Synthesis for PCTL in Parametric Markov Decision Processes

Ernst Moritz Hahn[1], Tingting Han[2], and Lijun Zhang[3]

[1] Saarland University, Saarbrücken, Germany
[2] Oxford University Computing Laboratory, United Kingdom
[3] DTU Informatics, Technical University of Denmark, Denmark

Abstract. In parametric Markov Decision Processes (PMDPs), transition probabilities are not fixed, but are given as functions over a set of parameters. A PMDP denotes a family of concrete MDPs. This paper studies the synthesis problem for PCTL in PMDPs: Given a specification Φ in PCTL, we synthesise the parameter valuations under which Φ is true. First, we divide the possible parameter space into hyper-rectangles. We use existing decision procedures to check whether Φ holds on each of the Markov processes represented by the hyper-rectangle. As it is normally impossible to cover the whole parameter space by hyper-rectangles, we allow a limited area to remain undecided. We also consider an extension of PCTL with reachability rewards. To demonstrate the applicability of the approach, we apply our technique on a case study, using a preliminary implementation.

1 Introduction

Markov processes [6, 26] have been applied successfully to reason about quantitative properties in networked, distributed, and recently biological systems. This paper considers *parametric* Markov processes [24], in which transition probabilities are not fixed, but depend on a set of parameters. As an example, consider a communication network with a lossy channel, where whenever a package is sent, it is received with probability x but lost with probability $1 - x$. In this context, we are interested in, for instance, determining the parametric reachability probability with respect to a given set of states. This probability is a function in x. By inserting an appropriate value for x in the function, we will obtain a concrete model without parameters. The synthesis problem asks, for example, what are the possible parameter valuations such that the reachability probability is below the a priori specified threshold.

Daws has devised a language-theoretic approach to solve the reachability problem in parametric Markov chains [11]. In this approach, the transition probabilities are considered as letters of an alphabet. Thus, the model is viewed as a finite automaton. Based on the *state elimination approach* [21], the regular expression describing the language of such an automaton is computed. In a post-processing step, this regular expression is transformed into a rational function over the parameters of the model. In previous works [17], we have improved this

method by intertwining the state elimination and the computation of the rational function. Briefly, in a state elimination step, we label the edges directly with the appropriate rational function representing the flow of probabilities. Once all states—except the initial one and the goal states—have been eliminated, we can obtain the probabilities directly from the remaining edges. This improved algorithm is implemented in our tool PARAM [16]. The tool also supports bounded reachability, relying on matrix-vector multiplication with rational function entries, and *reachability rewards* [7,13]. For the latter, we extended the model with parametric rewards assigned to both states and transitions, and considered the expected accumulated reward until a given set of states is reached.

In this paper, we extend our approach to solve the PCTL synthesis problem for parametric Markov decision processes (PMDPs). PCTL (Probabilistic CTL) [6, 19] is a probabilistic extension of the logic CTL for reasoning about properties over Markov models. In this paper, we extend the PCTL formulae with the reachability reward properties [20, 23] and can express properties like:

> *"The probability is larger than* 0.99*, that in the next step we move to a state where the accumulated reward until we are able to reach a state in which the property 'a' holds is less than* 5*."*

as $\mathcal{P}_{>0.99}(\mathcal{X}\,\mathcal{R}_{<5}(\Diamond a))$ in PCTL. We are interested in synthesising the concrete models fulfilling a given specification. Markov decision processes contain both probabilistic choices and nondeterministic choices. The notion of schedulers is used to resolve nondeterminism, leading to a parametric Markov chain. Previously [17], we considered a method for PMDPs by encoding nondeterminism in additional parameters. This method turned out to be limited by the number of nondeterministic choices, and can not be extended to treat nested properties. To handle the PCTL synthesis problem on PMDPs, we propose to divide the parameter valuations into *regions*, which are hyper-rectangles in the dimension of the number of variables. A region represents a family of concrete models. We aim at computing regions that subsume models with the same truth value of the specification. In general, it is not possible to cover the whole space completely. Thus, we stop as soon as the size of regions is below a pre-specified threshold, where it is unknown whether the specification is satisfied or not. To be on the safe side, the unknown regions are usually assumed not to fulfil the specification. To decide properties of a region of parameter valuations, we can use an approximate but fast method [18] which might derive false positive or false negative results. It can thus be used to get a quick overview for which areas the formula may hold, but should not be used if this information is critical. We can also use slower decision procedures with correctness guarantees [14, 25, 27].

In Fig. 1, we give an example for illustration. Zeroconf [9] is a protocol allowing the dynamic configuration of a network. When a new host enters a network, it randomly chooses an ID and asks the existing members of the network whether the ID is already in use. The request is conducted maximally n times, to minimise the probability of not getting an answer in an unreliable network even though the ID is used. If the host does not get an answer within n tries, it assumes the ID to

be unused. Here, we assume $n = 10$. The parameter p denotes the probability that the host gets no answer in case of a collision, and q denotes the probability that a chosen ID is already in use. We ask whether the expected number of requests till the protocol terminates (with an either unique or duplicate ID) is below 11. In PCTL (with reward extensions), this property can be expressed as $\mathcal{R}_{<11}(\Diamond \mathit{IDConfirmed})$. In Fig. 1, regions for which this holds (resp. does not hold) are given as white (resp. black) boxes, while the gray boxes are unknown regions. As we can see, an increase of p or q leads to an increase of the expected number of trials.

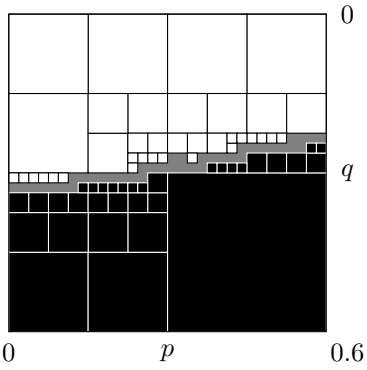

Fig. 1. Dividing the parameter space into regions in the Zeroconf example.

To the best of our knowledge, parameter synthesis for PCTL properties in PMDPs has not been handled before. The most closely related work is due to Fribourg and André [15]: For a given PMDP and an instantiation of the parameters, they compute a scheduler for this instantiation which is optimal for a certain (non-nested) property. Afterwards, they compute the set of parameter evaluations for which the scheduler is still optimal. Compared to their work, we can deal with nested formulae and do not have a fixed scheduler a priori, but use different optimising schedulers for different regions if necessary.

Organisation of the paper. In Section 2 we give some preliminaries and define the parametric models and the variant of PCTL used in this paper. Then, in Section 3, we describe our parameter synthesis algorithm. We provide experimental results in Section 4. Finally, Section 5 concludes the paper.

2 Preliminaries

In this section, we first introduce the definitions of non-parametric Markov models and the logic PCTL. Afterwards, we introduce our parametric extensions and hyper-rectangles needed later for the synthesis problem.

2.1 Non-parametric Models

Definition 1. *A* Markov chain (MC) *is a tuple $\mathcal{D} = (S, s_0, \mathbf{P}, L)$ where S is a finite set of states, s_0 is the initial state, $\mathbf{P} : S \times S \to [0, 1]$ denotes the probability matrix, where for all $s \in S$ we require that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$. Finally, $L : S \to 2^{AP}$ is a state labelling, mapping states to a subset of a given set of* atomic propositions *AP.*

Markov chains are the most basic model class. Next, we consider Markov decision processes which extend MCs by nondeterministic decisions.

Definition 2. *A* Markov decision process (MDP) *is defined as a tuple* $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ *where* S, s_0 *and* L *are as for MCs, and* Act *is a finite set of* actions. *The transition probability matrix* $\mathbf{P}$ *is a function* $\mathbf{P} : S \times Act \times S \to [0,1]$. *For all states* $s \in S$ *and actions* $\alpha \in Act$, *we require that* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$. *We also require that for each* $s \in S$ *there is at least one* $\alpha \in Act$ *with* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$.

With $Act(s) = \{\alpha \mid \sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1\}$ we specify the set of *enabled* actions of a state. The nondeterministic choices are resolved by the notion of *schedulers*. A *simple scheduler* is a function $\delta : S \to Act$ assigning one enabled action to each state. A *counting scheduler* is a function $\delta : S \times [1, n] \to Act$, for some $n \in \mathbb{N}$. Notice that for each $i \in \{1, \ldots, n\}$ we have that $\delta(\cdot, i)$ is a simple scheduler. For our purposes, simple and counting schedulers suffice. A simple scheduler induces an MC from an MDP as follows.

Definition 3. *Given an MDP* $\mathcal{M} = (S, s_0, Act, \mathbf{P}, V)$ *and a simple scheduler* δ, *the MC* induced *by* δ *is defined as* $\mathcal{M}^\delta := (S, s_0, \mathbf{P}^\delta, V)$ *where the transition matrix* $\mathbf{P}^\delta : S \times S \to [0,1]$ *is defined by* $\mathbf{P}^\delta(s, s') := \mathbf{P}(s, \delta(s), s')$.

For MDPs with exactly one enabled action for each state, there is a one-to-one correspondence to MCs, so we can consider MCs as a special case of MDPs.

When model checking PCTL formulae, we will have to consider modified versions of our models, in which certain states are made absorbing.

Definition 4. *Let* $\text{sink} : S \to \{\mathit{false}, \mathit{true}\}$ *be a function mapping states to boolean values. For the transition matrix* $\mathbf{P}$ *of an MDP, we define a transition matrix* $\mathbf{P}[\text{sink}]$ *where states* s *with* $\text{sink}(s) = \mathit{true}$ *are made* absorbing *by setting*

$$\mathbf{P}[\text{sink}](s, \alpha, s') := \begin{cases} \mathbf{P}(s, \alpha, s') & \text{if } \text{sink}(s) = \mathit{false}, \\ 1 & \text{if } \text{sink}(s) = \mathit{true} \wedge s = s', \\ 0 & \text{else.} \end{cases}$$

By skipping the action α *above we get the definition for MCs.*

We now extend our models by *rewards*, which can be interpreted as either costs or bonuses, depending on the model under consideration.

Definition 5. *A* reward structure *for an MDP with state space* S *and action set* Act *is a partial function* $\mathbf{r} : S \times Act \rightharpoonup \mathbb{R}_{\geq 0}$ *assigning a reward to each state and enabled action. For an MC, a reward structure is a function* $\mathbf{r} : S \to \mathbb{R}_{\geq 0}$ *assigning a reward to each state.*

Similar to the probability matrices, if $\mathbf{r}$ is a reward structure for an MDP and δ is a simple scheduler, we define $\mathbf{r}^\delta$ such that $\mathbf{r}^\delta(s) = \mathbf{r}(s, \delta(s))$. Given a function $\text{sink} : S \to \{\mathit{true}, \mathit{false}\}$ and a reward structure $\mathbf{r}$ for an MDP, we let $\mathbf{r}[\text{sink}](s, \alpha) = 0$ if $\text{sink}(s) = \mathit{true}$ and $\mathbf{r}[\text{sink}](s, \alpha) = \mathbf{r}(s, \alpha)$ otherwise. For an MC, we define $\mathbf{r}[\text{sink}](s) = 0$ if $\text{sink}(s) = \mathit{true}$ and $\mathbf{r}[\text{sink}](s) = \mathbf{r}(s)$ otherwise.

2.2 Probabilistic CTL

To specify properties, we consider the logic Probabilistic CTL (PCTL) [6, 19]. The syntax is given by:

$$\Phi = \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\bowtie p}(\varphi) \mid \mathcal{R}_{\bowtie m}(\Diamond\, \Phi), \qquad \varphi = \mathcal{X}\, \Phi \mid \Phi\, \mathcal{U}\, \Phi \mid \Phi\, \mathcal{U}^{\leq n}\, \Phi,$$

where $\bowtie \in \{<, \leq, \geq, >\}$, $n \in \mathbb{N}$, $p \in [0,1]$, $m \in \mathbb{R}$ and $a \in AP$. Here, Φ is a formula which has a boolean value in a state, whereas φ is interpreted on paths. PCTL can be interpreted on MDPs [6].

The truth values of *true*, a and $\wedge$ in a state are straightforward. For state s, the formula $\mathcal{P}_{\bowtie p}(\varphi)$ is fulfilled if for all schedulers the probability of paths which start in s and fulfil φ meets the bound $\bowtie p$. For $\bowtie \in \{<, \leq\}$, this is equivalent to asking whether the *maximal* probability fulfils $\bowtie p$, whereas for $\bowtie \in \{\geq, >\}$ we only need to consider the *minimal* probability.

Given a path, the *next* formula $\mathcal{X}\, \Psi$ asks whether on the second state of this path Ψ holds. The *unbounded until* formula $\Psi_1\, \mathcal{U}\, \Psi_2$ requires that a state on the path fulfils Ψ_2, and for all states on the path before that point, Ψ_1 must hold. The *bounded until* formula $\Psi_1\, \mathcal{U}^{\leq n}\, \Psi_2$ is similar, but additionally requires that Ψ_2 occurs at latest n steps after the first state of the path. The formal semantics of PCTL on MDPs has been introduced by Bianco and De Alfaro [6]. We write $\mathcal{M} \models \Phi$ if the initial state of an MDP fulfils the PCTL state formula Φ. The reachability reward formula [20, 23] $\mathcal{R}_{\bowtie m}(\Diamond\, \Psi)$ states that the expected accumulated reward until a state satisfying Ψ is reached should meet the bound $\bowtie m$. The formula holds, if under all schedulers this expectation fulfils $\bowtie m$.

2.3 Parametric Models

We fix $V = \{x_1, \ldots, x_n\}$ as the set of variables with domain $\mathbb{R}$. With each variable x, we associate a closed interval $\mathrm{range}(x) = [L_x, U_x]$ specifying which values of x are valid. An *evaluation* v is a function $v : V \to \mathbb{R}$ respecting the variable ranges. A *polynomial* g over V is a sum of monomials

$$g(x_1, \ldots, x_n) = \sum_{i_1, \ldots, i_n} a_{i_1, \ldots, i_n} x_1^{i_1} \cdots x_n^{i_n},$$

where each $i_j \in \mathbb{N}_0$ and each $a_{i_1,\ldots,i_n} \in \mathbb{R}$. A *rational function* f over a set of variables V is a fraction $f(x_1, \ldots, x_n) = \frac{g_1(x_1,\ldots,x_n)}{g_2(x_1,\ldots,x_n)}$ of two polynomials g_1, g_2 over V. Let $\mathcal{F}_V$ denote the set of rational functions from V to $\mathbb{R}$. Given $f \in \mathcal{F}_V$ and an evaluation v, we let $f\langle v \rangle := f(v(x_1), \ldots, v(x_n))$ denote the rational number obtained by substituting each occurrence of x_i with $v(x_i)$.

We now extend MCs to parametric models [11, 24]. The difference to the original model lies in the extension by parameters and the definition of the probability matrix.

Definition 6. *A* parametric Markov chain (PMC) *is defined as a tuple* $\mathcal{D} = (S, s_0, \mathbf{P}, L, V)$ *where* S, s_0 *and* L *are as in Definition 1,* $V = \{x_1, \ldots, x_n\}$ *is a finite set of parameters and* $\mathbf{P}$ *is the probability matrix* $\mathbf{P} : S \times S \to \mathcal{F}_V$.

A parameter evaluation induces a non-parametric MC from a PMC.

Definition 7. *Let $\mathcal{D} = (S, s_0, \mathbf{P}, L, V)$ be a PMC. The MC $\mathcal{D}_v$* induced *by an evaluation v is defined as $\mathcal{D}_v := (S, s_0, \mathbf{P}_v, L)$ where the transition matrix $\mathbf{P}_v : S \times S \to [0,1]$ is given by $\mathbf{P}_v(s,s') := \mathbf{P}(s,s')\langle v \rangle$ if this matrix fulfils the requirements of Definition 1.*

We already considered [16, 17] how to compute a rational function which represents the unbounded reachability probability from the initial state to a set of target states in a PMC. Evaluating this rational function with a certain parameter evaluation leads to the same result as first computing the induced PMCs and then computing the probability in this model. With a simple extension of our previous techniques, we can compute reachability values for all states of the model at the same time, which is necessary when checking nested formulae. Below we define parametric MDPs.

Definition 8. *A* parametric Markov decision process (PMDP) *is a tuple $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L, V)$ where S, s_0, L and V are as for PMCs, and Act is a finite set of actions. The transition matrix $\mathbf{P}$ is of the form $\mathbf{P} : S \times Act \times S \to \mathcal{F}_V$.*

As for PMCs, we introduce the MDP induced by a valuation function.

Definition 9. *Given a PMDP $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L, V)$ and an evaluation v, the MDP* induced *by v is defined by $\mathcal{M}_v := (S, s_0, Act, \mathbf{P}_v, L)$ where $\mathbf{P}_v : S \times Act \times S \to [0,1]$ is defined by $\mathbf{P}_v(s,\alpha,s') := \mathbf{P}(s,\alpha,s')\langle v \rangle$. For $\mathbf{P}_v$, the requirements of Definition 2 must be fulfilled.*

The notions of making a state absorbing as well as models and rewards induced by a scheduler are defined as in the non-parametric models. We allow reward structures to take rational functions as values. In turn, for an evaluation v, we define $\mathbf{r}_v$ as $\mathbf{r}_v(s,\alpha) := \mathbf{r}(s,\alpha)\langle v \rangle$ or $\mathbf{r}_v(s) := \mathbf{r}(s)\langle v \rangle$ respectively. As required by the optimality equation used in the later model checking algorithm, we assume nonnegative rewards, i.e., $\mathbf{r}_v \geq 0$, for all evaluations under consideration.

We assume that our evaluation functions fulfil the following assumption.

Assumption 1. *Let v be an evaluation function and let $\mathbf{P}$ be the probability matrix of a PMC or PMDP. Then no transition probability of $\mathbf{P}_v$ is zero or one, except this entry is zero or one for any evaluation.*

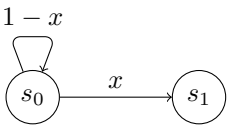

Fig. 2. Example PMC of Assumption 1.

Our assumption guarantees that the structure of the underlying graph of $\mathbf{P}$ remains unchanged from v. In other words, a transition with a parameter should not disappear (due to the null probability) no matter what value it takes. It excludes extreme cases such as when $x = 0$ or $x = 1$ (see Fig. 2). This is not a severe restriction, as such cases are seldom interesting in practice: They correspond to cases where an error happens either not at all or with certainty. These corner cases can be treated separately, with exponential blow-up in the number of variables, by fixing each such possible evaluation combinations before applying our approach.

2.4 Hyper-rectangles

A *region* is a high-dimensional rectangle $r = \bigtimes_{x \in V} [l_x, u_x]$ such that for all $x \in V$ it is $[l_x, u_x] \subseteq \text{range}(x)$. A region represents those evaluations v with $v(x) \in [l_x, u_x]$ for all $x \in V$: In this case, we write $v \in r$. We define the *centre* of a region $r = \bigtimes_{x \in V} [l_x, u_x]$ by $\text{centre}(r)(x) := \frac{l_x + u_x}{2}$ for $x \in V$. Later on, we might have to split a region r into several smaller parts, provided r is too coarse with respect to a property, i.e., we are not sure whether the property holds for all evaluations it represents. For this, we introduce the splitting function. For $A \subseteq V$, we let

$$\text{INT}_2(r, x, A) := \begin{cases} \{[l_x, u_x]\} & \text{if } x \notin A, \\ \{[l_x, \text{centre}(r)(x)], [\text{centre}(r)(x), u_x]\} & \text{if } x \in A \end{cases}$$

be the function dividing the interval $[l_x, u_x]$ (of region r) on dimension $x \in A$ into two halves. Define

$$\text{split}(r, 2, A) := \left\{ \bigtimes_{x \in V} Int_x \;\middle|\; \forall x \in V.\ Int_x \in \text{INT}_2(r, x, A) \right\}$$

as the set of split (small) regions, or sub-regions. Moreover, $\text{INT}_m(r, x, A)$ and $\text{split}(r, m, A)$ for $m > 2$ can be defined in a very similar way, where they equally divide the interval in each dimension for each $x \in A$ into m sub-intervals and compute the set of m-divided regions, respectively. The set A will be skipped in case of $A = V$, and we also write $\text{split}(r)$ for $\text{split}(r, 2)$.

We define the *volume* μ of a region $r = \bigtimes_{x \in V} [l_x, u_x]$ in a straight-forward way by setting $\mu(r) := \prod_{x \in V} \frac{u_x - l_x}{U_x - L_x}$. This way, the volume is the product of the relative lengths of sides of the hyper-rectangle. For a set $K = \{r_1, \ldots, r_n\}$ of regions, we define $\mu(K) := \sum_{i=1}^{n} \mu(r_i)$.

A *decision procedure* is a tool deciding the validity of formulae for a given region. There exist both approximate decision procedures [18] as well as precise ones [14, 25, 27]. Consider a predicate $constraint := f \bowtie q$ where f is a rational function over the variables in V, $\bowtie \in \{<, \leq, \geq, >\}$ and $q \in \mathbb{R}$. Let r be a given region. For an evaluation v, with $constraint\langle v \rangle$ we denote $f\langle v \rangle \bowtie q$, i.e., the constraint obtained under the valuation v. We assume that we are given a decision procedure $\text{CHECK}(constraint, r)$ which

- returns *true* only if for all $v \in r$ we have that $constraint\langle v \rangle$ is true, and
- returns *false* in case this does not hold or the result can not be decided.

3 Synthesis for PCTL

In this section we present the algorithm for synthesising PCTL formulae against PMDPs. The main routine of the algorithm is given in Algorithm 1. It maintains a set *unprocessed* of regions for which the result is still unknown, initially containing only $\bigtimes_{x \in V} \text{range}(x)$. Then, it takes a largest region out of this set

Algorithm 1: MAIN$(\mathcal{M} = (S, s_0, Act, \mathbf{P}, V), \Phi, \varepsilon)$

```
unprocessed := {⨉_{x∈V} range(x)}
result := ∅
while μ(unprocessed) ≥ ε do
    choose one largest r ∈ unprocessed
    unprocessed := unprocessed \ {r}
    b := checkState(r, Φ)
    if b = ? then
        unprocessed := unprocessed ∪ split(r)
    else
        result := result ∪ {(r, b)}
return result
```

and tries to decide its value using the procedure checkState. If checkState returns a definite answer, the pair (r, b) is added to *result*. In this case, the truth value for a state s is the same for all non-parametric MDPs represented by r. Then, $b(s)$ maps each state s to this truth value, which is constant within the region. If ? is obtained, we split the region and add the newly generated regions to *unprocessed*. The procedure is repeated until the volume of *unprocessed* is smaller than ε.

Algorithm 2 describes the procedure checkState discussed above. If successful, it returns a function mapping each state to either *true* or *false*. It may also return ? if either the truth value is different for certain parts of the region, or the truth values can not be decided for the whole region at once. Notice that for two functions $b, b' : S \to \{true, false, ?\}$, the boolean connectors $b \wedge b'$, etc. are to be understood state-wise, that is $(b \wedge b')(s) = b(s) \wedge b'(s)$, etc. For $\wedge$ and $\neg$ operations, the result is always ? in case one of the operands is ?. For $\bowtie \in \{<, \leq, \geq, >\}$, we define the negation as $\overline{<} := \geq$, $\overline{\leq} := >$, $\overline{\geq} := <$, $\overline{>} := \leq$. Boolean formulae are trivial. Below we discuss the probabilistic and reward formulae.

Since the maximal and minimal probabilities are dual, in the rest of the paper we will only consider the minimal properties and set $\bowtie \in \{>, \geq\}$ for simplicity.

3.1 Reward formula $\mathcal{R}_{\bowtie m}(\Diamond \Psi)$

Recursively, we first compute *reach* := checkState(r, Ψ). Then, we instantiate the PMDP with reward structure at centre(r) where r is the region under consideration. We obtain a non-parametric MDP, from which we compute the minimising scheduler. It is well-known that simple schedulers are sufficient to minimise (or maximise) reachability rewards for MDPs [8, 10, 29, 30]. The procedure minReachRewSched returns this simple scheduler δ such that the reachability reward is minimised for each state in the induced MDP with respect to the evaluation centre(r). A PMC $\mathcal{M}^\delta$ is further induced under this simple

Algorithm 2: checkState(r, Φ)

switch Φ **do**
- **case** a **return** b such that $b(s) = (\textbf{if } a \in AP(s) \textbf{ then } \mathit{true} \textbf{ else } \mathit{false})$
- **case** $\neg\Psi$ **return** $\neg$checkState(r, Ψ)
- **case** $\Psi_1 \wedge \Psi_2$ **return** checkState$(r, \Psi_1) \wedge$ checkState(r, Ψ_2)
- **case** $\mathcal{P}_{\bowtie p}(\varphi)$
  - $val :=$ computeProb(r, φ)
  - **if** $val = ?$ **then return** $?$
  - **for** $s \in S$ **do** $b(s) := \begin{cases} \mathit{true} \text{ if } \textsc{check}(val(s) \bowtie p, r), \\ \mathit{false} \text{ if } \textsc{check}(val(s) \overline{\bowtie} p, r), \\ ? \qquad\qquad \text{else} \end{cases}$
  - **if** $\exists s.b(s) = ?$ **then return** $?$ **else return** b
- **case** $\mathcal{R}_{\bowtie m}(\Diamond \Psi)$
  - $reach :=$ checkState(r, Ψ)
  - **if** $reach = ?$ **then return** $?$
  - $c :=$ centre(r)
  - $\delta :=$ minReachRewSched$(\mathbf{P}_c[reach], \mathbf{r}_c[reach], reach)$
  - $optRew :=$ reachRew$(\mathbf{P}^\delta[reach], \mathbf{r}^\delta[reach], reach)$
  - **for** $s \in S, \alpha \in Act(s)$ **do**
    - $checkRew(s) := \mathbf{r}[reach](s, \alpha) + \sum_{s' \in S} \mathbf{P}[reach](s, \alpha, s') \cdot optRew(s')$
    - $valid := valid \wedge \textsc{check}(optRew(s) \leq checkRew(s), r)$
  - **if** $\neg valid$ **then return** $?$
  - **for** $s \in S$ **do** $b(s) :=$ check$(optRew(s) \bowtie m, r)$
  - **if** $\exists s.b(s) = ?$ **then return** $?$ **else return** b

scheduler δ, with the corresponding matrix $\mathbf{P}^\delta$. Using reachRew, we compute the parametric reachability rewards function $optRew : S \to \mathcal{F}_V$ in this induced PMC (as in a previous publication [17]).

Recall that the scheduler δ is minimising with respect to the evaluation centre(r). Our next **for** loop checks whether this is also the case for all evaluations in the region r. It works in a similar way as the *optimality equation* [26,30]. For all states s and enabled actions α, we check whether δ is indeed minimising, but this time for all concrete models represented by the region, through the decision procedure check. In more detail, if the obtained reward $checkRew(s)$ in the **for** loop satisfies the constraint $optRew(s) \leq checkRew(s)$ for each concrete model in the region, then indeed $\delta(s)$ is minimising. In this case we have proven that δ is locally optimal for each state, which induces global optimality of the current scheduler.

3.2 Probabilistic formula $\mathcal{P}_{\bowtie p}(\varphi)$

The function computeProb, in Algorithm 3, returns a function mapping each state s to the minimal probability of all paths which fulfil φ when starting in s. Again, if this value can not be decided, the result is ?. The functions work

recursively: The cases for atomic propositions, negation and conjunction are as for usual model checking procedures. For $\mathcal{P}_{\bowtie p}(\varphi)$, we use the procedure CHECK discussed in Section 2.4 to decide the truth value for each state, if this is possible.

Algorithm 3: computeProb(r, φ)

switch φ **do**
  case $\Psi_1 \,\mathcal{U}\, \Psi_2$
    $left :=$ checkState(r, Ψ_1), $right :=$ checkState(r, Ψ_2)
    if *(*$left = ?$ or $right = ?$*)* **then return** ?
    $c :=$ centre(r)
    $\delta :=$ minUReachSched$(\mathbf{P}_c[\neg left \vee right], right)$
    $optProb :=$ uReachProb$(\mathbf{P}^\delta[\neg left \vee right], right)$
    $valid := true$
    for $s \in S, \alpha \in Act(s)$ **do**
      $checkProb(s) := \sum_{s' \in S} \mathbf{P}[\neg left \vee right](s, \alpha, s') \cdot optProb(s')$
      $valid := valid \wedge$ check$(optProb(s) \leq checkProb(s), r)$
    if $valid$ **then return** $optProb$ **else return** ?
  case $\Psi_1 \,\mathcal{U}^{\leq n}\, \Psi_2$
    $left :=$ checkState(r, Ψ_1), $right :=$ checkState(r, Ψ_2)
    if *(*$left = ?$ or $right = ?$*)* **then return** ?
    $c :=$ centre(r)
    $\delta :=$ minBReachSched$(\mathbf{P}_c[\neg left \vee right], right)$
    forall the s **do** $optProb(s) :=$ **if** $right(s)$ **then** 1 **else** 0
    $valid := true$
    for $step = n, \ldots, 1$ **do**
      $optProb' := \mathbf{P}^{\delta(\cdot, step)}[\neg left \vee right] \cdot optProb$
      for $s \in S, \alpha \in Act(s)$ **do**
        $checkProb(s) := \sum_{s' \in S} \mathbf{P}[\neg left \vee right](s, \alpha, s') \cdot optProb(s')$
        $valid := valid \wedge$ check$(optProb'(s) \leq checkProb(s), r)$
      $optProb := optProb'$
    if $valid$ **then return** $optProb$ **else return** ?

In computeProb, for $\Psi_1 \,\mathcal{U}\, \Psi_2$ we compute a minimising scheduler to fulfil the unbounded until formula for the centred parameter evaluation, using standard means, by calling minUReachSched. Notice that the minimising scheduler is a simple scheduler, which suffices for minimal reachability probabilities [6]. By uReachProb, we compute the reachability probability of the PMC induced by this scheduler (as in a previous work [17]). Note that the probability obtained this way is only valid for parameter evaluations which fulfil Assumption 1. Afterwards, we use another optimality equation [4] to check whether the decision is minimal for all parameter evaluations of the region.

For the bounded until $\Psi_1 \,\mathcal{U}^{\leq n}\, \Psi_2$, we need to consider the minimum over all *counting* schedulers. We compute the minimising scheduler for one instantiation.

Afterwards, we use a recursive (backward) characterisation [2] to prove that for each step the choices the scheduler takes are indeed optimal for the whole parameter region. We leave out the case $\mathcal{X}\,\Psi$, as it can be handled by a simpler variant of the algorithm for the bounded until.

3.3 Termination and Correctness

To guarantee termination of our algorithm, we need the following assumption.

Assumption 2. *Let $r_0 := \bigtimes_{x \in V} \mathrm{range}(x)$ denote the initial region, and ε the given precision. We assume that there exists $m \in \mathbb{N}$ with the following property. There exists a set $K \subseteq \mathrm{split}(r_0, m)$ of regions such that 1.) for all regions $r \in K$, either for all evaluations $v \in r$ it is $\mathcal{M}_v \models \Phi$, or for all evaluations $v \in r$ it is $\mathcal{M}_v \not\models \Phi$, 2.) $\mu(K) > 1 - \varepsilon$ and 3.) the decision procedure is able to decide all constraints occurring during the parameter synthesis of all regions $r \in K$.*

The assumption requires that by repeated splitting we arrive at a sufficiently large set of regions (with volume larger than $1 - \varepsilon$) in which each state has a constant truth value, decidable by the (possibly incomplete) decision procedure. It is similar to an assumption used to reason about the quasi-decidability of hybrid systems [28]. In case the assumption is valid, the following lemma guarantees termination.

Lemma 1. *Let $\mathcal{M}$ be a PMDP, Φ be a PCTL formula and $\varepsilon > 0$ the analysis precision. Then Algorithm 1 terminates in finite time with this input, given that Assumption 2 holds.*

Lemma 1 follows by a simple structural induction on the formula, provided Assumption 2 holds. We now state the correctness of the algorithm.

Lemma 2. *Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, V)$ be a PMDP, Φ be a PCTL state formula and $\varepsilon > 0$ the analysis precision. Further, assume we are using a precise decision procedure. Then Algorithm 1 is correct in the following sense. For each tuple (r, b) of the result, and for each $v \in r$ for which $\mathcal{M}_v$ is a valid MDP and for which Assumption 1 is valid, we have $\mathcal{M}_v \models \Phi$ **iff** $b(s_0)\langle v \rangle = true$.*

Notice that its correctness does not depend on Assumption 2, thus the result is correct also in case termination is not guaranteed. The proof of the correctness of Lemma 2 also follows by structural induction. For atomic propositions and boolean connectors, the induction step is trivial. For until and reachability rewards, we use the correctness of the corresponding optimality equations.

4 Experiments

We implemented the model checking procedure of Algorithm 1 in a prototypical way in our tool PARAM 2.0 α. For the analysis to be feasible, it was necessary to implement a number of optimisations. We minimise induced PMCs using

weak [3] or strong [12] bisimulation. We use a caching technique to avoid computing reachability probabilities in PMCs twice, in case the same PMCs are induced from several calls to checkState or checkRegion. We also reuse known truth values of constraints. Because we usually have to split large regions into smaller ones anyway, we do some pre-checks whether the truth value may be constant. To minimise the number of regions to be considered, and thus the overall time, we split regions along one widest side, i.e., $\text{split}(r, 2, \{x\})$ with variable x representing the (or a) widest side. For the case study under consideration, we extended an approximate decision method [18], which does not guarantee correctness. Initial experiments with exact solvers have not been successful, as verifying a single region did not terminate within several minutes. In the method used, for a constraint $f \bowtie q$ we evaluate f in the corners of the region as well as some randomly chosen points inside. The more points we evaluate, the more unlikely is a wrong result, but still correctness cannot be guaranteed formally.

We applied the implementation on a randomised consensus shared coin protocol by Aspnes and Herlihy [1], based on an existing Prism model [22]. In this case study, there are N processes sharing a counter c, which initially has the value 0. In addition, a value K is fixed for the protocol. Each process i decides to either decrement the counter with probability p_i or to increment it with probability $1 - p_i$. In contrast to the original Prism model, we do not fix the p_i to $\frac{1}{2}$, but use them as parameters of the model. After writing the counter, the process reads the value again and checks whether $c \leq -KN$ or $c \geq KN$. In the first case, the process votes 1, in the second it votes 2. In both cases, the process stops afterwards. If neither of the two cases hold, the process continues its execution. As all processes which have not yet voted try to access the counter at the same time, there is a nondeterministic choice on the access order.

A probabilistic formula. As the first property, we ask whether for each execution of the protocol the probability that all processes finally terminate with a vote of 2 is at least $\frac{K-1}{2K}$. With appropriate atomic propositions *finished* and *allCoinsEqualTwo*, this property can be expressed as $\mathcal{P}_{\geq \frac{K-1}{2K}}(\mathit{true}\,\mathcal{U}\,(\mathit{finished} \wedge \mathit{allCoinsEqualTwo}))$. For the case $N = 2$ and $K = 2$, we give results in Fig. 3.

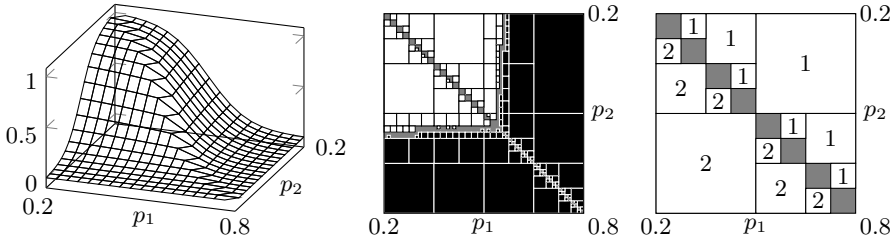

Fig. 3. Randomised consensus: $\mathcal{P}_{\geq \frac{K-1}{2K}}(\mathit{true}\,\mathcal{U}\,(\mathit{finished} \wedge \mathit{allCoinsEqualTwo}))$.

The leftmost part of the figure provides the minimal probabilities among all schedulers that all processes terminate with a vote of 2, depending on the

parameters p_i. With decreasing p_i, the probability that all processes vote 2 increases, since it becomes more likely that a process increases the counter and thus also the chance that finally $c \geq KN$ holds. The plot is symmetric, because both processes are independent and have an identical structure.

On the right part of the figure, we give an overview which schedulers are optimal for which parameter values. Here, boxes labelled with the same number share the same minimising scheduler. In case $p_1 < p_2$, to obtain the minimal probability the nondeterminism must be resolved such that the first process is activated if it has not yet voted. Doing so maximises the probability that we have $c \leq -KN$ before $c \geq KN$, and in turn minimises the probability that both processes vote 2. For $p_1 > p_2$, the second process must be preferred.

In the middle part of the figure, we give the truth values of the formula. White boxes correspond to regions where the property holds, whereas in black boxes it does not hold. In gray areas, the truth value is undecided. To keep the gray areas viewable, we chose a rather high tolerance of 0.15. The truth value decided is as expected by inspecting the plot on the left part of the figure, except for the gray boxes along the diagonal of the figure. In the gray boxes enclosed by the white area, the property indeed holds, while in the gray areas surrounded by the black area, it does not hold. The reason that these areas remain undecided is that the minimising scheduler changes at the diagonals, as discussed in the previous paragraph. If the optimal scheduler in a box is not constant for the region considered, we have to split it. Because the optimal scheduler always changes at the diagonals, there are always some gray boxes remaining.

A reward formula. As a second property, we ask whether the expected number of steps until all processes have voted is above 25, expressed as $\mathcal{R}_{>25}(\Diamond\, \mathit{finished})$. Results are given in Fig. 4. On the left part, we give the expected number of steps. This highest value is at $p_i = \frac{1}{2}$. Intuitively, in this case the counter does not have a tendency of drifting to either side, and is likely to stay near 0 for a longer time. Again, gray boxes surrounded by boxes of the same colour are those regions in which the minimising scheduler is not constant. We see from the right part of the figure that this happens along four axes. For some values of the parameters, the minimising scheduler is not always the one which always prioritises one of the processes. Instead, it may be necessary to schedule the first process, then the second again, etc. As we can see, this leads to a number of eight different schedulers to be considered for the considered variable ranges.

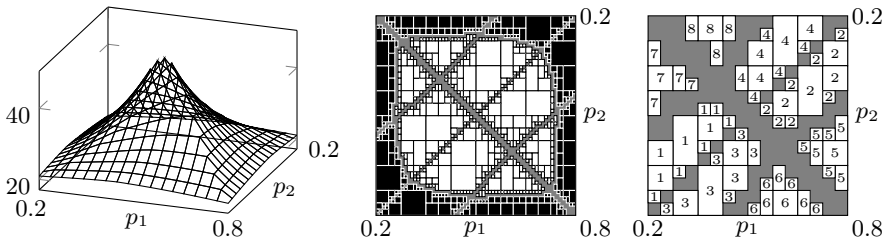

Fig. 4. Randomised consensus: $\mathcal{R}_{>25}(\Diamond\, \mathit{finished})$

Runtime. In Table 1 we give the runtime of our tool (on an Intel Core 2 Duo P9600 with 2.66 GHz running on Linux) for two processes and different constants K. Column"States" contains the number of states. The columns labelled with "Until" contain results of the first property while those labelled with "Reward" contain those of the second. Columns labelled with "min" contain just the time to compute the minimal values whereas those labelled with "truth" also include the time to compare this value against the bound of the formula. For all analyses, we chose a tolerance of $\varepsilon = 0.05$. The time is given in seconds, and "–" indicates that the analyses did not terminate within 90 minutes.

As we see, the performance drops quickly with a growing number of states. For reward-based properties, the performance is worse than for unbounded until. These analyses are more complex, as rewards have to be taken into account, and weak bisimulation can not be applied for minimisation of the induced models. In addition, a larger number of different schedulers has to be considered to obtain minimal values, which also increases the analysis time. We are however optimistic that we will be able to improve these figures, using a more advanced implementation.

K	States	Until		Reward	
		min	truth	min	truth
2	272	4.7	22.8	287.8	944.7
3	400	13.7	56.7	4610.1	–
4	528	31.7	116.1	–	–
5	656	65.5	215.2	–	–
6	784	123.4	374.6	–	–
7	912	272.6	657.4	–	–

Table 1. Randomised consensus: performance statistics

5 Conclusion

In this paper, we have studied the parameter synthesis problem of PCTL formulae for PMDPs. We have demonstrated the principal applicability of the method, using a prototypical implementation. As future work we aim to make the method applicable to models with larger state space. It will be necessary to improve the technique, from both the theory and implementation perspective. To guarantee correctness of the results, we intend to try out different solver tools, and to bring the rational functions into a form which is easier to be handled by the respective solver. Another possible future work is to extend the recent interesting work about model repair systems for PMCs [5] to PMDPs.

Acknowledgements. This work was supported by the SFB/TR 14 AVACS, FP7-ICT Quasimodo, NWO-DFG ROCKS, DAAD-MinCyT QTDDS, ERC Advanced Grant VERIWARE, MT-LAB—a VKR Centre of Excellence.

We thank Alexandru Mereacre for many comments and insightful discussions.

References

1. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. Journal of Algorithms 11(3), 441–461 (1990)
2. Baier, C.: On algorithmic verification methods for probabilistic systems (1998), Habilitationsschrift, Mannheim University

3. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: CAV. pp. 119–130. LNCS, Springer (1997)
4. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)
5. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: TACAS. LNCS, Springer (2011)
6. Bianco, A., Alfaro, L.D.: Model checking of probabilistic and nondeterministic systems. In: FSTTCS. pp. 499–513. LNCS, Springer (1995)
7. Blackwell, D.: On the functional equation of dynamic programming. Journal of Mathematical Analysis and Applications 2(2), 273–276 (1961)
8. Blackwell, D.: Positive dynamic programming. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability. pp. 415–418 (1967)
9. Bohnenkamp, H.C., van der Stok, P., Hermanns, H., Vaandrager, F.W.: Cost-optimization of the IPv4 Zeroconf protocol. In: DSN. pp. 531–540. IEEE Computer Society (2003)
10. van Dawen, R.: Finite state dynamic programming with the total reward criterion. Mathematical Methods of Operations Research 30, A1–A14 (1986)
11. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: ICTAC. pp. 280–294. LNCS, Springer (2004)
12. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. IPL 87(6), 309–315 (2003)
13. Dubins, L.E., Savage, L.: How to Gamble If You Must. McGraw-Hill (1965)
14. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT 1(3–4), 209–236 (2007)
15. Fribourg, L., André, É.: An inverse method for policy iteration based algorithms. In: INFINITY. pp. 44–61. EPTCS, Open Publishing Association (2009)
16. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: A model checker for parametric Markov models. In: CAV. pp. 660–664. LNCS, Springer (2010)
17. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. STTT 13, 3–19 (2010)
18. Han, T.: Diagnosis, synthesis and analysis of probabilistic models. Ph.D. thesis, RWTH Aachen University/University of Twente (2009)
19. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. FAC 6, 102–111 (1994)
20. Haverkort, B.R., Cloth, L., Hermanns, H., Katoen, J.P., Baier, C.: Model checking performability properties. In: DSN. pp. 103–112 (2003)
21. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 2nd edition. SIGACT News 32(1), 60–65 (2001)
22. Kwiatkowska, M., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In: CAV. LNCS, vol. 2102, pp. 194–206. Springer (2001)
23. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic model checking. In: SFM. pp. 220–270. LNCS, Springer (2007)
24. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Parametric probabilistic transition systems for system design and analysis. FAC 19(1), 93–109 (2007)
25. Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. In: Proceedings of the 16th Symposium, 8th International Conference. Held as Part of CICM '09 on Intelligent Computer Mathematics. Calculemus, Springer (2009)
26. Puterman, M.L.: Markov decision processes: Discrete stochastic dynamic programming. John Wiley and Sons (1994)
27. Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. CoRR cs.LO/0211016 (2002)
28. Ratschan, S.: Safety verification of non-linear hybrid systems is quasi-semidecidable. In: TAMC, LNCS, vol. 6108, pp. 397–408. Springer (2010)
29. Strauch, R.E.: Negative dynamic programming. Annals of Mathematical Statistics 37(4), 871–890 (1966)
30. van der Wal, J.: Stochastic dynamic programming. The Mathematical Centre, Amsterdam (1981)