Probabilistic Temporal Logic or: With What Probability Will the Swedish Chef Bork the Meatballs?

Monica Dinculescu

December 17, 2007

Abstract

In this paper we motivate, define and explain a probabilistic extension to CTL*, namely pCTL*. We provide examples to prove the expressive power of the probabilistic operator, as well as demonstrate the inadequacy of CTL* in dealing with real life systems. Finally, we give a brief overview of model checking for probabilistic systems and the PRISM model checker.

1 Introduction

Model checking is an automated verification method used to determine whether certain properties are true about a system. Much work has gone into using temporal modal logics, such as LTL, CTL and CTL* to express *qualitative* properties of a system, such as whether a program eventually terminates, or whether certain safety properties are violated at any given time.

Real systems, however, are quite often characterized by nondeterministic behaviour, such as stochastic transition systems and inherent built-in uncertainty, like in the case of randomized algorithms. Taking probabilities into account in addition to the deterministic behaviour would expand the aspects of the system that can be modelled, allowing the *quantification* of unpredictable errors, checking whether the specification holds with an arbitrary probability value and within a given time limit.

For example, consider the case of a communications protocol. Even if the protocol is perfectly implemented, one must account for uncertainty outside the protocol itself, such as hardware failures, power outages and grand pianos falling down on the server room. Thus, requiring that the system satisfies "every message is sent with probability 1" is rather unrealistic. In addition, we would like to be able to quantify the expected cost of two different scenarios, and express statements regarding "the expected size of a message" or "the expected cost of unpredictable delays".

The purpose of this paper is to study the expressive power of a temporal logic that introduces these probabilistic and cost operators. We define a probabilistic model that allows us to calculate the probability of observing a certain system behaviour, and define an appropriate notion of semantics based on it. Finally, through an example, we demonstrate that CTL* is not powerful enough to express properties of even the simplest probabilistic systems and show the easiness with which pCTL* can.

This paper is based on work by Kwiatkowska et al. [KNP04]. The alert reader will notice that, in the interest of clarity, most of the notation as well as the underlying probabilistic model have been modified from the original paper. These modifications will be detailed further in the relevant sections.

2 Probabilistic model

We enrich the familiar Kripke structure used when discussing CTL* by adding probabilistic transitions between states and real values costs associated with each of these transitions.

2.1 Markov decision processes

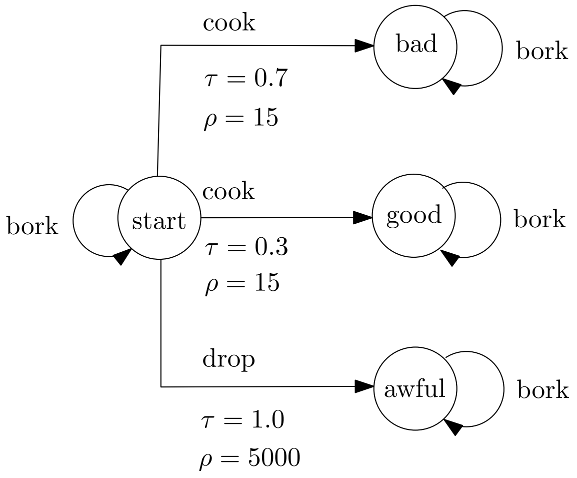
Definition 1 *A* ***Markov decision process*** *(MDP) is a quintuple*

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \tau : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1], \rho : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0})$$

where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, τ is a transition function and ρ is the cost of a transition. We will often write $\tau_a(s, s')$ for $\tau(s, s', a)$

For example, consider the model below, representing the behaviour of the swedish chef [Hen76] in a kitchen. The swedish chef can take any of three actions: `cook` a pot of meatballs, `drop` the pot on the ground, or `bork` (more precisely, "bork bork bork"). If the swedish chef chooses to make a pot of meatballs, he has a 70-30 chance of creating either bad or good meatballs (states *bad* and *good* respectively). This action uses up some ingredients and it has an associated cost of 15. Similarly, the swedish chef can drop the pot, and thus end his career as a chef (state $awful$). Because the pot is quite expensive, this transition has an associated cost of 5000. Finally, the chef can take the costless action of going "bork bork bork", after which he is none the wiser, and remains in the same state as he was before. Unlike the previous actions, the swedish chef can `bork` regardless of the state he is in.

In the interest of making the diagram less cluttered, we use τ and ρ informally, with the implicit assumption that the actions and states can be deduced from the arrows. The τ and ρ values for the `bork` action are 1 and 0 respectively, and are not displayed.



The MDP model, as defined above, is not the same as that defined in [KNP04]. The current model defines, for a given start state and action, a probability distribution over final states. The latter allows a nondeterministic choice in each state between a number of probability distributions over final states. However, this model is quickly simplified in the paper by assuming that each of these probability distributions can be uniquely labelled. We argue that this unique labelling is the same as having labelled actions in the model with a unique distribution over final states, and thus, that the simplified model used in the paper is conceptually equivalent to the model defined above.

In addition, it is worth noting that these are not the only models that can be used to define the logic pCTL*. For example, similar variants have been defined for Discrete time Markov chains (DTMCs), Petri nets and Continuous time Markov chains [CG04].

In order to be able to talk about the internal action choice, we introduce the idea of a policy, also known as an adversary.

Definition 2 *A **policy** π is a deterministic mapping from states to actions, $\pi : S \rightarrow A$*

Having the ability to specify different policies, i.e. courses of action, allows us to distinguish between different scenarios. Each scenario can be thus represented as a sequence of states and actions given by a policy. This allows us to introduce the idea of paths:

Definition 3 *A **path** under a policy π and starting in state s is defined as*

$$\alpha_s^\pi = s, a_1, s_1, a_2, s_2, \ldots$$

where $\pi(s) = a_1$ and $\forall s_i, \pi(s_i) = a_i$. Henceforth we will write α to mean α_s^π for an implicitely defined s and π

Paths can be either finite or infinite. For example, a policy in the previous MDP example, π_{bork} would say that in the *start* state, the chef should always bork. Thus, the allowed paths under this policy are of the form `[start,bork,start,bork...]`. Similarly, the policy π_{cook}, namely cooking in the *start* state would allow both the `[start, cook, good]` and the `[start, cook, bad]` paths.

Using this definition of paths, we can talk about paths starting with a pair of actions and states, or with an arbitrary sequence of such pairs:

$$s, a_1 \uparrow = \{\alpha | \alpha = s, a_1, \ldots\}$$

$$s, a_1, \ldots, a_n, s_n \uparrow = \{\alpha | \alpha = s, a_1, \ldots, a_n, s_n, \ldots\}$$

We are now ready to define a measure on paths, so that we can talk about the probability of observing a specific path from a state. Note that for the sake of brevity and clarity, this is again where our notation diverges from that used in [KNP04].

Definition 4 *The probability of observing a path under a policy π starting from a state s is denoted by Pr_s^π and can be defined as follows:*

$$Pr_s^\pi(s, a_1, s_1, a_2, s_2, \ldots a_n, s_n \uparrow) = \tau_{a_1}(s, s_1) \cdot \tau_{a_2}(s_1, s_2) \ldots \tau_{a_n}(s_{n-1}, s_n) \ldots$$

Note that the probability of observing a path from a state is 0 if anywhere in the path, the policy specifies taking an invalid action.

For example, the probability of observing the path `[start, cook, good]`, under a policy π is

$$Pr_{start}^\pi(start, cook, good) = \tau_{cook}(start, good) = 0.3$$

but

$$Pr_{good}^\pi(good, cook, good) = \tau_{cook}(good, good) = 0$$

Definition 5 *The expected cost of a path under a policy π starting from a state s is denoted by E_s^π and can be defined as follows:*

$$E_s^\pi(s, a_1, s_1, a_2, s_2, \ldots, a_n, s_n \uparrow) = \rho(s, a_1) + \rho(s_1, a_2) + \ldots + \rho(s_{n-1}, a_n) \ldots$$

Using the same example of the swedish chef, the expected cost of the path `[start, drop, awful]` is

$$E_{start}^\pi(start, drop, awful) = \rho(start, drop) = 5000$$

while

$$E_{start}^\pi(start, bork, start, cook, good) = \rho(start, bork) + \rho(start, cook) = 0 + 15 = 15$$

Finally, we conclude the definition of the MDP model with the following theorem, that allows us to reason about measurable infinite paths as well.

Theorem 1 *The set of paths given by $s, a_1, \ldots, a_n, s_n \uparrow$ generates a σ-algebra and forms a semi-ring. Thus, Pr_s^π extends uniquely to all measurable sets of paths.*

3 Probabilistic Logic Specification

3.1 Syntax

Because pCTL* is an extension of CTL*, its syntax contains both path formulas ϕ and state formulas ψ, as follows:

$$\begin{array}{lcl} \phi & := & true \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_{\bowtie c}[\psi] \mid E_{\bowtie c}[\psi] \\ \psi & := & \phi \mid \phi_1 U \phi_2 \mid \phi_1 U^{\leq k} \phi_2 \mid a \cdot \phi \mid \end{array}$$

where the $\bowtie$ operator is used to indicate any of $\leq, \geq, <, >$ and $c \in [0,1]$ is a probability value.

3.2 Semantics

In order to define the semantics of pCTL*, we must augment our original MDP model by adding a set of labels to each state. For all states s, the set $\mathcal{L}(s)$, represents the propositions that are true in that particular state.

Definition 6 *Let α be a path under a policy π, with $\alpha = s, a, s_1, s_2, a_2, \ldots$ The satisfaction relation for path formulas is defined inductively as follows:*

$$\begin{array}{lcl} \alpha \models \phi & \Leftrightarrow & s \models \phi \\ \alpha \models a \cdot \phi & \Leftrightarrow & \alpha = s, a, s_1, \ldots \ \ and \ \ s_1 \models \phi \\ \alpha \models \phi_1 U \phi_2 & \Leftrightarrow & \exists i \ \ s.t \ \ s_i \models \phi_2 \ \ and \ \ \forall j < i, s_j \models \phi_1 \\ \alpha \models \phi_1 U^{\leq k} \phi_2 & \Leftrightarrow & \exists i \leq k \ \ s.t \ \ s_i \models \phi_2 \ \ and \ \ \forall j < i, s_j \models \phi_1 \end{array}$$

Definition 7 *For any state $s \in \mathcal{S}$ we can define the satisfaction relation for state formulas as follows:*

$$\begin{array}{ll} s \models true & \forall s \in \mathcal{S} \\ s \models p & \Leftrightarrow p \in \mathcal{L}(s) \\ s \models \neg\phi & \Leftrightarrow s \not\models \phi \\ s \models \phi_1 \wedge \phi_2 & \Leftrightarrow s \models \phi_1 \wedge s \models \phi_2 \\ s \models P_{\bowtie c}[\psi] & \Leftrightarrow Pr_s^\pi(\{\alpha | \alpha \models \psi\} \bowtie c \ \ \forall \ policies \ \ \pi \\ s \models E_{\bowtie c}[\phi] & \Leftrightarrow E_s^\pi(\{\alpha | \alpha \models \phi\} \bowtie c \ \ \forall \ policies \ \ \pi \end{array}$$

Definition 8 *We can now derive the standard path formula operators:*

$$\begin{array}{ll} \Diamond\phi & = true \ \ U \ \ \phi \\ \Diamond^{\leq k}\phi & = true \ \ U^{\leq k} \ \ \phi \\ P_{\leq p}[\Box\phi] & = P_{\geq 1-p}[\Diamond\neg\phi] \\ P_{\leq p}\left[\Box^{\leq k}\phi\right] & = P_{\geq 1-p}\left[\Diamond^{\leq k}\neg\phi\right] \end{array}$$

3.3 Example

Let us now illustrate some typical sentences that are expressible in pCTL*:

1. With probability at least 0.85 I can legally reach the goal state:

$$P_{\geq 0.85}\left[\neg illegal \;\; U \;\; goal\right]$$

2. With probability at least 0.85 I can legally reach the goal state in at most 23 steps

$$P_{\geq 0.85}\left[\neg illegal \;\; U^{\leq 23} \;\; goal\right]$$

3. With probability at least 0.85 I can legally reach the goal state in at most 23 steps and stay there for exactly 17 steps

$$P_{\geq 0.85}\left[\neg illegal \;\; U^{\leq 23} \;\; (P_{=1}\left[\Box^{=17} goal\right]\right]$$

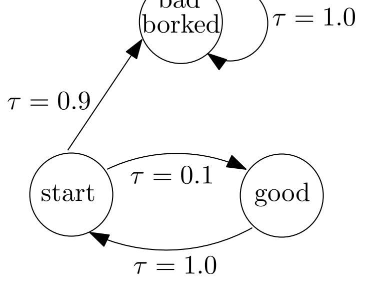
In CTL*, the most that we could express would be the formula "eventually, I can legally reach the goal state":

$$\Box(\neg illegal \;\; U \;\; goal)$$

Although pCTL* allows us to form much more expressive specifications, it has its limitations. For example, we cannot determine the actual probability of satisfying a formula, but rather, we can just place a bound on it.

3.4 Expressivity of pCTL* vs. CTL*

We will now demonstrate that CTL* is not expressive enough to reason about probabilistic systems. Let us simplify our original model to only consider the `cook` action. We will bias the distribution on the final states of this action, by specifying that the swedish chef will make good meatbals only 10% of the time. Finally, we will allow the `cook` action to be taken from the *good* state as well: if the swedish chef makes a batch of good meatballs, he can try again.



What we want to be able to check with this model is whether the start state satisfies the formula:

ϕ=`Eventually the swedish chef will bork the meatballs`

In order to do this, we add the proposition `borked` to the *bad* meatballs state. The other states do not have any propositions, and thus satisfy $\neg$`borked`.

Intuitively, we know that because there is a much higher probability to end up in the *bad* state, it should not take the swedish chef too many tries to bork the meatballs. Let us now try and express this fact in both CTL* and pCTL*.

1. In CTL*, this formula would be expressed as $\phi = \Diamond$`borked`.
   However, we know that $start \not\models \phi$ because there exists the path `[start, bad, start, bad...]` on which `borked` is never satisfied.

2. In pCTL*, the formula is $\phi = P_{\geq 1}(\Diamond$`borked`$)$.
   To check whether $s \models \phi$ we must check whether $P_{\leq 0}(\Box\neg$`borked`$) = true$. This is given by the probability of observing the infinite path `[start, bad, start, bad ...]`

   $$Pr^{\pi}_{start}(start, bad, start, bad, \ldots) = 0.1 \times 0.1 \times 0.1 \ldots = 0$$

   Thus, because the path on which we never see `borked` is never observed, that means that eventually, the swedish chef will indeed bork the meatballs.

4 Probabilistic model checking

We will now briefly summarize the model checking algorithms for pCTL* over MDPs, as presented in [KNP04].

We begin to the Sat algorithm, that takes as input a labelled MDP and a pCTL* formula ϕ, and outputs a set of states that satisfy that formula, i.e

$$Sat(\phi) = \{s \in \mathcal{S} | s \models \phi\}$$

The algorithm is almost identical to that for CTL*. The idea is that one constructs a parse tree of a formula, and labels each node of the tree with a subformula of ϕ. The algorithm can be summarized in the following way:

$$\begin{array}{ll}
Sat(true) & = S \\
Sat(false) & = \emptyset \\
Sat(p) & = \{s | p \in \mathcal{L}(s)\} \\
Sat(\neg\phi) & = S \backslash Sat(\phi) \\
Sat(\phi_1 \wedge \phi_2) & = Sat(\phi_1) \cap Sat(\phi_2) \\
Sat(P_{\bowtie c}[\psi]) & = \{s \in \mathcal{S} | Pr^{max/min}_s(\psi) \bowtie c\} \\
Sat(E_{\bowtie c}[\phi]) & = \{s \in \mathcal{S} | E^{max/min}_s(\phi) \bowtie c\}
\end{array}$$

where, if $\bowtie$ is either $>$ or $\geq$

$$\begin{array}{ll} P_s^{max}(\psi) & = \sup_\pi Pr_s^\pi(\psi) \\ E_s^{max}(\phi) & = \sup_\pi Pr_s^\pi(\phi) \end{array}$$

and analoguously for $<$ or $\leq$ and inf.

Computing the values for the first operators is identical to the model checking algorithms for CTL, as the operators have the same behaviour. The interesting cases, however, are the operators involving P and E, namely $P_{\bowtie c}[\phi]$, $E_{\bowtie c}[\phi]$, $P_{\bowtie c}\left[\phi_1 U^{\leq k}\phi_2\right]$ and $P_{\bowtie c}[\phi_1 U \phi_2]$. We will discuss the first three operators, leaving the rather complex operator, $P_{\bowtie c}[\phi_1 U \phi_2]$, to the reader.

4.1 The $P_{\bowtie c}[\phi]$ and $E_{\bowtie c}[\phi]$ operator

From the formula above we can see that we need to compute for all states the value of either $\sup_\pi Pr_s^\pi(\psi)$ or the analoguous form for inf, depending on the bowtie operator. The actual implementation of the algorithm uses linear programming and matrix multiplications. The algorithm for the $E_{\bowtie c}[\phi]$ operator is almost identical to the one described above.

4.2 The $P_{\bowtie c}\left[\phi_1 U^{\leq k}\phi_2\right]$ operator

Here, as before, we need to compute $Pr_s^{max/min}(\phi_1 U^{\leq k}\phi_2)$. This is done by dividing the set of states, $\mathcal{S}$, in three disjoint subsets:

$$\begin{array}{rcl} \mathcal{S}^{no} & = & \mathcal{S}\backslash(Sat(\phi_1) \cup Sat(\phi_2)) \\ \mathcal{S}^{yes} & = & Sat(\phi_2) \\ \mathcal{S}^{?} & = & \mathcal{S}\backslash(\mathcal{S}^{no} \cup \mathcal{S}^{yes}) \end{array}$$

For $s \in \mathcal{S}^{no}$,

$$Pr_s^{max}(\phi_1 U^{\leq k}\phi_2) = Pr_s^{min}(\phi_1 U^{\leq k}\phi_2) = 0$$

and for $s \in \mathcal{S}^{yes}$,

$$Pr_s^{max}(\phi_1 U^{\leq k}\phi_2) = Pr_s^{min}(\phi_1 U^{\leq k}\phi_2) = 1$$

For $s \in \mathcal{S}^?$, we define the probabilities recursively:
If $k = 0$, then

$$Pr_s^{max}(\phi_1 U^{\leq k}\phi_2) = Pr_s^{min}(\phi_1 U^{\leq k}\phi_2) = 0$$

If $k > 0$, then

$$Pr_s^{max}(\phi_1 U^{\leq k}\phi_2) = max_{a\in\mathcal{A}}\left\{\sum_{s'\in\mathcal{S}} \tau_a(s,s')\cdot Pr_{s'}^{max}(\phi_1 U^{\leq k-1}\phi_2)\right\}$$

and

$$Pr_s^{min}(\phi_1 U^{\leq k} \phi_2) = min_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} \tau_a(s, s') \cdot Pr_{s'}^{min}(\phi_1 U^{\leq k-1} \phi_2) \right\}$$

Thus the computation of each of the above can be carried out in k iterations. Each iteration can be though of as checking the "next" operator, and contains one matrix-vector multiplication, which we will not describe in this paper.

5 PRISM model checker

Finally, we present a series of results regarding symbolic model checkers. Such model checkers have been developed since 1999, modelling a variety of probabilistic models, and modal logics.

The most notable of such model checkers is PRISM [KNP02], a tool developed at the University of Birmingham. PRISM supports three probabilistic models: DTMCs, CTMCs, and MDPs, and verifies specifications written in both PCTL and CSL. PRISM has been successfully used to analyse probabilistic termination, performance, and quality of service properties for a range of systems, such as randomized distributed algorithms (Randomized Dining Philosophers, Randomised Mutual Exclusion, Sync/Async Leader Election, Dining Cryptographers), communication systems (Bluetooth and IEEE protocols), security systems (Crowds anonimity protocols, PIN cracking schemes) and others.

6 Conclusion

In this paper we gave an overview of pCTL*, a probabilistic temporal modal logic that extends CTL* by introducing a probabilistic and an expected cost operator. We defined a notion of paths on an MDP, thus being able to fully define the syntax and semantics of pCTL*. With the aid of an example imported from the fine kitchens of Sweden, we were able to demonstrate that CTL* is not powerful enough to reason about probabilistic systems, while pCTL* is perfectly able, and explicitly designed to.

References

[CG04] F. Ciesinski and F. Grossner. On probabilistic computation tree logic. In *Validation of Stochastic Systems*. Springer Berlin / Heidelberg, 2004. Volume 2925/2004.

[Hen76] J. Henson. The muppet show, 1976. Season 1 of 5.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, pages 200–204, 2002.

[KNP04] M. Kwiatkowska, G. Norman, and D. Parker. Modelling and verification of probabilistic systems. In P. Panangaden and F. Van Breugel, editors, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*. American Mathematical Society, 2004. Volume 23 of CRM Monograph Series.