Tableaux for Policy Synthesis for MDPs with PCTL* Constraints

Peter Baumgartner, Sylvie Thiébaux, and Felipe Trevizan

Data61/CSIRO and Research School of Computer Science, ANU, Australia
Email: first.last@anu.edu.au

Abstract. Markov decision processes (MDPs) are the standard formalism for modelling sequential decision making in stochastic environments. Policy synthesis addresses the problem of how to control or limit the decisions an agent makes so that a given specification is met. In this paper we consider PCTL*, the probabilistic counterpart of CTL*, as the specification language. Because in general the policy synthesis problem for PCTL* is undecidable, we restrict to policies whose execution history memory is finitely bounded a priori. Surprisingly, no algorithm for policy synthesis for this natural and expressive framework has been developed so far. We close this gap and describe a tableau-based algorithm that, given an MDP and a PCTL* specification, derives in a non-deterministic way a system of (possibly nonlinear) equalities and inequalities. The solutions of this system, if any, describe the desired (stochastic) policies. Our main result in this paper is the correctness of our method, i.e., soundness, completeness and termination.

1 Introduction

Markov decision processes (MDPs) are the standard formalism for modelling sequential decision making in stochastic environments, where the effects of an agent's actions are only probabilistically known. The core problem is to synthesize a policy prescribing or restricting the actions that the agent may undertake, so as to guarantee that a given specification is met. Popular specification languages for this purpose include CTL, LTL, and their probabilistic counterparts PCTL and probabilistic LTL (pLTL). Traditional algorithms for policy synthesis and probabilistic temporal logic model-checking [9,17] are based on bottom-up formula analysis [15,16] or Rabin automata [2,11,21].

We deviate from this mainstream research in two ways. The first significant deviation is that we consider PCTL* as a specification language, whereas previous synthesis approaches have been limited to pLTL and PCTL. PCTL* is the probabilistic counterpart of CTL* and subsumes both PCTL and pLTL. For example, the PCTL* formula $\mathbf{P}_{\geq 0.8}\,\mathbf{G}\,((T > 30^\circ) \rightarrow \mathbf{P}_{\geq 0.5}\,\mathbf{F}\,\mathbf{G}\,(T < 24^\circ))$ says *"with probability at least 0.8, whenever the temperature exceeds 30° it will eventually stay below 24° with probability at least 0.5"*. Because of the nested probability operator $\mathbf{P}$ the formula is not in pLTL, and because of the nested temporal operators $\mathbf{FG}$ it is not in PCTL either.

Because in its full generality the policy synthesis problem for PCTL* is highly undecidable [5], one has to make concessions to obtain a decidable fragment. In this paper we chose to restrict to policies whose execution history memory is finitely bounded a priori. (For example, policies that choose actions in the current state dependent on

the last ten preceding states.) However, we do target synthesizing stochastic policies, i.e., the actions are chosen according to a probability distribution (which generalizes the deterministic case and is known to be needed to satisfy certain formulas [2]). Surprisingly, no algorithm for policy synthesis in this somewhat restricted yet natural and expressive framework has been developed so far, and this paper closes this gap.

The second significant deviation from the mainstream is that we pursue a different approach based on analytic tableau and mathematical programming. Our tableau calculus is goal-oriented by focusing on the given PCTL* formula, which leads to analysing runs only on a by-need basis. This restricts the search space to partial policies that only cover the states reachable from the initial state under the policy and for which the formula imposes constraints on the actions that can be selected. In contrast, traditional automata based approaches require a full-blown state space exploration. (However, we do not have an implementation yet that allows us to evaluate the practical impact of this.) We also believe that our approach, although using somewhat non-standard tableau features, is conceptually simpler and easier to comprehend. Of course, this is rather subjective.

On a high level, the algorithm works as follows. The input is an MDP, the finite-history component of the policy to be synthesized, and a PCTL* formula to be satisfied. Starting from the MDP's initial state, the tableau calculus symbolically executes the transition system given by the MDP by analysing the syntactic structure of the given PCTL* formula, as usual with tableau calculi. Temporal formulas (e.g., **FG**-formulas) are expanded repeatedly using usual expansion laws and trigger state transitions. The process stops at trivial cases or when a certain loop condition is met. The underlying loop checking technique was developed only recently, by Mark Reynolds, in the context of tableau for satisfiability checking of LTL formulas [18]. It is an essential ingredient of our approach and we adapted it to our probabilistic setting.

Our tableaux have two kinds of branching. One kind is traditional or-branching, which represents non-deterministic choice by going down exactly one child node. It is used, e.g., in conjunction with recursively calling the tableau procedure itself. Such calls are necessary to deal with nested **P**-operators, since at the time of analyzing a **P**-formula it is, roughly speaking, unknown if the formula will hold true under the policy computed only later, as a result of the algorithm. The other kind of branching represents a union of alternatives. It is used for disjunctive formulas and for branching out from a state into successor states. Intuitively, computing the probability of a disjunctive formula $\phi_1 \vee \phi_2$ is a function of the probabilities of *both* ϕ_1 and ϕ_2, so both need to be computed. Also, the probability of an **X**-formula $\mathbf{X}\phi$ at a given state is a function of the probability of ϕ at *all* successor states, and so, again, all successor states need to be considered.

The tableau construction always terminates and derives a system of (possibly non-linear) equalities and inequalities over the reals. The solutions of this system, if any, describe the desired stochastic, finite-history policies. The idea of representing policies as the solutions of a set of mathematical constraints is inspired by the abundant work in operations research, artificial intelligence, and robotics that optimally solves MDPs with simpler constraints using linear programming [1,12,10,22].

Our main result in this paper is the correctness of our algorithm, i.e., soundness, completeness and termination. To our knowledge, it is the first and only policy synthesis algorithm for PCTL* that doesn't restrict the language (but only slightly the policies).

Related work. Methods for solving the PCTL* *model checking* problem over Markov Chains are well established. The (general) policy synthesis however is harder than the model checking problem; it is known to be undecidable for even PCTL. The main procedure works bottom-up from the syntax tree of the given formula, akin to the standard CTL/CTL* model checking procedure. Embedded $\mathbf{P}$-formulas are recursively abstracted into boolean variables representing the sets of states satisfying these formulas, which are computed by LTL model checking techniques using Rabin automata. Our *synthesis* approach is rather different. While there is a rough correspondence in terms of recursive calls to treat $\mathbf{P}$ formulas, we do not need Rabin (or any other) automata; they are supplanted by the loop-check technique mentioned above.

The work the most closely related to ours is that of Brázdil *et. al.* [8,6,7]. Using Büchi automata, they obtain complexity results depending on the variant of the synthesis problem studied. However, they consider only *qualitative* fragments. For the case of interest in this paper, PCTL*, they obtain results for the fragment qPCTL*. The logic qPCTL* limits the use of the path quantifier $\mathbf{P}$ to formulas of the form $\mathbf{P}_{=1}\,\psi$ or $\mathbf{P}_{=0}\,\psi$, where ψ is a path formula. On the other hand, we cover the full logic PCTL* which has arbitrary formulas of the form $\mathbf{P}_{\sim z}\,\psi$ where $\sim \in \{<, \leq, >, \geq\}$ and $z \in [0, 1]$. In contrast to the works mentioned, we have to restrict to memory-dependent policies with an *a priori* limited finite memory. Otherwise the logic becomes highly undecidable [5].

2 Preliminaries

We assume the reader is familiar with basic concepts of Markov Decision Processes (MDPs), probabilistic model checking, and policy synthesis. See [17,13,3] for introductions and overviews. In the following we summarize the notions relevant to us and we introduce our notation.

Given a fixed finite vocabulary AP of *atomic propositions* $a, b, c, \ldots$, a *(propositional) interpretation* I is any subset of AP. It represents the assignment of each element in I to *true* and each other atomic proposition in $AP \setminus I$ to *false*. A *distribution on a countable set* X is a function $\mu\colon X \mapsto [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$, and $Dist(X)$ is the set of all distributions on X.

A *Markov Decision Process (MDP)* is a tuple $\mathcal{M} = (S, s_{\text{init}}, A, P, L)$ where: S is a finite set of states; $s_{\text{init}} \in S$ is the *initial state*; A is a finite set of *actions* and we denote by $A(s) \subseteq A$ the *set of actions enabled* in $s \in S$; $P(t|s, \alpha)$ is the probability of transitioning to $t \in S$ after applying $\alpha \in A(s)$ in state s; and $L\colon S \mapsto 2^{AP}$ labels each state in S with an interpretation. We assume that every state has at least one enabled action, i.e., $A(s) \neq \emptyset$ for all $s \in S$, and that P is a distribution on enabled actions, i.e., $P(\cdot|s, \alpha) \in Dist(S)$ iff $\alpha \in A(s)$. For any s and $\alpha \in A(s)$ let $Succ(s, \alpha) = \{t \mid P(t|s, \alpha) > 0\}$ be the states reachable from s with non-zero probability after applying α.

Given a state $s \in S$ of $\mathcal{M}$, a *run from s (of $\mathcal{M}$)* is an infinite sequence $r = (s = s_1) \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \cdots$ of states $s_i \in S$ and actions $\alpha_i \in A(s_i)$ such that $P(s_{i+1}|s_i, \alpha_i) > 0$, for all $i \geq 1$. We denote by $Runs(s)$ the set of all runs from $s \in S$ and $Runs = \cup_{s \in S} Runs(s)$. A *path from $s \in S$ (of $\mathcal{M}$)* is a finite prefix of a run from s and we define $Paths(s)$ and $Paths$ in analogy to $Runs(s)$ and $Runs$. We often write runs and paths in abbreviated form as

state sequences $s_1 s_2 \cdots$ and leave the actions implicit. Given a path $p = s_1 s_2 \cdots s_n$ let $\text{first}(p) = s_1$ and $\text{last}(p) = s_n$. Similarly, for a run $r = s_1 s_2 \cdots$, $\text{first}(r) = s_1$.

A policy π represents a decision rule on how to choose an action given some information about the environment. In its most general form, a *history-dependent (stochastic) policy (for $\mathcal{M}$)* is a function $\pi\colon \mathit{Paths} \mapsto \mathit{Dist}(A)$ such that, for all $p \in \mathit{Paths}$, $\pi(p)(\alpha) > 0$ only if $\alpha \in A(\text{last}(p))$. Technically, the MDP $\mathcal{M}$ together with π induces an infinite-state Markov chain $\mathcal{M}_\pi$ over *Paths* and this way provides a probability measure for runs of $\mathcal{M}$ under π [14,3]. However, since *Paths* is an infinite set, a history-dependent policy might not be representable; moreover, the problem of finding such a policy that satisfies PCTL* constraints is undecidable [5]. To address these issues we limit ourselves to finite-memory policies. Such policies provide a distribution on actions for a current state from S and a current *mode*, and are more expressive than Markovian policies.

Formally, a *finite-memory policy (for $\mathcal{M}$)* is a DFA $\pi_{\text{fin}} = (M, \text{start}, \Delta, \text{act})$ where M is a finite set of *modes*, $\text{start}\colon S \mapsto M$ returns an initial mode to pair with a state $s \in S$, $\Delta\colon M \times S \mapsto M$ is the *(mode) transition function*, and $\text{act}\colon M \times S \mapsto \mathit{Dist}(A)$ is a function such that, for all $\langle m, s\rangle \in M \times S$, $\text{act}(m, s)(\alpha) > 0$ only if $\alpha \in A(s)$. We abbreviate $\text{act}(m, s)(\alpha)$ as $\text{act}(m, s, \alpha)$.

Any finite-memory policy can be identified with a history-dependent policy, see again [3] for details. Essentially, an MDP $\mathcal{M}$ together with π_{fin} again induces a Markov chain $\mathcal{M}_{\pi_{\text{fin}}}$, this time over the finite state space $M \times S$, labelling function $L_{\pi_{\text{fin}}}(\langle m, s\rangle) := L(s)$, and transition probability function $P^{\mathcal{M}_{\pi_{\text{fin}}}}(\langle m', s'\rangle | \langle m, s\rangle) := \Sigma_{\alpha \in A(s)} \text{act}(m, s, \alpha) \cdot P(s' | s, \alpha)$ if $m' = \Delta(m, s)$ and 0 otherwise. A *path from $\langle m_1, s_1\rangle$ (of $\mathcal{M}_{\pi_{\text{fin}}}$)* is a sequence of the form $\langle m_1, s_1\rangle \cdots \langle m_n, s_n\rangle$ such that $m_{i+1} = \Delta(m_i, s_i)$ and $P^{\mathcal{M}_{\pi_{\text{fin}}}}(\langle m_{i+1}, s_{i+1}\rangle | \langle m_i, s_i\rangle) > 0$, for all $1 \leq i < n$. If $m_1 = \text{start}(s_1)$ we get a *path from s_1 (of $\mathcal{M}_{\pi_{\text{fin}}}$)*, similarly for runs. The definition of the satisfaction relation "$\models$" below applies to such runs $\langle \text{start}(s_1), s_1\rangle \cdots$ from s_1 of $\mathcal{M}_{\pi_{\text{fin}}}$ if π is a finite-memory policy π_{fin}.

The definition of finite-memory policies π_{fin} can be made more sophisticated, e.g., by letting Δ depend also on actions, or by making modes dependent on a given PCTL* specification. In its current form, the Δ-component of π_{fin} can be setup already, e.g., to encode in $\langle m, s\rangle$ "the last ten states preceding s".

Policy synthesis for PCTL. (PCTL*) formulas* follow the following grammar:

$$\phi \quad := \quad \mathit{true} \mid a \in AP \mid \phi \wedge \phi \mid \neg\phi \mid \mathbf{P}_{\sim z}\, \psi \qquad \text{(State formula)}$$

$$\psi \quad := \quad \phi \mid \psi \wedge \psi \mid \neg\psi \mid \mathbf{X}\, \psi \mid \psi\, \mathbf{U}\, \psi \qquad \text{(Path formula)}$$

In the definition of state formulas, $\sim \in \{<, \leq, >, \geq\}$ and $0 \leq z \leq 1$. A *proper path formula* is a path formula that is not a state formula. A formula is *classical* iff it is made from atomic propositions and the Boolean connectives $\neg$ and $\wedge$ only (no occurrences of $\mathbf{P}$, $\mathbf{X}$ or $\mathbf{U}$). We write *false* as a shorthand for $\neg\mathit{true}$.

Given an MDP $\mathcal{M}$, a history-dependent policy π, state $s \in S$ and state formula ϕ, define a satisfaction relation $\mathcal{M}, \pi, s \models \phi$, briefly $s \models \phi$, as follows:

$$s \models \mathit{true} \qquad\qquad s \models \phi_1 \wedge \phi_2 \text{ iff } s \models \phi_1 \text{ and } s \models \phi_2$$

$$s \models a \text{ iff } a \in L(s) \qquad\qquad s \models \neg\phi \text{ iff } s \not\models \phi$$

$$s \models \mathbf{P}_{\sim z}\, \psi \text{ iff } \Pr^{\mathcal{M}_\pi}(\{r \in \mathit{Runs}^{\mathcal{M}_\pi}(s) \mid \mathcal{M}, \pi, r \models \psi\}) \sim z$$

In the preceding line, $Runs^{\mathcal{M}_\pi}(s)$ denotes the set of all runs from s of $\mathcal{M}_\pi$, and $\Pr^{\mathcal{M}_\pi}(R)$ denotes the probability of a (measurable) set $R \subseteq Runs^{\mathcal{M}_\pi}$. That is, the probability measure for $\mathcal{M}$ and π is defined via the probability measure of the Markov chain $\mathcal{M}_\pi$.

We need to define the satisfaction relation $\mathcal{M}, \pi, r \models \psi$, briefly $r \models \psi$, for path formulas ψ. Let $r = s_1 s_2 \cdots$ be a run of $\mathcal{M}$ and $r[n] := s_n s_{n+1} \cdots$, for any $n \geq 1$. Then:

$r \models \phi$ iff $\mathrm{first}(r) \models \phi$ $\quad$ $r \models \psi_1 \wedge \psi_2$ iff $r \models \psi_1$ and $r \models \psi_2$

$r \models \neg\psi$ iff $r \not\models \psi$ $\quad$ $r \models \mathbf{X}\,\psi$ iff $r[2] \models \psi$

$r \models \psi_1 \,\mathbf{U}\, \psi_2$ iff exists $n \geq 1$ s.t. $r[n] \models \psi_2$ and $r[m] \models \psi_1$ for all $1 \leq m < n$

In this paper we focus on the problem of synthesizing only the act-component of an otherwise fully specified finite memory policy. More formally:

Definition 2.1 (Policy Synthesis Problem). *Let $\mathcal{M} = (S, s_{\text{init}}, A, P, L)$ be an MDP, and $\pi_{\text{fin}} = (M, \text{start}, \Delta, \cdot)$ be a partially specified finite-memory policy with* act *unspecified. Given state formula ϕ, find* act *s.th. $\mathcal{M}, \pi_{\text{fin}}, s_{\text{init}} \models \phi$ if it exists, otherwise report failure.*

Useful facts about PCTL operators.* Next we summarize some well-known or easy-to-prove facts about PCTL* operators. By the *expansion laws* for the $\mathbf{U}$-operator we mean the following equivalences:

$$\psi_1 \,\mathbf{U}\, \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2)) \quad \neg(\psi_1 \,\mathbf{U}\, \psi_2) \equiv \neg\psi_2 \wedge (\neg\psi_1 \vee \mathbf{X}\,\neg(\psi_1 \,\mathbf{U}\, \psi_2)) \quad \text{(E)}$$

For $\sim \in \{<, \leq, >, \geq\}$ define the operators $\overline{\sim}$ and $[\sim]$ as follows:

$$\overline{<} = \geq \quad \overline{\leq} = > \quad \overline{>} = \leq \quad \overline{\geq} = < \quad [<] = > \quad [\leq] = \geq \quad [>] = < \quad [\geq] = \leq$$

Some of the following equivalences cannot be used for "model checking" PCTL* (the left (P1) equivalence, to be specific) which involves reasoning over *all* policies. In the context of Markov Chains, which we implicitly have, there is no problem:

$$\neg\mathbf{P}_{\sim z}\,\psi \equiv \mathbf{P}_{\overline{\sim} z}\,\psi \qquad \mathbf{P}_{\sim z}\,\neg\psi \equiv \mathbf{P}_{[\sim]\,1-z}\,\psi \qquad \text{(P1)}$$

$$\mathbf{P}_{\geq 0}\,\psi \equiv \mathit{true} \qquad \mathbf{P}_{>1}\,\psi \equiv \mathit{false} \qquad \text{(P2)}$$

$$\mathbf{P}_{\leq 1}\,\psi \equiv \mathit{true} \qquad \mathbf{P}_{<0}\,\psi \equiv \mathit{false} \qquad \text{(P3)}$$

$$\mathbf{P}_{\geq u}\,\mathbf{P}_{\sim z}\,\psi \equiv \mathbf{P}_{\sim z}\,\psi \quad \text{if } u \neq 0 \qquad \mathbf{P}_{>u}\,\mathbf{P}_{\sim z}\,\psi \equiv \mathbf{P}_{\sim z}\,\psi \quad \text{if } u \neq 1 \qquad \text{(P4)}$$

$$\mathbf{P}_{\leq u}\,\mathbf{P}_{\sim z}\,\psi \equiv \mathbf{P}_{\geq 1-u}\,\mathbf{P}_{\overline{\sim} z}\,\psi \qquad \mathbf{P}_{<u}\,\mathbf{P}_{\sim z}\,\psi \equiv \mathbf{P}_{>1-u}\,\mathbf{P}_{\overline{\sim} z}\,\psi \qquad \text{(P5)}$$

Nonlinear programs. Finally, a *(nonlinear) program* is a set Γ of constraints of the form $e_1 \bowtie e_2$ where $\bowtie \in \{<, \leq, >, \geq, \doteq\}$ and e_1 and e_2 are arithmetic expressions comprised of numeric real constants and variables. The numeric operators are $\{+, -, \cdot, /\}$, all with their expected meaning (the symbol $\doteq$ is equality). All variables are implicitly bounded over the range $[0, 1]$. A solver (for nonlinear programs) is a decision procedure that returns a satisfying variable assignment (a solution) for a given Γ, and reports unsatisfiability if no solution exists. We do not further discuss solvers in the rest of this paper, we just assume one as given. Examples of open source solvers include Ipopt and Couenne.[1]

[1] `http://projects.coin-or.org/`.

3 Tableau Calculus

Introduction and overview. We describe a tableau based algorithm for the policy synthesis problem in Definition 2.1. Hence assume as given an MDP $\mathcal{M} = (S, s_{\text{init}}, A, P, L)$ and a partially specified finite-memory policy $\pi_{\text{fin}} = (M, \text{start}, \Delta, \cdot)$ with act unspecified.

A *labelled formula* $\mathcal{F}$ is of the form $\langle m, s\rangle : \Psi$ where $\langle m, s\rangle \in M \times S$ and Ψ is a possibly empty set of path formulas, interpreted conjunctively. When we speak of the *probability of* $\langle m, s\rangle : \Psi$ we mean the value of $\Pr^{\mathcal{M}_{\pi_{\text{fin}}}}(\{r \in \mathit{Runs}(\langle m, s\rangle) \mid \mathcal{M}, \pi_{\text{fin}}, r \models \bigwedge \Psi\})$ for the completed π_{fin}. For simplicity we also call Ψ a "formula" and call $\langle m, s\rangle$ a *policy state*. A *sequent* is an expression of the form $\Gamma \vdash \mathcal{F}$ where Γ is a program.

Our algorithm consists of three steps, the first one of which is a tableau construction. A *tableau for* $\Gamma \vdash \mathcal{F}$ is a finite tree whose root is labelled with $\Gamma \vdash \mathcal{F}$ and such that every inner node is labelled with the premise of an inference rule and its children are labelled with the conclusions, in order. If $\Gamma \vdash \mathcal{F}$ is the label of an inner node we call $\mathcal{F}$ the *pivot of the node/sequent/inference*. By a *derivation from* $\Gamma \vdash \mathcal{F}$, denoted by $\textsc{Tableau}(\Gamma \vdash \mathcal{F})$, we mean any tableau for $\Gamma \vdash \mathcal{F}$ obtained by stepwise construction, starting from a root-node only tree and applying an inference rule to (the leaf of) every branch as long as possible. There is one inference rule, the *P*-rule, which recursively calls the algorithm itself. A branch is terminated when no inference rule is applicable, which is exactly the case when its leaf is labelled by a pseudo-sequent, detailed below. The inference rules can be applied in any way, subject to only preference constraints.

Given a state formula ϕ, the algorithm starts with a derivation from $\Gamma_{\text{init}} \vdash \mathcal{F}_{\text{init}} := \{x^{\{\phi\}}_{\langle \text{start}(s_{\text{init}}), s_{\text{init}}\rangle} \doteq 1\} \vdash \langle \text{start}(s_{\text{init}}), s_{\text{init}}\rangle : \{\phi\}$. (The constraint Γ_{init} forces ϕ to be "true".) The derivation represents the obligation to derive a satisfiable extension $\Gamma_{\text{final}} \supseteq \Gamma_{\text{init}}$. A (any) solution σ then determines the act-component act_σ of π_{fin} such that $\mathcal{M}, \pi_{\text{fin}}, s_{\text{init}} \models \phi$. In more detail, Γ_{final} will contain constraints of the form $x^{\alpha}_{\langle m,s\rangle} \doteq 0$ or $x^{\alpha}_{\langle m,s\rangle} > 0$ for the probability of applying action α in policy state $\langle m, s\rangle$. Let the *policy domain of a program* Γ be the set of all policy states $\langle m, s\rangle \in M \times S$ such that $x^{\alpha}_{\langle m,s\rangle}$ occurs in Γ, for some α. This lets us initially define $\text{act}_\sigma(m, s, \alpha) := \sigma(x^{\alpha}_{\langle m,s\rangle})$ for every $\langle m, s\rangle$ in the policy domain of Γ_{final}. Only for the purpose of satisfying the definition of finite memory policies, we then make act_σ trivially total by choosing an *arbitrary* distribution for $\text{act}_\sigma(m, s)$ for all remaining $\langle m, s\rangle \in M \times S$. (The latter are not reachable and hence do not matter.) We call $\pi_{\text{fin}}(\sigma) := (M, \text{start}, \Delta, \text{act}_\sigma)$ the *policy completed by* σ.

Similarly, Γ_{final} contains variables of the form $x^{\Psi}_{\langle m,s\rangle}$, and $\sigma(x^{\Psi}_{\langle m,s\rangle})$ is the probability of $\langle m, s\rangle : \Psi$ under the policy $\pi_{\text{fin}}(\sigma)$. (We actually need these variable indexed by tableau nodes, see below.) If Ψ is a state formula its value will be 0 or 1, encoding truth values.

Contrary to traditional tableau calculi, the result of the computation – the extension Γ_{final} – cannot always be obtained in a branch-local way. To explain, there are two kinds of branching in our tableaux: *don't-know (non-deterministic) branching* and *union branching*. The former is always used for exhaustive case analysis, e.g., whether $x^{\alpha}_{\langle m,s\rangle} \doteq 0$ or $x^{\alpha}_{\langle m,s\rangle} > 0$, and the algorithm guesses which alternative to take (cf. step 2 below). The latter analyzes the Boolean structure of the pivot. Unlike as with traditional tableaux, *all* children need to be expanded, and each fully expanded branch contributes to Γ_{final}.

More precisely, we formalize the synthesis algorithm as a three-step procedure. *Step one* consists in deriving $\textsc{Tableau}(\Gamma_{\text{init}} \vdash \mathcal{F}_{\text{init}})$. *Step two* consists in removing from the

step one tableau every don't-know branching by retaining exactly one child of the parent node of the don't-know branching, and deleting all other children and the subtrees below them. This itself is a don't-know non-deterministic process; it corresponds to going down one branch in traditional tableau. The result is denoted by $\textsc{Choose}(T_1)$, where T_1 is the step one tableau. *Step three* consists in first building a combined program by taking the union of the Γ's in the leaves of the branches of the step two tableau. This program then is extended with a set of constraints by the Force operator. More precisely, Forceing captures the situation when a run reaches a bottom strongly connected component (BSCC). Any formula is satisfied in a BSCC with probability 0 or 1, which can be determined solely by qualitative formula evaluation in the BSCC. Details are below. For now let us just define $\textsc{Gamma}(T_2) = \bigcup\{\Gamma \mid \Gamma \vdash \cdot \textit{ is the leaf of a branch in } T_2\} \cup \textsc{Force}(T_2)$ *where* $T_2 = \textsc{Choose}(T_1)$.

We can formulate our main results now. Proofs are in the long version [4].

Theorem 3.1 (Soundness). *Let* $\mathcal{M} = (S, s_{\text{init}}, A, P, L)$ *be an MDP,* $\pi_{\text{fin}} = (M, \text{start}, \Delta, \cdot)$ *be a partially specified finite-memory policy with* act *unspecified, and* ϕ *a state formula. Suppose there is a program* $\Gamma_{\text{final}} := \textsc{Gamma}(\textsc{Choose}(\textsc{Tableau}(\{x^{\{\phi\}}_{\langle \text{start}(s_{\text{init}}), s_{\text{init}}\rangle} \doteq 1\} \vdash \langle \text{start}(s_{\text{init}}), s_{\text{init}}\rangle : \{\phi\}))))$ *such that* Γ_{final} *is satisfiable. Let* σ *be any solution of* Γ_{final} *and* $\pi_{\text{fin}}(\sigma)$ *be the policy completed by* σ*. Then it holds* $\mathcal{M}, \pi_{\text{fin}}(\sigma), s_{\text{init}} \models \phi$.

Theorem 3.2 (Completeness). *Let* $\mathcal{M} = (S, s_{\text{init}}, A, P, L)$ *be an MDP,* $\pi_{\text{fin}} = (M, \text{start}, \Delta, \text{act})$ *a finite-memory policy, and* ϕ *a state formula. Suppose* $\mathcal{M}, \pi_{\text{fin}}, s_{\text{init}} \models \phi$*. Then there is a satisfiable program* $\Gamma_{\text{final}} := \textsc{Gamma}(\textsc{Choose}(\textsc{Tableau}(\{x^{\{\phi\}}_{\langle \text{start}(s_{\text{init}}), s_{\text{init}}\rangle} \doteq 1\} \vdash \langle \text{start}(s_{\text{init}}), s_{\text{init}}\rangle : \{\phi\}))))$ *and a solution* σ *of* Γ_{final} *such that* $\text{act}_\sigma(m, s, \alpha) = \text{act}(m, s, \alpha)$ *for every pair* $\langle m, s\rangle$ *in the policy domain of* Γ_{final}*. Moreover* $\mathcal{M}, \pi_{\text{fin}}(\sigma), s_{\text{init}} \models \phi$.

Inference rules. There are two kinds of inference rules, giving two kinds of branching:

$$\textit{Name}\ \frac{\Gamma \vdash \langle m, s\rangle : \Psi}{\Gamma_{\textit{left}} \vdash \langle m, s\rangle : \Psi \qquad \Gamma_{\textit{right}} \vdash \langle m, s\rangle : \Psi}\ \text{if } \textit{condition} \qquad \text{(Don't-know branching)}$$

The pivot in the premise is always carried over into both conclusions. Only the constraint Γ is modified into $\Gamma_{\textit{left}} \supseteq \Gamma$ and $\Gamma_{\textit{right}} \supseteq \Gamma$, respectively, for an exhaustive case analysis.

$$\textit{Name}\ \frac{\Gamma \vdash \langle m, s\rangle : \Psi}{\Gamma_1 \vdash \langle m_1, s_1\rangle : \Psi_1 \quad \cup \quad \cdots \quad \cup \quad \Gamma_n \vdash \langle m_n, s_n\rangle : \Psi_n}\ \text{if } \textit{condition } (n \geq 1)$$

(Union branching)

All union branching rules satisfy $\Gamma_i \supseteq \Gamma$, and $\langle m_i, s_i\rangle = \langle m, s\rangle$ or $\langle m_i, s_i\rangle = \langle \Delta(m, s), t\rangle$ for some state t. The $\cup$-symbol is decoration for distinguishing the two kinds of branching but has no meaning beyond that. Union branching stands for the union of the runs from $\langle m_i, s_i\rangle$ satisfying Ψ_i, and computing its probability requires developing *all* n children.

We need to clarify a technical add-on. Let u be the tableau node with the premise pivot $\langle m, s\rangle : \Psi$. A union branching inference extends u with children nodes, say, $u_1, \ldots, u_n$, with conclusion pivots $\langle m_i, s_i\rangle : \Psi_i$. The program Γ_n will contain a constraint that makes a variable $(x_u)^{\Psi}_{\langle m,s\rangle}$ for the premise dependent on all variables $(x_{u_i})^{\Psi_i}_{\langle m_i,s_i\rangle}$ for the respective conclusions. *This is a key invariant and is preserved by all inference*

rules. In order to lighten the notation, however, we usually drop the variable's index, leaving the node implicit. For instance, we write $x^{\Psi}_{\langle m,s\rangle}$ instead of $(x_u)^{\Psi}_{\langle m,s\rangle}$. The index u is needed for not inadvertently identifying the same pivot at different points in the symbolic execution of a run. Fresh names $x, y, z, \ldots$ for the variables would do as well.

Most unary union branching rules have a premise $\Gamma \vdash \langle m,s\rangle : \{\psi\} \uplus \Psi$ and the conclusion is $\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \Psi'$, for some Ψ'. The pivot is specified by pattern matching, where $\uplus$ is disjoint union, and γ_{one} is a macro that expands to $x^{\{\psi\}\uplus\Psi}_{\langle m,s\rangle} \doteq x^{\Psi'}_{\langle m,s\rangle}$.

Other inference rules derive pseudo-sequents of the form $\Gamma \vdash \text{✗}, \Gamma \vdash \text{✓}, \Gamma \vdash \mathsf{Yes\text{-}Loop}$ and $\Gamma \vdash \mathsf{No\text{-}Loop}$. They indicate that the probability of the pivot is 0, 1, or that a loop situation arises that may need further analysis. Pseudo-sequents are always leaves.

Now we turn to the concrete rules. They are listed in decreasing order of preference.

$$\top \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \Psi} \begin{cases}\text{if } \psi \text{ is classical and} \\ L(s) \models \psi\end{cases} \qquad \text{✗} \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\psi\} \uplus \Psi}{\Gamma, x^{\{\psi\}\uplus\Psi}_{\langle m,s\rangle} \doteq 0 \vdash \text{✗}} \begin{cases}\text{if } \psi \text{ is classical and} \\ L(s) \not\models \psi\end{cases}$$

$$\text{✓} \quad \frac{\Gamma \vdash \langle m,s\rangle : \emptyset}{\Gamma, x^{\emptyset}_{\langle m,s\rangle} \doteq 1 \vdash \text{✓}} \qquad \neg\neg \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\neg\neg\psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\psi\} \cup \Psi}$$

$$\neg\mathbf{P} \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\neg \mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\mathbf{P}_{\not\sim z}\, \psi\} \cup \Psi} \qquad \mathbf{P}\neg \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\mathbf{P}_{\sim z}\, \neg\psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\mathbf{P}_{[\sim]\, 1-z}\, \psi\} \cup \Psi}$$

These are rules for evaluating classical formulas and for negation. The ✗ rule terminates the branch and assigns a probability of 0 to the premise pivot, as no run from $\langle m,s\rangle$ satisfies (the conjunction of) $\{\psi\} \uplus \Psi$, as ψ is false in s. A similar reasoning applies to the $\top$ and ✓ rules. The $\neg\mathbf{P}$ and $\mathbf{P}\neg$ rules are justified by law (P1). The $\mathbf{P}\neg$ rule is needed for removing negation between $\mathbf{P}$-formulas as in $\mathbf{P}_{\sim z}\, \neg \mathbf{P}_{\sim v}\, \psi$.

$$\wedge \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\psi_1 \wedge \psi_2\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\psi_1, \psi_2\} \cup \Psi}$$

$$\neg\wedge \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\neg(\psi_1 \wedge \psi_2)\} \uplus \Psi}{\Gamma \vdash \langle m,s\rangle : \{\neg\psi_1\} \cup \Psi \quad \cup \quad \Gamma, \gamma \vdash \langle m,s\rangle : \{\psi_1, \neg\psi_2\} \cup \Psi}$$

$$\text{where } \gamma \;=\; x^{\{\neg(\psi_1\wedge\psi_2)\}\uplus\Psi}_{\langle m,s\rangle} \doteq x^{\{\neg\psi_1\}\cup\Psi}_{\langle m,s\rangle} + x^{\{\psi_1,\neg\psi_2\}\cup\Psi}_{\langle m,s\rangle}$$

These are rules for conjunction. Not both ψ_1 and ψ_2 can be classical by preference of the $\top$ and ✗ rules. The $\wedge$ rule is obvious with the conjunctive reading of formula sets. The $\neg\wedge$ rule deals, essentially, with the disjunction $\neg\psi_1 \vee \neg\psi_2$, which requires splitting. However, unlike to the classical logic case, $\neg\psi_1 \vee \neg\psi_2$ represents the union of the runs from s satisfying $\neg\psi_1$ and the runs from s satisfying $\neg\psi_2$. As these sets may overlap the rule works with a *disjoint* union by taking $\neg\psi_1$ on the one side, and $\psi_1 \wedge \neg\psi_2$ on the other side so that it is correct to add their probabilities up in γ.

$$\mathbf{P}1 \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\psi'\} \cup \Psi} \begin{cases}\text{if } \mathbf{P}_{\sim z}\, \psi \text{ is the left hand side of an equivalence} \\ \text{(P2)-(P5) and } \psi' \text{ is its right hand side}\end{cases}$$

$$\mathbf{P}2 \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\psi\} \cup \Psi} \text{ if } \textit{see text} \qquad \mathbf{P}3 \quad \frac{\Gamma \vdash \langle m,s\rangle : \{\mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\neg\psi\} \cup \Psi} \text{ if } \textit{see text}$$

These are rules for simplifying **P**-formulas. The condition in **P**2 is "$\sim \in \{>, \geq\}$ and ψ is a state formula", and in **P**3 it is "$\sim \in \{<, \leq\}$ and ψ is a state formula". In the rules **P**2 and **P**3 trivial cases for z are excluded by preference of **P**1. Indeed, this preference is even needed for soundness. The rule **P**2 can be explained as follows: suppose we want to know if $\mathcal{M}, \pi, \langle m, s\rangle \models \mathbf{P}_{\sim z}\, \psi$. For that we need the probability of the set of runs from $\langle m, s\rangle$ that satisfy ψ and compare it with z. Because ψ is a *state* formula this set is comprised of all runs from s if $\mathcal{M}, \pi, \langle m, s\rangle \models \psi$, or the empty set otherwise, giving it probability 1 or 0, respectively. With $\sim \in \{>, \geq\}$ conclude $\mathcal{M}, \pi, s \models \mathbf{P}_{\sim z}\, \psi$, or its negation, respectively. The rule **P**3 is justified analogously. The only difference is that $\sim \in \{<, \leq\}$ and so the $\mathbf{P}_{\sim z}$ quantifier acts as a negation operator instead of idempotency.

At this stage, when all rules above have been applied exhaustively to a given branch, the leaf of that branch must be of the form $\Gamma \vdash \langle m, s\rangle : \{\mathbf{P}_{\sim z_1}\, \psi_1, \ldots, \mathbf{P}_{\sim z_n}\, \psi_n\}$, for some $n \geq 0$, where each ψ_i is a non-negated proper path formula.

$$\mathbf{P}\quad \frac{\Gamma \vdash \langle m, s\rangle : \Psi}{\Gamma, \Gamma', \gamma_{\text{left}} \vdash \langle m, s\rangle : \Psi \qquad \Gamma, \Gamma', \gamma_{\text{right}} \vdash \langle m, s\rangle : \Psi} \begin{cases}\text{if } \mathbf{P}_{\sim z}\, \psi \in \Psi, \text{ and}\\ \gamma_{\text{left}} \notin \Gamma \text{ and } \gamma_{\text{right}} \notin \Gamma\end{cases}$$

$$\mathbf{P}\top\quad \frac{\Gamma \vdash \langle m, s\rangle : \{\mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m, s\rangle : \Psi} \text{ if } \gamma_{\text{left}} \in \Gamma$$

$$\mathbf{P}\boldsymbol{\times}\quad \frac{\Gamma \vdash \langle m, s\rangle : \{\mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}{\Gamma, x^{\{\mathbf{P}_{\sim z}\, \psi\} \uplus \Psi}_{\langle m, s\rangle} \doteq 0 \vdash \boldsymbol{\times}} \text{ if } \gamma_{\text{right}} \in \Gamma$$

$$\text{where}\quad \Gamma' = \textsc{Gamma}(\textsc{Choose}(\textsc{Tableau}(\emptyset \vdash \langle \text{start}(s), s\rangle : \{\psi\}))),$$

$$\gamma_{\text{left}} = x^{\{\psi\}}_{\langle \text{start}(s), s\rangle} \sim z, \text{ and } \gamma_{\text{right}} = x^{\{\psi\}}_{\langle \text{start}(s), s\rangle} \overline{\sim} z$$

These are rules for **P**-formulas. Unlike classical formulas, **P**-formulas cannot be evaluated in a state, because their truth value depends on the solution of the program Γ_{final}. The **P** rule analyzes $\mathbf{P}_{\sim z}\, \psi$ in a deferred way by first getting a constraint $x^{\{\psi\}}_{\langle \text{start}(s), s\rangle} \doteq e$, for some expression e, for the probability of $\langle \text{start}(s), s\rangle : \{\psi\}$ by a recursive call.[2] This call is not needed if Γ already determines a truth value for $\mathbf{P}_{\sim z}\, \psi$ because of $\gamma_{\text{left}} \in \Gamma$ or $\gamma_{\text{right}} \in \Gamma$. These tests are done modulo node labels of variables, i.e., $(x_u)^{\{\psi\}}_{\langle \text{start}(s), s\rangle}$ and $(x_v)^{\{\psi\}}_{\langle \text{start}(s), s\rangle}$ are considered equal for any u, v. Because the value of e is not known at the time of the inference, the **P** rule don't-know non-deterministically branches out into whether $x^{\{\psi\}}_{\langle \text{start}(s), s\rangle} \sim z$ holds or not, as per the constraints γ_{left} and γ_{right}. The $\mathbf{P}\top$ and $\mathbf{P}\boldsymbol{\times}$ rules then lift the corresponding case to the evaluation of $\mathbf{P}_{\sim z}\, \psi$, which is possible now thanks to γ_{left} or γ_{right}.

Observe the analogy between these rules and their counterparts $\top$ and $\boldsymbol{\times}$ for classical formulas. Note that the rules **P**, $\mathbf{P}\top$ and $\mathbf{P}\boldsymbol{\times}$ cannot be combined into one, because γ_{left} or γ_{right} could have been added earlier, further above in the branch, or in a recursive call. In this case only $\mathbf{P}\top/\mathbf{P}\boldsymbol{\times}$ can applied.

At this stage, in a leaf $\Gamma \vdash \langle m, s\rangle : \Psi$ the set Ψ cannot contain any state formulas, as they would all be eliminated by the inference rules above; all formulas in Ψ now are possibly negated **X**-formulas or **U**-formulas.

[2] By the semantics of the **P**-operator, the sub-derivation has to start from $\langle \text{start}(s), s\rangle$, not $\langle m, s\rangle$.

$$\mathbf{U}\quad \frac{\Gamma \vdash \langle m,s\rangle : \{\psi_1 \,\mathbf{U}\, \psi_2\} \uplus \Psi}{\Gamma \vdash \langle m,s\rangle : \{\psi_2\} \cup \Psi \qquad \cup \qquad \Gamma, \gamma \vdash \langle m,s\rangle : \{\psi_1, \neg\psi_2,\, \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2)\} \cup \Psi}$$

$$\text{where } \gamma \;=\; x^{\{\psi_1 \,\mathbf{U}\, \psi_2\} \uplus \Psi}_{\langle m,s\rangle} \doteq x^{\{\psi_2\} \cup \Psi}_{\langle m,s\rangle} + x^{\{\psi_1, \neg\psi_2, \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2)\} \cup \Psi}_{\langle m,s\rangle}$$

$$\neg\mathbf{U}\quad \frac{\Gamma \vdash \langle m,s\rangle : \{\neg(\psi_1 \,\mathbf{U}\, \psi_2)\} \uplus \Psi}{\Gamma \vdash \langle m,s\rangle : \{\neg\psi_1, \neg\psi_2\} \cup \Psi \quad \cup \quad \Gamma, \gamma \vdash \langle m,s\rangle : \{\psi_1, \neg\psi_2, \mathbf{X}\,\neg(\psi_1 \,\mathbf{U}\, \psi_2)\} \cup \Psi}$$

$$\text{where } \gamma \;=\; x^{\{\neg(\psi_1 \,\mathbf{U}\, \psi_2)\} \uplus \Psi}_{\langle m,s\rangle} \doteq x^{\{\neg\psi_1, \neg\psi_2\} \cup \Psi}_{\langle m,s\rangle} + x^{\{\psi_1, \neg\psi_2, \mathbf{X}\,\neg(\psi_1 \,\mathbf{U}\, \psi_2)\} \cup \Psi}_{\langle m,s\rangle}$$

These are expansion rules for **U**-formulas. The standard expansion law is $\psi_1 \,\mathbf{U}\, \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2))$. As with the $\neg\wedge$ rule, the disjunction in the expanded formula needs to be disjoint by taking $\psi_2 \vee (\psi_1 \wedge \neg\psi_2 \wedge \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2))$ instead. Similarly for $\neg\mathbf{U}$.

$$\neg\mathbf{X}\quad \frac{\Gamma \vdash \langle m,s\rangle : \{\neg\mathbf{X}\,\psi\} \uplus \Psi}{\Gamma, \gamma_{\text{one}} \vdash \langle m,s\rangle : \{\mathbf{X}\,\neg\psi\} \cup \Psi}$$

The $\neg\mathbf{X}$ rule is obvious.

At this stage, if $\Gamma \vdash \langle m,s\rangle : \Psi$ is a leaf sequent then Ψ is of the form $\{\mathbf{X}\,\psi_1, \ldots, \mathbf{X}\,\psi_n\}$, for some $n \geq 1$. This is an important configuration that justifies a name: we say that a labelled formula $\langle m,s\rangle : \Psi$, a sequent $\Gamma \vdash \langle m,s\rangle : \Psi$ or a node labelled with $\Gamma \vdash \langle m,s\rangle : \Psi$ is *poised* if Ψ is of the form $\{\mathbf{X}\,\psi_1, \ldots, \mathbf{X}\,\psi_n\}$ where $n \geq 1$. (The notion "poised" is taken from [18].) A poised $\langle m,s\rangle : \{\mathbf{X}\,\psi_1, \ldots, \mathbf{X}\,\psi_n\}$ will be expanded by transition into the successor states of s by using enabled actions $\alpha \in A(s)$. That some α is enabled does not, however, preclude a policy with $\text{act}_\sigma(m,s,\alpha) = 0$. The rule **A** makes a guess whether this is the case or not:

$$\mathsf{A}\quad \frac{\Gamma \vdash \langle m,s\rangle : \Psi}{\Gamma, \gamma_{\text{left}} \vdash \langle m,s\rangle : \Psi \qquad \Gamma, \gamma_{\text{right}} \vdash \langle m,s\rangle : \Psi} \begin{cases} \text{if } \Gamma \vdash \langle m,s\rangle : \Psi \text{ is poised,} \\ \alpha \in A(s),\ \gamma_{\text{left}} \notin \Gamma \text{ and } \gamma_{\text{right}} \notin \Gamma \end{cases}$$

$$\text{where } \gamma_{\text{left}} \;= x^{\alpha}_{\langle m,s\rangle} \doteq 0 \text{ and } \gamma_{\text{right}} \;= x^{\alpha}_{\langle m,s\rangle} > 0$$

With a minor modification we get a calculus for *deterministic* policies. It only requires to re-define γ_{right} as $\gamma_{\text{right}} = \; x^{\alpha}_{\langle m,s\rangle} \doteq 1$. As a benefit the program Γ_{final} will be linear.

After the **A** rule has been applied exhaustively, for each $\alpha \in A(s)$ either $x^{\alpha}_{\langle m,s\rangle} > 0 \in \Gamma$ or $x^{\alpha}_{\langle m,s\rangle} \doteq 0 \in \Gamma$. If $x^{\alpha}_{\langle m,s\rangle} > 0 \in \Gamma$ we say that α *is prescribed in* $\langle m,s\rangle$ *by* Γ and define $\mathit{Prescribed}(\langle m,s\rangle, \Gamma) = \{\alpha \mid x^{\alpha}_{\langle m,s\rangle} > 0 \in \Gamma\}$.

The set of prescribed actions in a policy state determines the *Succ*-relation of the Markov chain under construction. To get the required distribution over enabled actions, it suffices to enforce a distribution over prescribed actions, with this inference rule:

$$\mathit{Prescribed}\quad \frac{\Gamma \vdash \langle m,s\rangle : \Psi}{\Gamma, \gamma^{\alpha}_{\langle m,s\rangle} \vdash \langle m,s\rangle : \Psi} \begin{cases} \text{if } \Gamma \vdash \langle m,s\rangle : \Psi \text{ is poised,} \\ \alpha \in A(s) \text{ and } \gamma^{\alpha}_{\langle m,s\rangle} \notin \Gamma \end{cases}$$

$$\text{where } \gamma^{\alpha}_{\langle m,s\rangle} = \Sigma_{\alpha \in \mathit{Prescribed}(\langle m,s\rangle,\Gamma)}\, x^{\alpha}_{\langle m,s\rangle} \doteq 1$$

If the Choose operator in step two selects the leftmost branch among the **A**-inferences then Γ_{final} contains $x^{\alpha}_{\langle m,s\rangle} \doteq 0$, for all $\alpha \in A(s)$. This is inconsistent with the constraint

introduced by the *Prescribed*-inference, corresponding to the fact that runs containing $\langle m, s\rangle$ in this case do not exist.

We are now turning to a "loop check" which is essential for termination, by, essentially, blocking the expansion of certain states into successor states that do not mark progress. For that, we need some more concepts. For two nodes u and v in a branch we say that *u is an ancestor of v* and write $u \leq v$ if $u = v$ or u is closer to the root than v. An ancestor is *proper*, written as $u < v$, if $u \leq v$ but $u \neq v$. We say that two sequents $\Gamma_1 \vdash \mathcal{F}_1$ and $\Gamma_2 \vdash \mathcal{F}_2$ are *indistinguishable* iff $\mathcal{F}_1 = \mathcal{F}_2$, i.e., they differ only in their Γ-components. Two nodes u and v are *indistinguishable* iff their sequents are. We write Ψ_u to denote the formula component of u's label, i.e., to say that the label is of the form $\Gamma \vdash \langle m, s\rangle : \Psi_u$; similarly for $\mathcal{F}_u$ to denote u's labelled formula.

Definition 3.3 (Blocking). *Let w be a poised leaf and $v < w$ an ancestor node. If (i) v and w are indistinguishable, and (ii) for every* $\mathbf{X}$*-eventuality $\mathbf{X}(\psi_1 \mathbf{U} \psi_2)$ in Ψ_v there is a node x with $v < x \leq w$ such that $\psi_2 \in \Psi_x$ then w is* yes-blocked by *v. If there is an ancestor node $u < v$ such that (i) u is indistinguishable from v and v is indistinguishable from w (and hence u is indistinguishable from w), and (ii) for every* $\mathbf{X}$*-eventuality $\mathbf{X}(\psi_1 \mathbf{U} \psi_2)$ in Ψ_u, if there is a node x with $\psi_2 \in \Psi_x$ and $v < x \leq w$ then there is a node y with $\psi_2 \in \Psi_y$ and $u < y \leq v$, then w is* no-blocked by *u.*

When we say that a sequent is yes/no-blocked we mean that its node is yes/no-blocked.

In the yes-blocking case all $\mathbf{X}$-eventualities in Ψ_v become satisfied along the way from v to w. This is why w represents a success case. In the no-blocking case some $\mathbf{X}$-eventualities in Ψ_v may have been satisfied along the way from u to v, but not all, as this would be a yes-blocking instead. Moreover, no progress has been made along the way from v to w for satisfying the missing $\mathbf{X}$-eventualities. This is why w represents a failure case. The blocking scheme is adapted from [18] for LTL satisfiability to our probabilistic case. See [19,18] for more explanations and examples, which are instructive also for its usage in our framework.

Blocking is used in the following inference rules, collectively called the Loop rules. In these rules, the node v is an ancestor node of the leaf the rule is applied to.

$$\textsf{Yes-Loop}\ \frac{\Gamma \vdash \langle m, s\rangle : \Psi}{\Gamma, x^{\Psi}_{\langle m,s\rangle} \doteq (x_v)^{\Psi}_{\langle m,s\rangle} \vdash \textsf{Yes-Loop}}\ \text{if } \Gamma \vdash \langle m, s\rangle : \Psi \text{ is yes-blocked by } v$$

$$\textsf{No-Loop}\ \frac{\Gamma \vdash \langle m, s\rangle : \Psi}{\Gamma, x^{\Psi}_{\langle m,s\rangle} \doteq (x_v)^{\Psi}_{\langle m,s\rangle} \vdash \textsf{No-Loop}}\ \text{if } \Gamma \vdash \langle m, s\rangle : \Psi \text{ is no-blocked by } v$$

In either case, if v is indistinguishable from w then the probability of $\mathcal{F}_v$ and $\mathcal{F}_w$ are exactly the same, just because $\mathcal{F}_v = \mathcal{F}_w$. This justifies adding $x^{\Psi}_{\langle m,s\rangle} \doteq (x_v)^{\Psi}_{\langle m,s\rangle}$.

The Loop rules have a side-effect that we do not formalize: they add a link from the conclusion node (the new leaf node) to the blocking node v, called the *backlink*. It turns the tableau into a graph that is no longer a tree. The backlinks are used only for reachability analysis in step three of the algorithm. Figure 1 has a graphical depiction.

By preference of inference rules, the $\mathbf{X}$ rule introduced next can be applied only if a Loop rule does not apply. The Loop rules are at the core of the termination argument.[3]

[3] The argument is standard for calculi based on formula expansion, as embodied in the $\mathbf{U}$ and $\neg\mathbf{U}$ rules: the sets of formulas obtainable by these rules is a subset of an a priori determined *finite*

For economy of notation, when $\Psi = \{\psi_1, \ldots, \psi_n\}$, for some $\psi_1, \ldots, \psi_n$ and $n > 0$, let $\mathbf{X}\,\Psi$ denote the set $\{\mathbf{X}\,\psi_1, \ldots, \mathbf{X}\,\psi_n\}$.

$$\mathbf{X}\ \frac{\Gamma \vdash \langle m, s\rangle : \mathbf{X}\,\Psi}{\Gamma \vdash \langle m', t_1\rangle : \Psi \quad \cup \quad \cdots \quad \cup \quad \Gamma \vdash \langle m', t_{k-1}\rangle : \Psi \quad \cup \quad \Gamma, \gamma_1 \vdash \langle m', t_k\rangle : \Psi}$$

where
$$\begin{aligned} m' &= \Delta(m, s) \\ \{t_1, ..., t_k\} &= \textstyle\bigcup_{\alpha \in \mathit{Prescribed}(\langle m,s\rangle, \Gamma)} \mathit{Succ}(s, \alpha)\ \text{, for some } k \geq 0 \\ \gamma_1 &= x^{\mathbf{X}\Psi}_{\langle m,s\rangle} \doteq \Sigma_{\alpha \in \mathit{Prescribed}(\langle m,s\rangle, \Gamma)} [x^{\alpha}_{\langle m,s\rangle} \cdot (\Sigma_{t \in \mathit{Succ}(s,\alpha)} P(t|s, \alpha) \cdot x^{\Psi}_{\langle m',t\rangle})] \end{aligned}$$

This is the (only) rule for expansion into successor states. If u is the node the $\mathbf{X}$ rule is applied to and $u_1, \ldots, u_k$ are its children then each u_i is called an $\mathbf{X}$-*successor (of u)*.

The $\mathbf{X}$ rule follows the set of actions prescribed in $\langle m, s\rangle$ by Γ through to successor states. This requires summing up the probabilities of carrying out α, as represented by $x^{\alpha}_{\langle m,s\rangle}$, multiplied by the sums of the successor probabilities weighted by the respective transition probabilities. This is expressed in the constraint γ_1. Only these k successors need to be summed up, as all other, non-prescribed successors, have probability 0.

Forcing probabilities. We are now turning to the Force operator which we left open in step three of the algorithm. It forces a probability 0 or 1 for certain labelled formulas occurring in a bottom strongly connected component in a tree from step two. The tree in the figure to the right helps to illustrate the concepts introduced in the following.

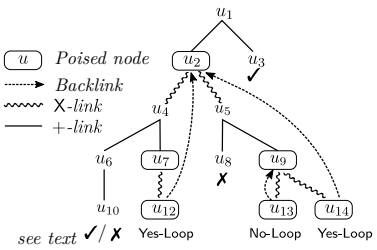


Fig. 1. An example tableau T from step 2. The subgraph below u_2 is a strongly connected component if u_{10} is ✗-ed.

We need some basic notions from graph theory. A subset M of the nodes N of a given graph is *strongly connected* if, for each pair of nodes u and v in M, v is reachable from u passing only through states in M. A *strongly connected component (SCC)* is a maximally strongly connected set of nodes (i.e., no superset of it is also strongly connected). A *bottom strongly connected component (BSCC)* is a SCC M from which no state outside M is reachable from M.

Let $T = \textsc{Choose}(\textsc{Tableau}(\Gamma \vdash \mathcal{F}))$ be a tree without don't-know branching obtained in step 2. We wish to take T together with its backlinks as the graph under consideration and analyse its BSCCs. However, for doing so we cannot take T as it is. On the one hand, our tableaux describe state transitions introduced by $\mathbf{X}$ rule applications. Intuitively, these are amenable to BSCC analysis as one would do for state transition systems. On the other hand, T has interspersed rule applications for analysing Boolean structure, which distort the state transition structure. These rule applications have to be taken into account prior to the BSCC analysis proper.

set of formulas. This set consists of all subformulas of the given formula closed under negation and other operators. Any infinite branch hence would have to repeat one of these sets infinitely often, which is impossible with the loop rules. Moreover, the state set S and the mode set M are finite and so the other rules do not cause problems either.

For this, we distinguish between **X**-links and +-links in T. An **X**-link is an edge between a node and its child if the **X** rule was applied to the node, making its child an **X**-successor, otherwise it is a +-link. ("+-link" because probabilities are summed up.)

Let u be a node in T and $Subtree_T(u)$, or just $Subtree(u)$, the subtree of T rooted at u without the backlinks. We say that u is a *0-deadend (in T)* if $Subtree_T(u)$ has no **X**-links and every leaf in $Subtree_T(u)$ is ✗-ed. In a 0-deadend the probabilities all add up to a zero probability for the pivot of u. This is shown by an easy inductive argument.

Definition 3.4 (Ambiguous node). *Let u be a node in T. We say that u* is ambiguous (in T) *iff (i) $Subtree_T(u)$ contains no ✓-ed leaf, and (ii) $Subtree_T(u)$ contains no* **X***-successor 0-deadend node. We say that u* is unambiguous *iff u is not ambiguous.*

The main application of Definition 3.4 is when the node u is the root of a BSCCs, defined below. The probability of u's pivot $\langle m, s\rangle : \Psi$ then is not uniquely determined. This is because expanding u always leads to a cycle, a node with the same pivot, and there is no escape from that according to conditions (i) or (ii) in Definition 3.4. In other words, the probability of $\langle m, s\rangle : \Psi$ is defined only in terms of itself.[4]

In the figure above, the node u_1 is unambiguous because of case (i) in Definition 3.4. Assuming u_{10} is ✓-ed, the node u_2 is unambiguous by case (i). The pivot in u_{10}, then, has probability 1 which is propagated upwards to u_4 (and enforces probability 0 for the pivot of u_7). It contributes a non-zero probability to the transition from u_2 to u_4 and this way escapes a cycle. If u_{10} is ✗-ed, the node u_2 is ambiguous.

If case (ii) in Definition 3.4 is violated there is an **X**-successor node whose pivot has probability 0. Because every **X**-link has a non-zero transition probability, the probabilities obtained through the other **X**-successor nodes add up to a value strictly less than 1. This also escapes the cycle leading to underspecified programs (not illustrated above).

Let $0(T) = \{w \mid w \text{ is a node in some 0-deadend of } T\}$ be all nodes in all 0-deadends in T. In the example, $0(T) = \{u_6, u_{10}, u_8\}$ if u_{10} is ✗-ed and $0(T) = \{u_8\}$ if u_{10} is ✓-ed.

Let u be a node in T and $M(u) = \{w \mid w \text{ is a node in } Subtree(u)\} \setminus 0(T)$. That is, $M(u)$ consists of the nodes in the subtree rooted at u after ignoring the nodes from the 0-deadend subtrees. In the example $M(u_2) = \{u_2, u_4, u_5, u_7, u_9, u_{12}, u_{13}, u_{14}\}$ if u_{10} is ✗-ed. If u_{10} is ✓-ed then u_6 and u_{10} have to be added.

We say that *u is the root of a BSCC (in T)* iff u is poised, ambiguous and $M(u)$ is a BSCC in T (together with the backlinks). In the example, assume that u_{10} is ✗-ed. Then u_2 is poised, ambiguous and the root of a BSCC. In the example, that $M(u_2)$ is a BSCC in T is easy to verify.

Now suppose that u is the root of a BSCC with pivot $\langle m, s\rangle : \mathbf{X}\,\Psi$. This means that the probability of $\langle m, s\rangle : \mathbf{X}\,\Psi$ is not uniquely determined. This situation then is fixed

[4] In terms of the resulting program, $(x_u)^{\psi}_{\langle m,s\rangle}$ is not constrained to any specific value in [0..1]. This can be shown by "substituting in" the equalities in Γ_{final} for the probabilities of the pivots in the subtree below u and arithmetic simplifications.

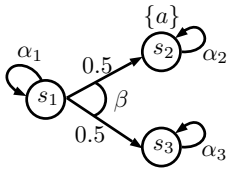
by means of the Force operation, generally defined as follows:

$$\begin{aligned}\textsc{Bscc}(T) &:= \{u \mid u \text{ is the root of a BSCC in } T\}\\ \textsc{Force}(T) &:= \{(x_u)^{\mathbf{X}\Psi}_{\langle m,s\rangle} \doteq \chi \mid u \in \textsc{Bscc}(T), \text{ and}\\ &\qquad \text{if some leaf of the subtree rooted at } u \text{ is a } \mathsf{Yes\text{-}Loop}\\ &\qquad\quad \text{then } \chi = 1 \text{ else } \chi = 0\}\end{aligned}$$

That is, Forceing removes the ambiguity for the probability of the pivot $\langle m, s\rangle : \mathbf{X}\,\Psi$ at the root u of a BSCC by setting it to 1 or to 0. If Forceing adds $(x_u)^{\mathbf{X}\Psi}_{\langle m,s\rangle} \doteq 1$ then there is a run that satisfies every $\mathbf{X}$-eventuality in $\mathbf{X}\,\Psi$, by following the branch to a Yes-Loop. Because we are looking at a BSCC, for fairness reasons, *every* run will do this, and infinitely often so, this way giving $\mathbf{X}\,\Psi$ probability 1. Otherwise, if there is no Yes-Loop, there is some $\mathbf{X}$-eventuality in $\mathbf{X}\,\Psi$ that cannot be satisfied, forcing probability 0.

4 Example

The following is only a brief summary of an example spelled out in detail in the long version [4] of this paper. Consider the MDP on the right. The initial state is s_1. Action β leads non-deterministically to states s_2 and s_3, each with probability 0.5. The actions α_i for $i \in \{1, 2, 3\}$ are self-loops with probability one (not shown). The label set of s_2 is $\{a\}$ in all other states it is empty. The partially specified finite-memory policy $\pi_{\text{fin}} = (\{\text{m}\}, \text{start}, \Delta, \cdot)$ has a single mode m, making π_{fin} Markovian. The functions start and Δ hence always return m, allowing us to abbreviate $\langle m, s_i\rangle$ as just s_i. Let the state formula of interest be $\phi = \mathbf{P}_{\geq 0.3}\,\mathbf{F}\,\mathbf{G}\,a$. We wish to obtain a Γ_{final} such that any solution σ synthesizes a suitable act_σ, i.e., the policy $\pi_{\text{fin}}(\sigma)$ completed by σ satisfies $\mathcal{M}, \pi_{\text{fin}}(\sigma), s_1 \models \phi$.



The BSCCs depend on whether $\text{act}_\sigma(\text{m}, s_1, \beta) > 0$ holds, i.e., if β can be executed at s_1. (This is why the calculus needs to make a corresponding guess, with its A-rule.) If not, then s_2 and s_3 are unreachable, and the self-loop at s_1 is the only BSCC, which does not satisfy $\mathbf{G}\,a$. If yes, then there are two BSCCs, the self-loop at s_2 and the self-loop at s_3, and the BSCC at s_2 satisfies $\mathbf{G}\,a$. By fairness of execution, with probability one some BSCC will be reached, and the BSCC at s_2 is reached with probability 0.5, hence, if $\text{act}_\sigma(\text{m}, s_1, \beta) > 0$. In other words, devising *any* policy that reaches s_2 will hence suffice to satisfy ϕ. The expected result thus is just a constraint on σ saying $\text{act}_\sigma(\text{m}, s_1, \beta) > 0$. Indeed, the derivation will show that.

In brief, the derivation starts with the initial sequent $\{x^{\phi}_{s_1} \doteq 1\} \vdash s_1 : \{\phi\}$. The first inference is a P-inference, branching out on $x^{\mathbf{F}\mathbf{G}\,a}_{s_1} \geq 0.3$ and its negation. (The latter case quickly leads to an unsatisfiable program.) The P-inference triggers a recursive call with the start sequent $\emptyset \vdash x^{\mathbf{F}\mathbf{G}\,a}_{s_1}$. This tableau leads to an open branch with sequent $\emptyset \vdash x^{\mathbf{X}\mathbf{F}\mathbf{G}\,a}_{s_1}$ and all other branches ✗-ed, inducing a constraint $x^{\mathbf{F}\mathbf{G}\,a}_{s_1} \doteq x^{\mathbf{X}\mathbf{F}\mathbf{G}\,a}_{s_1}$. This is plausible, as s_1 falsifies $\mathbf{G}\,a$, and hence exactly the successor states of s_1 need to be considered.

As said, the interesting case is if β can be executed at s_1, so let us assume that. The tableau derivation continues the open branch and arrives at s_3. Further expansion leads

to No-Loop leaves only. This gives a trivial constraint $x_{s_3}^{\mathbf{FG}\,a} \doteq x_{s_3}^{\mathbf{FG}\,a}$ only, but Forceing adds $x_{s_3}^{\mathbf{FG}\,a} \doteq 0$. The tableau derivation also arrives at s_2, this time with a BSCC with a Yes-Loop leaf, contributing $x_{s_2}^{\mathbf{FG}\,a} \doteq 1$.

If desired, the resulting program Γ_{final} can be simplified so that it becomes obvious that only the constraint $x_{s_1}^{\beta} > 0$ is essential for satisfiability.

5 Conclusions and Future Work

In this paper we presented a first-of-its kind algorithm for the controller synthesis problem for Markov Decision Processes whose intended behavior is described by PCTL* formulas. The only restriction we had to make – to get decidability – is to require policies with finite history. We like to propose that the description of the algorithm is material enough for one paper, and so we leave many interesting questions for future work.

The most pressing theoretical question concerns the exact worst-case complexity of the algorithm. Related to that, it will be interesting to specialize and analyze our framework for fragments of PCTL*, such as probabilistic LTL and CTL or simpler fragments and restricted classes of policies that might lead to *linear* programs (and ideally to solving only a polynomial number of such programs). For instance, we already mentioned that computing deterministic policies leads to linear programs in our tableau (see the description of the A inference rule how this is done.) Moreover, it is well-known that cost-optimal stochastic policies for classes of MDPs with simple constraints bounding the probability of reaching a goal state can be synthesized in linear time in the size of the MDP by solving a *single* linear program [1,12]. An interesting question is how far these simple constraints can be generalised towards PCTL* whilst remaining in the linear programming framework (see e.g. [20]).

On implementation, a naïve implementation of the algorithm as presented above would perform poorly in practice. However, it is easy to exploit some straightforward observations for better performance. For instance, steps one (tableau construction) and two (committing to a don't-know non-deterministic choice) should be combined into one. Then, if a don't know non-deterministic inference rule is carried out the first time, every subsequent inference with the same rule and pivot can be forced to the same conclusion, at the time the rule is applied. Otherwise an inconsistent program would result, which never needs to be searched for. Regarding space, although all children of a union branching inference rule need to be expanded, this does not imply they always all need to be kept in memory simultaneously. Nodes can be expanded in a one-branch-at-a-time fashion and using a global variable for Γ_{final} for collecting the programs in the leaves of the branches *if they do not belong to a bottom strongly connected component*. Otherwise, the situation is less obvious and we leave it to future work. Another good source of efficiency improvements comes from more traditional tableau. It will be mandatory to exploit techniques such as dependency-directed backtracking, lemma learning, and early failure checking for search space pruning.

Acknowledgements

This research was funded by AFOSR grant FA2386-15-1-4015. We would also like to thank the anonymous reviewers for their constructive and helpful comments.

References

1. E. Altman. *Constrained Markov Decision Processes*, Volume 7. CRC Press, 1999.
2. C. Baier, M. Größer, M. Leucker, B Bollig, and F. Ciesinski. Controller Synthesis for Probabilistic Systems. In *TCS2004*, 2004.
3. C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. Peter Baumgartner, Sylvie Thiébaux, and Felipe Trevizan. Tableaux for Policy Synthesis for MDPS With PCTL* Constraints. *CoRR*, abs/1706.10102, 2017.
5. T. Brázdil, V. Brozek, V. Forejt, and A. Kucera. Stochastic Games With Branching-time Winning Objectives. In *21th IEEE Symp. on Logic in Computer Science LICS*, 2006.
6. T. Brázdil and V. Forejt. Strategy Synthesis for Markov Decision Processes and Branching-time Logics. In *18th Int. Conf. on Concurrency Theory CONCUR*, 2007.
7. T. Brázdil, V. Forejt, and A. Kucera. Controller Synthesis and Verification for Markov Decision Processes With Qualitative Branching Time Objectives. In *ICALP*, 2008.
8. T. Brázdil, A. Kucera, and O. Strazovský. On the Decidability of Temporal Properties of Probabilistic Pushdown Automata. In *STACS*, 2005.
9. C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907, 1995.
10. X. C. Ding, A. Pinto, and A. Surana. Strategic Planning Under Uncertainties via Constrained Markov Decision Processes. In *IEEE Int. Conf. on Robotics and Automation ICRA*, 2013.
11. X. C. Ding, S. Smith, C. Belta, and D. Rus. Optimal Control of Markov Decision Processes With Linear Temporal Logic Constraints. *IEEE Trans. Automat. Contr.*, 59(5):1244–1257, 2014.
12. D. Dolgov and E. Durfee. Stationary Deterministic Policies for Constrained Mdps With Multiple Rewards, Costs, and Discount Factors. In *IJCAI*, 2005.
13. V. Forejt, M. Z. Kwiatkowska, G. Norman, and D. Parker. Automated Verification Techniques for Probabilistic Systems. In *SFM*, 2011.
14. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains: With a Chapter of Markov Random Fields by David Griffeath*, volume 40. Springer, 2012.
15. A. Kucera and O. Strazovský. On the Controller Synthesis for Finite-state Markov Decision Processes. In *Theoretical Computer Science*, 2005.
16. M. Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic Model Checking. In *SFM*, 2007.
17. M. Z. Kwiatkowska and D. Parker. Automated Verification and Strategy Synthesis for Probabilistic Systems. In *ATVA*, 2013.
18. M. Reynolds. A New Rule for LTL Tableaux. In *GandALF*, 2016.
19. M. Reynolds. A Traditional Tree-style Tableau for LTL. *CoRR*, abs/1604.03962, 2016.
20. J. Sprauel, A. Kolobov, and F. Teichteil-Königsbuch. Saturated Path-constrained MDP: Planning Under Uncertainty and Deterministic Model-checking Constraints. In *AAAI*, 2014.
21. M. Svorenová, I. Cerna, and C. Belta. Optimal Control of Mdps With Temporal Logic Constraints. In *CDC*, 2013.
22. F. Trevizan, S. Thiébaux, P. Santana, and B. Williams. Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems. In *ICAPS*, 2016.