf{call}_1, \dots\} \to 2^S$. We denote by $\tau[S'/\mathsf{call}_i]$ the valuation such that $\tau(\mathsf{call}_i) = S'$ and for every $i \neq j$ we have $\tau[S'/\mathsf{call}_i](\mathsf{call}_j) = \tau(\mathsf{call}_j)$. The semantics of a formula φ, denoted $[\varphi]^M_\tau$ is defined as follows:

$$\begin{aligned}
[p_i]^M_\tau &= L(p_i) \\
[\neg p_i]^M_\tau &= S \setminus L(p_i) \\
[\mathsf{call}_i]^M_\tau &= \tau(\mathsf{call}_i) \\
[\varphi_1 \wedge \varphi_2]^M_\tau &= [\varphi_1]^M_\tau \cap [\varphi_2]^M_\tau \\
[\varphi_1 \vee \varphi_2]^M_\tau &= [\varphi_1]^M_\tau \cup [\varphi_2]^M_\tau \\
[\mathcal{P}_J(\Psi)]^M_\tau &= \{s \in S \mid \mathsf{measure}_M(s, \Psi) J\} \\
[\mathsf{rec}_i.\Phi]^M_\tau &= \mathsf{gfp}\{S' \subseteq S \mid S' = [\Phi]^M_{\tau[S'/\mathsf{call}_i]}\}
\end{aligned}$$

Let us illustrate the intuition behind the operators call and rec with some examples, consider the following formula:

$$\mathsf{rec}.p \wedge \mathcal{P}_{>0}(\mathsf{X}\mathsf{call}), \tag{2}$$

where, for the sake of simplicity, we avoid the indexes in rec and call. This formula holds in a location s if p *holds in* s, *and the probability that formula* 2 *holds in next locations is greater than* 0. That is, p holds in s and s has a successor satisfying p, which has a successor satisfying p, and so on. This property is in fact equivalent to the CTL property $\mathsf{EG}p$. The formula:

$$\mathsf{rec}.\mathcal{P}_{>0.5}(\mathsf{call}\, \mathcal{U}\, p), \tag{3}$$

holds in a location s if *recursively, there is a probability of more than half to continue to locations that satisfy the same property until* p *becomes true.* That is, every location encountered on the way to the satisfaction of p has more than half of its successors satisfying the same property. As we show below this property is not expressible in either CTL or PCTL.

Intuitively, we have to treat each bound variable call_i as a new proposition. Each location labeled by one of the new propositions needs to satisfy the PCTL formula obtained from the appropriate recursive call, where calls are replaced by the corresponding formula. We provide further intuitions with some examples below.

3.1 Expressive Power

We show that RPCTL is more expressive than PCTL and incomparable with CTL.

Theorem 1. RPCTL *is more expressive than* PCTL. *There are properties expressed by* RPCTL *that are not expressible in* CTL.

Proof. We first note that PCTL is syntactically included in RPCTL. The property $\mathsf{rec}.p \wedge \mathcal{P}_{>0.5}(\mathsf{X} \wedge \mathsf{call})$ is not expressible in either PCTL or CTL [10].

We note that including existential and universal path quantification in RPCTL would not increase the complexity of model checking algorithms. This would allow us to include CTL in RPCTL, should we wish to do so.

In Sect. 4 we use properties similar to the property in the proof of Theorem 1. That is, these properties enforce a repetition of a certain pattern of probability even if that pattern occurs in a set whose measure is zero. It would be possible to show that these properties are not expressible neither in CTL nor in PCTL. It is worth noting that this "repetition" feature of RPCTL that reasons about sets of paths of measure zero is the main novelty that is afforded by RPCTL.

4 Motivating Examples

In this section we describe two examples with the aim of illustrating the use of RPCTL in practice.

4.1 A Token Ring Network

Our first example consists of a simple system composed of three connected nodes, whose activities are regulated via a token ring protocol. The three nodes are connected to each other via a network with a ring topology; in this setting, a token is passed through by the nodes in such a way of guaranteeing the access to a particular resource to the actual owner of the node, e.g., permission to send information across the network.

Let us state a few properties which might be thought of as requirements for this system. One of these properties is: *there is always exactly one node that has the token, and whenever a node hold onto a token, it eventually passes it to the next node in the ring.* A simple fault that can be conceived in this context is one in which, due to the unreliability of the medium, the token is lost while it is being sent from one node to another one, we assign some probability to the occurrence of this event. A probabilistic abstraction of this situation, including the fault detection, is depicted in Fig. 1. In this model, the proposition n_i becomes true when the token is passed to node i. While ${n_i}'$ represents the situation in which the token stays in node i, before passing to the next one. It is simple to see that the probability that a token reaches node 1 when it is sent by node 0 is $\frac{1}{2}$. Note that for the other cases similar probabilities are obtained. Simple calculations show the following: $\mathcal{P}(n_1 \dots n_2) = \mathcal{P}(n_2 \dots n_0) = \frac{1}{2}$.

It is interesting to investigate the properties that hold in the non-faulty part of the system, an example is the following one.

Example Property 1. *When no faults are observed, the token could stay in a given state or move to the next one, the probability of doing this is at least one half, and this pattern can be repeated an unbounded number of times.*

Note that in this property we have an implicit notion of stability, which in some sense characterizes the normative (or expected) behavior of the system. A natural candidate to express this property is the following formula:

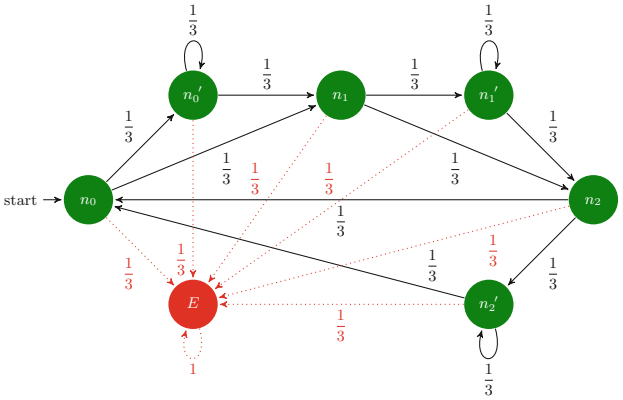

Fig. 1. A model of a token ring of nodes, where tokens can be lost.

$$\mathsf{rec.}\begin{bmatrix}(n_0 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_1 \ \wedge \mathsf{call}))) \wedge \\ (n_1 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_2 \ \wedge \mathsf{call}))) \wedge \\ (n_2 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_0 \ \wedge \mathsf{call})))\end{bmatrix} \tag{4}$$

Roughly speaking, this formula expresses that, if the token is held by node i, then the probability that the token reaches node $i+1$, and that this pattern is repeated, is one half.

If we consider the set of states that satisfy the first occurrence of call to be the state labeled by n_1, the set of states that satisfy the second appearance of call to be the state labeled by n_2, and the set of states that satisfy the last occurrence to be the state labeled by n_0, then, the PCTL property obtained by replacing bound variables by propositions denoting these sets of states, holds for states n_0, n_1, and n_2.

Let us introduce a variant of the scenario presented above. Now, when the token is held by node 2, it could stay in that state or move to node 1 or node 0, that is, now we have the possibility of returning the token to the previous node or passing it to the next one; this may be the case, for instance, in a scenario where the channel connecting node 2 with node 1 is corrupt, and the token has to be returned to the original sender. This new situation is depicted in Fig. 2. The probability of the token going from node 2 to node 0 is: $\mathcal{P}(n_2 \ldots n_0) = 0.3636$.

The formula does not hold for this last model, note that state n_2 in the model does not satisfy the subformula $(n_2 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_0 \ \wedge \mathsf{call}))$. That is, RPCTL makes possible to distinguishing different patterns of repetition.

One may try to capture this property using the following PCTL formula:

$$\varphi = \begin{bmatrix}(n_0 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_1))) \wedge \\ (n_1 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_2))) \wedge \\ (n_2 \rightarrow \mathcal{P}_{\geq\frac{1}{2}}(\mathbf{F}(n_0)))\end{bmatrix} \tag{5}$$

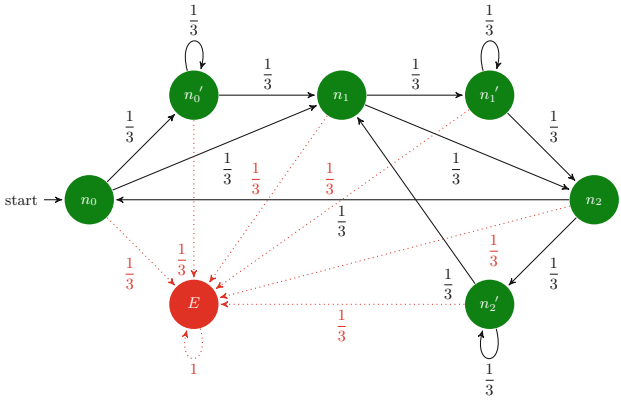

Fig. 2. Another model of a token ring of nodes, where tokens can be lost.

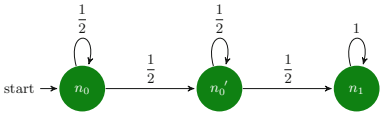

Fig. 3. Another model that satisfy φ.

In fact φ distinguishes the two models analyzed above, note that it is true in the model of Fig. 1, and false in the model presented in Fig. 2. However, notice that this PCTL formula does not capture the notion of repetition. Formula 5 is true in structures where the pattern of repetition is not available; an example is shown in Fig. 3, where we have $n_0 \vDash \varphi$.

Another possible way of capturing recursive properties like this one using PCTL is by using a globally operator of the kind: $\mathcal{P}_{>0}(\mathsf{G}\varphi)$ (where φ is the property presented above). But the set of paths that satisfy these kinds of properties has measure 0; so in this model it is equivalent to $\mathcal{P}_{=0}(\mathsf{G}\varphi)$, which prevents us from analyzing these traces.

4.2 A Mutual Exclusion Problem

Let us now consider a standard example in concurrency and fault-tolerance: the mutual exclusion problem for two processes (namely P_1 and P_2). We introduce faults in this model by allowing processes to go into an error state, in this particular case they may be down for an undetermined amount of time. The basis of this example is introduced in [13], here we add probabilities to be able to perform a quantitative analysis of this system.

In every state the probabilities of moving to its successors are distributed in a uniform way. The model that captures this scenario is shown in Fig. 4. The region M_1 (enclosed by a dashed line) contains those states that can be reached either, during the normal behavior of the system, or when P_1 is down but with

a positive probability of recovering. In order to simplify the analysis, we only consider failures in process P_1, once P_1 fails, P_2 may fail too. The transitions corresponding to the failure of P_2 followed by the failure of P_1 are similar to the ones shown in that figure. In this model, the proposition N_i becomes true when process P_i is in the *non-critical* region, propositions T_i and C_i represent the situation in which the process P_i moves into its trying section, or critical region, respectively. Finally, the proposition D_i indicates that the process P_i is down, denoting the occurrence of a fault. Note that, for the sake of clarity, we have gathered the states into regions, and transitions were added from these sets of states to failure regions. Let us state some properties of this model.

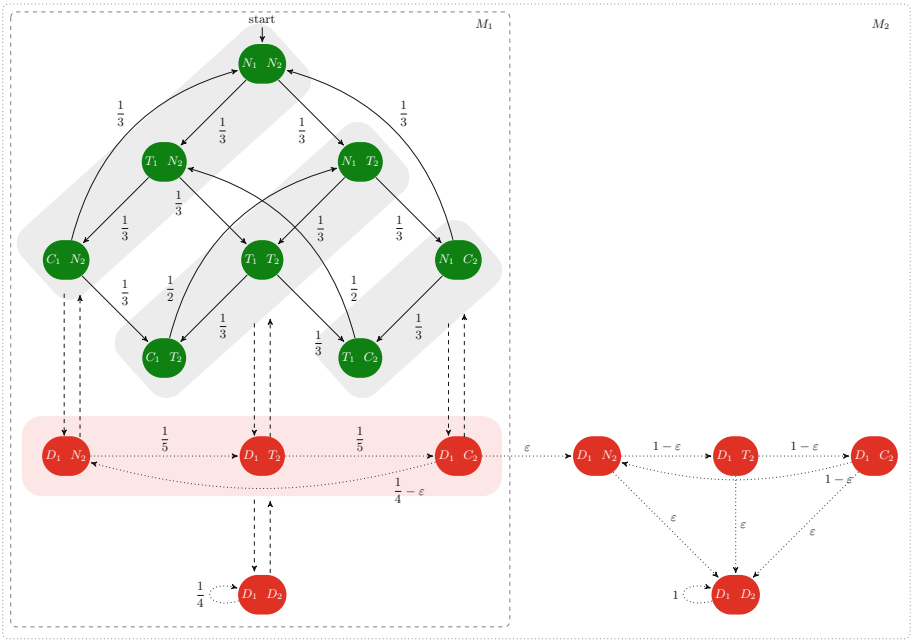

Fig. 4. Two-process mutual exclusion (Color figure online).

Example Property 2. *When there are no faults, the process P_1 stays in its safe region (N_1, T_1 or C_1) with probability greater than or equal to one half.*

This property can be expressed in RPCTL using the following formula:

$$\mathsf{rec.}\left[\neg D_1 \ \wedge \ \mathcal{P}_{\geq \frac{1}{2}}(\mathsf{X}\ \mathsf{call})\right]$$

Observe that this formula characterizes the idea of staying in a safe set of states, we can think of this collection of states as representing the normal behavior of

the system (i.e., the green states in Fig. 4), the recursive part of the formula expresses that those states will be revisited with certain probability.

As we explain in Example 1, these kinds of properties cannot be captured in PCTL. One could mix CTL and PCTL operators to express properties of traces with measure 0. Consider for instance:

$$\mathcal{P}_{\geq \frac{1}{2}}[(N_1 \vee T_1 \vee C_1)\, \mathcal{U}\, D_1] \quad \wedge \quad \mathsf{EG}(N_1 \vee T_1 \vee C_1) \tag{6}$$

This formula says that the probability of keeping the system in the safe zone until the process is down is at least to $\frac{1}{2}$, while the CTL subformula says that there exists an execution where the process is always up. However, note that the probability of repeating that pattern is not reflected in this formula.

Furthermore, we spice up this model with the possibility that process P_1 stays down forever, this new scenario is also depicted in Fig. 4, the region labeled M_2 represents a collection of states where process P_1 cannot recover from failures. Note that, in this modified model, we have a probability $0 < \varepsilon \leq 0.05$ representing the chance that P_i stays down forever. Another desirable property of this model is the following one.

Example Property 3. *The probability that the process P_1 is down with the possibility of getting up at some point is greater than or equal to one fifth.*

This property can be characterized with the following RPCTL formula:

$$\mathsf{rec.}\ \Big[D_1 \ \wedge\ \mathcal{P}_{\geq(1-\varepsilon)}(\mathsf{F}\neg D_1) \ \wedge\ \mathcal{P}_{\geq \frac{1}{5}}(\mathsf{X}\ \mathsf{call})\Big] \tag{7}$$

Intuitively, this formula characterizes the region of the system where the process P_1 fails, but there is a positive probability of returning to the safe zone. Graphically, it represents the notion of staying and revisiting infinitely often the set of red states in the M_1 part of the model, shown in Fig. 4.

Finally, a key feature of this model is the possibility of moving between normal regions (when no faults are present) and the idea that this behavior can be repeated an unbounded number of times with a probability greater than or equal to $\frac{1}{3}$.

Example Property 4. *When there are no faults, the probability that the process P_2 moves to the next region is greater or equal to one third.*

This property is expressed by the following RPCTL formula

$$\mathsf{rec.}\ \begin{bmatrix}(N_2 \rightarrow [\mathcal{P}_{\geq \frac{1}{3}}(\mathsf{X}T_2) \wedge \mathcal{P}_{\geq \frac{1}{3}}(\mathsf{X}\ \mathsf{call})]) \ \wedge \\ (T_2 \rightarrow [\mathcal{P}_{\geq \frac{1}{3}}(\mathsf{X}C_2) \wedge \mathcal{P}_{\geq \frac{1}{3}}(\mathsf{X}\ \mathsf{call})]) \ \wedge \\ (C_2 \rightarrow [\mathcal{P}_{\geq \frac{1}{3}}(\mathsf{X}N_2) \wedge \mathcal{P}_{\geq \frac{1}{3}}(\mathsf{X}\ \mathsf{call})])\end{bmatrix} \tag{8}$$

Algorithm 1. Algorithm for RPCTL Model Checking.

```
let ∀ i . W_i = ∅;
let ∀ i . S_i = S;
do {
   let ∀ i . W_i = S_i;
   let ∀ i . S_i =
      { s | M(S_1, ..., S_n), s ⊨ rec_i . φ_i( call_j ← c_j | j ∈ [1..n]) };
   }
} while (∃ i . S_i ≠ W_i);
if (M(S_1 , ..., S_n), s ⊨ φ((rec_j . φ_j) ← c_j|j ∈ [1..n])) print ``Yes!'';
else print ``No!'';
```

5 Model Checking

In this section we consider model checking of RPCTL. We give an algorithm for model checking RPCTL on finite-state Markov chains that is polynomial in the Markov chain and the size of the formula. The algorithm is the restriction of the algorithm in [10] to the usage of just greatest fixpoints. We include it here for the sake of completeness.

Consider a Markov chain $M = \langle S, P, L, s_0 \rangle$ and an RPCTL formula φ. Suppose that the set of calls appearing in φ is $\{\mathsf{call}_1, \ldots, \mathsf{call}_n\}$, and let $\{S_1, \ldots S_n\}$ be sets of states of M. That is, for every $1 \leq i \leq n$ we have $S_i \subseteq S$. We denote by $M(S_1, \ldots, S_n)$ the structure over $AP \cup \{c_1, \ldots, c_n\}$ obtained from M by setting $L(c_i) = S_i$. For the formula $\mathsf{rec}_i.\varphi_i$, let $\mathsf{rec}_i.\varphi_i(\mathsf{call}_j \leftarrow c_j | j \in [1..n])$ be the formula obtained from φ_i, where every reference to call_j is replaced by c_j. Finally, let $\varphi(\mathsf{rec}_j.\varphi_j \leftarrow c_j | j \in [1..n])$ denote the formula obtained from φ by replacing every occurrence of $\mathsf{rec}_j.\varphi_j$ by c_j. Then, Algorithm 1 computes whether a state s of M satisfies φ. The algorithm calls PCTL model checking as a subroutine.

Theorem 2. *For a* RPCTL *formula ϕ, Algorithm 1 answers "yes" iff $s \in [\phi]_\tau^M$, where τ is an arbitrary valuation.*

Proof. The proof follows from the approximation of greatest fixpoints. The algorithm computes the greatest fixpoints by initializing their approximation by the set of all states and removing all states that cannot satisfy the obligation. When the fixpoint terminates the sets are the actual fixpoints.

Theorem 3. *Algorithm 1 can be computed in time polynomial in the size of M and φ.*

Proof. We rely on the fact that PCTL model checking is polynomial both in the size of the formula and in the size of the model. We note that the propositions c_j appear positively in $\varphi(\mathsf{rec}_j.\varphi_j \leftarrow c_j)$ and in $\mathsf{rec}_i.\varphi_i(\mathsf{call} \leftarrow c_j)$ Let ψ be one of these formulas. From monotonicity it follows that if $S_i \subseteq S_i'$ then $\{s \mid M(S_1, \ldots, S_i, \ldots, S_n), s \models \psi\} \subseteq \{s \mid M(S_1, \ldots, S_i', \ldots, S_n), s \models \psi\}$. Hence,

the sets S_i are monotonically decreasing. If follows that after at most $n \cdot |S|$ iterations through the loop the loop exits.

We conclude that the algorithm calls a polynomial number of times the model checking procedure for PCTL and is polynomial.

6 Related Work

Over the years there have been several suggestions of probabilistic μ-calculi. Notably, the work of Huth and Kwiatkowska [14] and McIver and Morgan [15] both suggest quantitative μ-calculi that replace the Boolean interpretation of the classical μ-calculus with a quantitative interpretation. That is, the semantics of a formula instead of being a set of locations is a function associating a value with each location. These logics, however, fail to capture PCTL and do not have a means to get formulas back to the Boolean domain. Mio [9] extends these logics with different interpretations of quantitative conjunction. In order to reason about different types of conjunction he introduces parity games with independent products and tree games. Mio's quantitative μ-calculus does capture PCTL and has a fragment for which certain subformulas live in the Boolean domain and others in the quantitative domain [16]. Unfortunately, the complexity of model checking for Mio's logic is very high.

In our work [10] we introduced a well behaved subset of Mio's probabilistic μ-calculus, called μ^p-calculus. Its advantage is that its model checking procedure is in NP just like the μ-calculus. We then further suggested μPCTL, that incorporates fixpoint operators in PCTL. The disadvantage, as we have learned from the μ-calculus, is that fixpoint alternation are very hard for users to understand. A fragment of μPCTL is considered also in [17].

7 Final Remarks

In this paper we have introduced RPCTL, an extension of PCTL with recursive statements. It allows to specify properties describing possible "repetitions" in the Markov chain and the probability of events occurring within these repetitions. The extra expressive power comes at a very low price as model checking for this logic is by repeated calls to a PCTL model checker.

One of the main benefits of RPCTL is that it allows one to capture properties of internal regions of models. This is useful for instance in the case of systems where it is needed to reason about the repetition of a certain pattern with a given probability. This is the case, for example, of fault-tolerant systems where one needs to reason about the pattern of faults and the probability of avoiding them or recovering from them. We have presented some examples that show the application of this logic in practice. We leave as further work the implementation of a model checker for this logic, and the investigation of more complex case studies. We also intend to consider extensions to the types of regular properties that can be included within probability quantification.

References

1. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
4. Ciesinski, F., Baier, C.: LiQuor: a tool for qualitative and quantitative linear time analysis of reactive systems. In: QEST, pp. 131–132. IEEE Computer Society (2006)
5. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Springer, New York (2006)
6. Cohen, B., Venkataramanan, S., Kumari, A., Piper, L.: System Verilog Assertions Handbook. VhdlCohen Publishing (2010)
7. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., et al.: The ForSpec temporal logic: a new temporal property-specification language. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, p. 296. Springer, Heidelberg (2002)
8. Eisner, C., Fisman, D., Havlicek, J., McIsaac, A., Campenhout, D.V.: The definition of a temporal clock operator. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 857–870. Springer, Heidelberg (2003)
9. Mio, M.: Game Semantics for Probabilistic μ-Calculi. Ph.D. thesis, University of Edinburgh (2012)
10. Castro, P.F., Kilmurray, C., Piterman, N.: Tractable probabilistic μ-calculus that expresses probabilistic temporal logics. In: 32nd International Symposium on Theoretical Aspects of Computer Science. LIPIcs, Garching, Germany, pp. 211–223 (2015)
11. Huth, M., Piterman, N., Wagner, D.: p-automata: new foundations for discrete-time probabilistic verification. Perform. Eval. **69**, 356–378 (2012)
12. Arora, A., Gouda, M.: Closure and convergence: a foundation of fault-tolerant computing. TOSEM **19**, 1015–1027 (1993)
13. Attie, P., Arora, A., Emerson, A.: Synthesis of fault-tolerant concurrent programs. TOPLAS **26**, 125–185 (2004)
14. Huth, M., Kwiatkowska, M.: Quantitative analysis and model checking. In: 12th IEEE Symposium on Logic in Computer Science, pp. 111–122. IEEE Computer Society (1997)
15. McIver, A., Morgan, C.: Results on the quantitative qmμ. ACM Trans. Comput. Log. **8** (2007)
16. Mio, M., Simpson, A.: Łukasiewicz μ-calculus. In: FICS (2013)
17. Liu, W., Song, L., Wang, J., Zhang, L.: A simple probabilistic extension of model μ-calculus. In: 23rd International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina. AAAI Press, pp. 882–888 (2015)