This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

Probabilistic Preference Planning Problem for Markov Decision Processes

Meilun Li, Andrea Turrini, Ernst Moritz Hahn, Zhikun She and Lijun Zhang

Abstract—The classical planning problem aims to find a sequence of permitted actions leading a system to a designed state, i.e., to achieve the system's task. However, in many realistic cases we also have requirements on how to complete the task, indicating that some behaviors and situations are more preferred than others. In this paper, we present the probabilistic preference-based planning problem ($\mathrm{P4}$) for Markov decision processes, where the preferences are defined based on an enriched probabilistic LTL-style logic. We first recall $\mathrm{P4Solver}$, an SMT-based planner computing the preferred plan by reducing the problem to a quadratic programming one previously developed to solve $\mathrm{P4}$. To improve computational efficiency and scalability, we then introduce a new encoding of the probabilistic preference-based planning problem as a multi-objective model checking one, and propose the corresponding planner $\mathrm{P4Solver}_{MO}$. We illustrate the efficacy of both planners on some selected case studies to show that the model checking-based algorithm is considerably more efficient than the quadratic-programming-based one.

Index Terms—Planning, Markov Decision Processes, preference, quadratic programming, multi-objective model checking.

1 Introduction

Planning problem aims to identify a policy that mandates the sequence of actions to perform so that the system reaches the desired goal states, i.e., the system achieves its task. Many real world tasks can be modeled as planning problems, such as automatic driving with navigation systems [15], [44], [51], scheduling of multi-agent networks [31], [47], automobile assembly lines [49], and recently, drones which automatically track subjects (such as vehicles). Also in software engineering many problems are closely related to planning. As an example, consider program analysis based on static analyzers [25], [39]. To check whether the reported bugs are spurious or real, state-of-the-art SMT constraint solvers can be used to synthesize the sequence of actions leading to the bug. This can be considered as a planning problem, in which the goal is the buggy states. Similarly, software validation to determine the parameters so that the software satisfies desired properties [21], [54] can also be interpreted as a planning problem.

For many concrete problems, however, users may have extra requirements on the systems or software, that is, besides completing the given task, users may also propose requirements on the procedure solving the task. For instance, one may prefer some particular route to others in navigation due to traffic jam, or may hope to pass through some specific states before reaching the goal when handling multiple tasks. The planning problem can thus be enriched by adding user-defined preferences on the way the goal is achieved, resulting in a preference-based planning problem.

As an extension of [43], in this paper we consider the probabilistic preference-based planning problem ($\mathrm{P4}$) on Markov Decision Processes (MDPs) [6]. Markov Decision Processes, as widespread probabilistic model for planning problems [8], in many cases provide more accurate representations of real world systems than nondeterministic transition systems (NTSs). For instance, consider the transmission of messages on an unreliable channel: in the majority of the cases, the message is delivered correctly, but in a small fraction, the message is corrupted or even lost. When we use an NTS for representing such an unreliable channel, we are not able to describe the fraction of correctly delivered messages: we can only model whether the message has been delivered correctly. This means that in many cases using purely nondeterministic transitions is not sufficient to describe accurately the overall underlying causality of the system; using an MDP instead allows us to easily model probabilistic outcomes, since we can for instance represent the fact that 98% of the messages are delivered correctly.

To formally describe the properties verified in $\mathrm{P4}$, we introduce preference and goal formulas which are encoded in an extension of the probabilistic linear-time temporal logic (probabilistic LTL, PLTL) [30], with semantics defined on finite paths, since the goal formula is evaluated once the execution terminates. In this way we can express properties like "the probability of terminating in goal states is at least 0.99", "the probability of remaining in the safe states is between 0.85 and 0.95", etc.

- *This work was partially supported by the Guangdong Science and Technology Department (Grant No. 2018B010107004), the National Natural Science Foundation of China (Grants Nos. 11422111, 61532019, 61761136011, 61572024, 61836005), the Beijing Natural Science Foundation (Grant No. Z180005), NWO under grant PrimaVera NWA.1160.18.238, STW under the project 15474 SEQUOIA, the EU under project 864075 CAESAR, ProRail and Deutsche Bahn.*
- *Meilun Li is with School of Mathematics and Systems Science, Beihang University, Beijing, China.*
  E-mail: meilun.li@buaa.edu.cn
- *Andrea Turrini (co-first author) and Lijun Zhang (corresponding author) are with State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China as well as with Institute of Intelligent Software, Guangzhou, Guangzhuo, China.*
  E-mail: {turrini, zhanglj}@ios.ac.cn
- *Ernst Moritz Hahn is with University of Twente, Enschede, The Netherlands.*
  E-mail: e.m.hahn@utwente.nl
- *Zhikun She (co-corresponding author) is with LMIB and School of Mathematics and Systems Science, Beihang University, Beijing, China.*
  E-mail: zhikun.she@buaa.edu.cn

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

Together with the formal framework of P4, we present the P4Solver planner [43] synthesizing property-dependent policies for P4 instances. The idea underlying P4Solver relies on a forward technique based on formula progression [3], [40], that is, from the initial state of the MDP, both the goal and preference formulas are unrolled according to the performed transitions and the reached states, resulting in the parts of each formula still to be satisfied. We can encode the progression conditions as a quadratically constrained programming problem (QCPP), solve it, and then derive the policy from the QCPP solution.

P4Solver can be considered as a proof-of-concept showing that P4 is indeed decidable. However, P4Solver does not scale well. As the experiments in [43] suggest, P4Solver can solve only rather small systems. In this extended paper, we also present a novel approach for solving P4 instances, based on techniques developed for model checking. Observe that standard model checking techniques are not directly applicable to solve P4. For instance, model checking considers only one formula at a time, while in P4 we have to simultaneously consider both goal and preference formulas (detailed discussion is in Section 8). However, as we will see in Section 6, after encoding the termination of finite paths, action occurrences, and property formulas, P4 can be transformed to a multi-objective model checking problem [22], [24], [42] that aims to find a policy satisfying multiple logical formulas at the same time. This leads to a new planner P4Solver_{MO} allowing for better scalability than P4Solver. As shown in Section 7, P4Solver_{MO} allows us to solve much larger problems than P4Solver.

The use of algorithms developed for model checking as planners and viceversa is getting more and more attention, with efforts bridging the gap between the two communities (see, e.g. [32], [36], [55] and their bibliography). Model checking and planning communities have multiple points in common, like MDPs as models, finding policies/strategies, satisfaction of logical constraints; in this work we add yet another stone to the bridge between the two communities.

Preference planning on non-probabilistic models has got an extensive attention at the start of the 21st century, when a number of languages for specifying propositional preferences were proposed [7], [9], [14], [53]. An example of these languages is the Qualitative Choice Logic (QCL) [9], which is used to represent alternative, ranked options for problem solutions via the concept of satisfaction degree, similar to the preset order in the preference formula used in [7], [43] and this paper. QCL is a propositional language and does not support specification of temporal properties as in P4. $\mathcal{PP}$ [53] and $\mathcal{LPP}$ [7] support qualitative temporal preferences, with $\mathcal{LPP}$ further allowing arbitrary nesting of LTL formulas and preference over action occurrences. We follow their idea when defining the preference properties with a preset order, and to some extend propose a simplified language for planning on MDP. PDDL3 [27] is an outstanding quantitative language supporting temporal path constraints and preferences. It is used in many preference planners such as MIPS-BDD [20] and HTNPLAN-P [52] and the planner with users' pre-defined strategies [35].

Planning problem on MDPs attracts continuous interests of researchers. A value iteration method based on complete structure of Bellman equations was first proposed in [6], followed by policy iteration [26]. Two variants, UCT [38] and LRTDP [37], were later proposed to handle the planning problems for large state spaces via trial-based methods [34]. UCT uses a Monte-Carlo tree search algorithm to unfold the state space and has heuristics balancing the choice between repeatedly visiting paths with high value and exploring new parts of the MDP. LRTDP heuristically subsamples the transition function and uses optimizations to make it amenable for planning on MDPs. Lastly, it is worthy to mention the solution proposed in [55], which uses linear programming methods to solve the planning problem on MDPs, with the aim to optimize the discounted rewards under constraints given as PLTL formulas. Compared to the PLTL constraints in [55], the syntax of preference properties proposed in this paper further support preference over action occurrences. Moreover, the major difference of [55] and this paper is that [55] considers planning over reward structures, while we consider planning over LTL-like goal formulas.

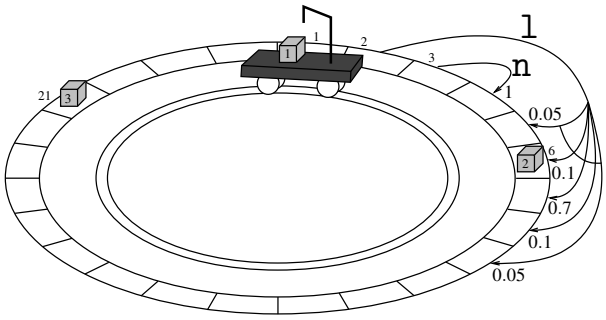

Fig. 1. The robot MDP

Organization of the Paper.

We organize this paper as follows. In Section 2, we present our running example describing a robot operating on a rail that moves boxes between different locations. In Section 3, we provide definitions of Markov decision processes and probability of finite paths. After that, in Section 4 we present the probabilistic preference planning problem (P4) and then in Section 5 the corresponding algorithm P4Solver. In Section 6, we introduce our novel approach P4Solver_{MO} for solving P4, based on techniques developed for multi-objective model checking. In Section 7, we show that P4Solver_{MO} allows for better scalability so that it is able to solve much larger problems than P4Solver. In Section 8, we consider other potential methods for solving P4 and its extensions. We finally conclude the paper in Section 9.

2 Running Example: The Rail Robot

In literature, manufacturing systems, such as automobile assembly lines, can often be logically modeled as a network of loops [23], [49]. Moreover, reliability of machines operating in such systems can be characterized by probabilistic components. Inspired by such scenarios, in the remainder of this section we present a simplified example exhibiting such behaviors. We shall also use this example to illustrate our logic for specifying properties.

Consider Fig. 1, where we have a robot moving in one direction on a ring rail surrounded by N loading and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

unloading areas (or positions). The task of the robot is to move B boxes between these areas, with $B < N$, according to their given initial position and designed final location. The robot alternates between two modes, control ($mode(\texttt{c})$) and action ($mode(\texttt{a})$), which we explain as follows.

In the control mode, the robot decides what to do next: either to move on the rail (`m`) or to use its arm to load/unload boxes (`a`). As movement, either it can go to the next area (`n`), or it can go quickly 5 areas ahead (`l`); the actual reached area depends on the distance: starting from the area i, action `n` reaches the area $(i+1) \bmod N$ with probability 1 while action `l` reaches the area $(i+5) \bmod N$ with probability 0.7, the areas $(i+4) \bmod N$ and $(i+6) \bmod N$ with probability 0.1 each, and the areas $(i+3) \bmod N$ and $(i+7) \bmod N$ with probability 0.05 each. In the action mode, when facing the area i, the robot can pick up the box j ($\texttt{p}(j,i)$) if area i contains box j and no box is already carried by the robot, or drop the carried box k ($\texttt{d}(k,i)$) if area i is empty. Both operations succeed with probability 0.95; with the remaining probability 0.05 the operation fails and the box remains where it was. Finally, we assume in the example that only one box can be carried by the robot at a time.

We can note that the robot is nondeterministic, since it has multiple choices about what action to perform next. This allows the robot to behave in different ways, depending on the actions actually performed. Solving a planning problem for the robot means resolving such nondeterminism so to achieve a given goal, i.e., finding the proper sequences of actions that make the robot fulfill its requirements, such as to place the boxes in the designed areas. Solving a preference planning problem requires the sequences of actions and robot configurations to respect some given constraint, like never move quickly, besides achieving the goal. The choice of the next action can depend on the past actions and configurations, as well as the information about the goal and the constraint to fulfill.

In the next section we recall the definition of MDP and use the robot example to explain MDPs and related notions.

3 Markov Decision Processes

In this section we recall the definition of *Markov Decision Processes* (MDPs) [6] that we use as our model. Informally, a *Markov Decision Process* is a transition system equipped with the freedom to choose the probability distribution it follows to evolve at present state, which is determined by the choice of the action. Given a finite set of states S, we denote the set of probability distributions over S by $Dist(S)$. We also denote by $\delta_x \in Dist(S)$ the *point* distribution of $x \in S$ such that $\delta_x(y) = 1$ if $y = x$, and $\delta_x(y) = 0$ otherwise. We fix Σ as the finite set of *atomic propositions*. Each state s in the MDP is *labelled* with a set of atomic propositions satisfied in state s. The formal definition of MDP is as follows.

Definition 1. *A* Markov Decision Process *is a tuple* $\mathcal{M} = (S, \Sigma, L, Act, \bar{s}, \mathrm{P})$ *where*

- S *is a finite set of* states,
- Σ *is a finite set of* atomic propositions,
- $L\colon S \to 2^{\Sigma}$ *is a* labeling function,
- Act *is a finite set of* actions,
- $\bar{s} \in S$ *is the* initial state, *and*
- $\mathrm{P}\colon S \times Act \to Dist(S)$ *is the partial* transition probability function.

We denote by $Act(s)$ the set of actions enabled by state s, i.e., $Act(s) = \{\, a \in Act \mid \mathrm{P}(s,a) \text{ is defined} \,\}$. Let $\natural \notin Act$ be a meta-action representing the choice to terminate the current execution; we extend Act and $Act(s)$ to $Act_\natural = Act \cup \{\natural\}$ and $Act_\natural(s) = Act(s) \cup \{\natural\}$, respectively. Note that we can use variables with finite domain to generate the MDP states and Boolean formulas to encode L. For instance, the states S of the rail robot example include the current location of the robot ($robotAt(p)$), which box (if any) is carried by the robot ($carryBox(b)$), the location of each box ($boxAt(b,p)$) and the mode of the robot ($mode(m)$).

A *finite path* $\rho = s_0 a_1 s_1 \dots a_n s_n$ in an MDP $\mathcal{M}$ is a finite sequence of alternating states and actions such that for each $0 < i \leq n$, we have $a_i \in Act(s_{i-1})$ and $\mathrm{P}(s_{i-1}, a_i)(s_i) > 0$. The first s_0 and last s_n state of ρ is denoted by $first(\rho)$ and $last(\rho)$, respectively. We denote by $|\rho|$ the length of ρ, i.e., the number of actions occurring in ρ. For each $1 \leq i \leq n$, we denote by $action(\rho, i)$ the ith action a_i and, for $0 \leq j \leq n$ by $state(\rho, j)$ the $(j+1)$-th state s_j in ρ and by $suffix(\rho, j)$ the suffix of ρ starting from s_j. We denote by $Paths^*_{\mathcal{M}}$ (or $Paths^*$ if $\mathcal{M}$ is clear) the set of all finite paths of $\mathcal{M}$.

The choice of the next step to perform, if any, is resolved by a *policy* π based on the previous history. A policy for an MDP $\mathcal{M}$ is a function $\pi\colon Paths^* \to Dist(Act_\natural)$ such that for each $\rho \in Paths^*$, it is $\pi(\rho) \in Dist(Act_\natural(last(\rho)))$, i.e., the policy π can only choose probabilistically between enabled actions and termination.

Given an MDP $\mathcal{M}$, a policy π and a state s induce a subprobability distribution $\mu_{\pi,s}$ over $Paths^*_{\mathcal{M}}$ as follows. For a finite path $\rho = s_0 a_1 s_1 \dots a_n s_n$, the probability of ρ (denoted as $\mu_{\pi,s}(\rho)$), i.e., the probability of terminating the evolution after ρ occurs, is defined as $\mu_{\pi,s}(\rho) = \delta_s(s_0) \cdot \big(\prod_{i=1}^{n} \pi(\rho_{i-1})(a_i) \cdot \mathrm{P}(s_{i-1}, a_i)(s_i)\big) \cdot \pi(\rho)(\natural)$, where $\rho_j = s_0 a_1 s_1 \dots a_j s_j$. This definition can be extended to a set of finite paths: given a set of finite paths $\mathcal{B}$, we define $\mu_{\pi,s}(\mathcal{B})$ as $\mu_{\pi,s}(\mathcal{B}) = \sum_{\rho \in \mathcal{B}} \mu_{\pi,s}(\rho)$. In the remainder of the paper we assume that $\mu_{\pi,s}(Paths^*) = 1$, i.e., π eventually chooses to stop for sure.

As an example of path and its probability, consider the case where the empty robot moves from area 0 to area 4 to pick up the box 2 and then stops. The corresponding path is $\rho = s_0 \texttt{m} s_1 \texttt{l} s_2 \texttt{a} s_3 \texttt{p}(2,4) s_4$, where each action is chosen by the policy with probability 1 and the states have the following meaning:

State	Robot at	Mode	CarryBox	Box 2 at
s_0	0	c	-	4
s_1	0	m	-	4
s_2	4	c	-	4
s_3	4	a	-	4
s_4	4	c	2	-

Therefore $\mu_{\pi,s_0}(\rho)$ is equal to

$$\begin{aligned}
&\delta_{s_0}(s_0) \cdot \pi(s_0)(\texttt{m}) \cdot \mathrm{P}(s_0, \texttt{m})(s_1) \\
&\quad \cdot \pi(s_0 \texttt{m} s_1)(\texttt{l}) \cdot \mathrm{P}(s_1, \texttt{l})(s_2) \\
&\quad \cdot \pi(s_0 \texttt{m} s_1 \texttt{l} s_2)(\texttt{a}) \cdot \mathrm{P}(s_2, \texttt{a})(s_3) \\
&\quad \cdot \pi(s_0 \texttt{m} s_1 \texttt{l} s_2 \texttt{a} s_3)(\texttt{p}(2,4)) \cdot \mathrm{P}(s_3, \texttt{p}(2,4))(s_4) \\
&\quad \cdot \pi(s_0 \texttt{m} s_1 \texttt{l} s_2 \texttt{a} s_3 \texttt{p}(2,4) s_4)(\natural),
\end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

i.e., $1 \cdot (1 \cdot 1) \cdot (1 \cdot 0.1) \cdot (1 \cdot 1) \cdot (1 \cdot 0.95) \cdot 1 = 0.095$.

4 Definition of Probabilistic Preference Planning Problem

In this section we present the Probabilistic Preference Planning Problem (P4) introduced in [43]. This is the extension to probabilistic settings of the classical preference-based planning problem given in [7]. The definition of the preference-based planning problem (being it P4 or the classical setting given in [7]) is based on the concept of *property formulas*, which are from an enriched LTL-style logic over finite paths.

Definition 2. *The syntax of a* property formula φ *is defined as:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{final}(\varphi) \mid \mathbf{occ}(a) \mid \mathbf{X}(\varphi) \mid \mathbf{U}(\varphi, \varphi) \mid \forall x(\varphi)$$

where $p \in \Sigma$ is an atomic proposition, $a \in Act$ is an action, and x is a variable that can only appear either in p, or as an action $b \in Act$ in $\mathbf{occ}(b)$.

In this work we assume that the variables x occurring in the $\forall x(\cdot)$ operators have a finite domain, in line with [7], [43] and the state of the art in automated classical planning. Thus, in practice, $\forall x(\varphi)$ can be seen as syntactic sugar for a finite conjunction of parameterized formulas.

Differently from the usual LTL setting, the semantics of property formulas is given with respect to finite paths. Given an MDP $\mathcal{M}$, a finite path ρ, and a formula φ, we use $\rho \models_{\mathcal{M}} \varphi$ (or $\rho \models \varphi$ if $\mathcal{M}$ is clear from the context) to represent that the path ρ satisfies the property φ. The formal semantics of property formulas is defined inductively as:

$\rho \models p$	if $p \in L(\mathit{first}(\rho))$
$\rho \models \neg\varphi$	if $\rho \not\models \varphi$, i.e., it is not the case that $\rho \models \varphi$
$\rho \models \varphi \wedge \psi$	if $\rho \models \varphi$ and $\rho \models \psi$
$\rho \models \mathbf{final}(\varphi)$	if $\mathit{last}(\rho) \models \varphi$
$\rho \models \mathbf{X}(\varphi)$	if $\lvert\rho\rvert \geq 1$ and $\mathit{suffix}(\rho, 1) \models \varphi$
$\rho \models \mathbf{U}(\varphi, \psi)$	if $\rho \models \psi$ or $\rho \models \varphi \wedge \mathbf{X}(\mathbf{U}(\varphi, \psi))$
$\rho \models \mathbf{occ}(a)$	if $\lvert\rho\rvert \geq 1$ and $\mathit{action}(\rho, 1) = a$
$\rho \models \forall x(\varphi)$	if $\rho \models \bigwedge_{v \in \mathrm{dom}(x)} \varphi[x/v]$

where $\mathrm{dom}(x)$ is the domain of x and $\varphi[x/v]$ represents the formula where all free occurrences of x in φ have been replaced by $v \in \mathrm{dom}(x)$.

As an example, consider $\mathbf{final}(\forall i(\mathit{boxAt}(i, i)))$ in the rail robot example which means that, when the robot stops, all boxes are placed in the corresponding areas, i.e., box 1 is in area 1, box 2 in area 2, and so on.

Notice that the formula $\mathbf{occ}(a)$ holds on ρ if the *first* action of ρ is a. Temporal operators $\mathbf{X}(\varphi)$ and $\mathbf{U}(\varphi, \psi)$ are defined in the standard way [56]. The formula $\forall x(\varphi)$ holds if φ holds on ρ for all assignments of x in $\mathrm{dom}(x)$, where x is a free variable appearing in φ. Throughout the paper, we use standard derived operators such as $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, $\mathbf{True} = p \vee \neg p$, $\mathbf{False} = \neg\mathbf{True}$, $\mathbf{F}(\varphi) = \mathbf{U}(\mathbf{True}, \varphi)$, $\mathbf{G}(\varphi) = \neg\mathbf{F}(\neg\varphi)$, $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$, and $\exists x(\varphi) = \neg\forall x(\neg\varphi)$.

Equipped with property formulas, we define a *probabilistic property formula* as $\mathcal{P}_J(\varphi)$ where φ is a property formula and $J \subseteq [0, 1]$ is a closed interval. For given MDP $\mathcal{M}$, $s \in S$, and policy π, we say $s \models^{\pi}_{\mathcal{M}} \mathcal{P}_J(\varphi)$ if $\mu_{\pi,s}(\{\, \rho \in \mathit{Paths}^{*}_{\mathcal{M}} \mid \rho \models \varphi \,\}) \in J$. The notion $s \models^{\pi}_{\mathcal{M}} \mathcal{P}_J(\varphi)$ will be abbreviated as $s \models \mathcal{P}_J(\varphi)$ if $\mathcal{M}$ and π are clear. Then we define *preference formulas* following the style for the Atomic Preference Formulas in [7].

Definition 3. *A* preference formula Φ_P *is of the form $\psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$, where for each $1 \leq i \leq k$, ψ_i is a probabilistic property formula and $\psi_k = \mathcal{P}_{[1,1]}(\mathbf{True})$.*

The order of probabilistic property formulas represents the priority of preferences to be satisfied. The least preferred formula $\mathcal{P}_{[1,1]}(\mathbf{True})$ means no preference is considered. With these notions, now we introduce the preference planning problem for the probabilistic setting.

Definition 4 (P4 [43])**.** *A* Probabilistic Preference Planning Problem *(*P4*) is a tuple $(\mathcal{M}, \Phi_G, \Phi_P)$ where*

- *$\mathcal{M}$ is an MDP,*
- *Φ_G is a* goal formula *which is a probabilistic property formula of the form $\Phi_G = \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ where γ is a property formula, and*
- *$\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$ is a preference formula.*

A *solution* to a P4 instance is a policy π such that $\bar{s} \models^{\pi}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$. An *optimal solution* to a P4 is a solution π^* such that there exists $i \in \{1, \ldots, k\}$ such that $\bar{s} \models^{\pi^*}_{\mathcal{M}} \psi_i$ and for each $0 < j < i$ and each solution π, it is $\bar{s} \not\models^{\pi}_{\mathcal{M}} \psi_j$.

Since we assume that $\psi_k = \mathcal{P}_{[1,1]}(\mathbf{True})$, we have that an optimal solution for $(\mathcal{M}, \Phi_G, \Phi_P)$ exists whenever there is a policy π such that $\bar{s} \models^{\pi}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$.

We give some illustrating formulas in our robot example.

- $\psi_g = \mathcal{P}_{[1,1]}(\mathbf{final}(\mathit{boxAt}(1, 5) \vee \mathit{boxAt}(2, 5)))$ mandates that with probability 1 on termination either box 1 or box 2 is in area 5;
- $\psi_1 = \mathcal{P}_{[0.5,1]}(\mathbf{G}(\neg\mathbf{occ}(\mathtt{d}(1, 5))))$ requires that with probability at least 0.5 box 1 is never dropped in area 5;
- $\psi_2 = \mathcal{P}_{[1,1]}(\mathbf{F}(\exists B(\exists P(\mathbf{occ}(\mathtt{p}(B, P))))))$ asks the robot to eventually use its arm to pick-up a box.

An example of P4 is $(\mathcal{R}, \Phi_G, \Phi_P)$ where $\Phi_G = \psi_g$ and $\Phi_P = \psi_1 \ll \psi_2 \ll \mathcal{P}_{[1,1]}(\mathbf{True})$, with $\mathcal{R}$ being the MDP modelling the rail robot. It is clear that there are solutions satisfying ψ_1 and solutions satisfying ψ_2. Only the former are optimal.

5 A Quadratic Programming-Based Planner

In this section we recall the quadratic programming based planner P4Solver introduced in [43] to compute the optimal solution to P4 instance, as shown in Algorithm 1. We start with the most preferred formula ψ_1. If we find a solution, then it is optimal and returned. Otherwise, we continue to the next preferred formula. If we have enumerated all property formulas and have not found a solution, then we say that the P4 instance is unsatisfiable.

The procedure *PolFinder* is the core of the algorithm. It finds the policy π^* for $\mathcal{M}$ satisfying Φ_G and ψ_i. We call this a *sub*-P4, written as the triple $(\mathcal{M}, \Phi_G, \psi_i)$. As we show later in the proof of Theorem 2, *PolFinder* constructs a new MDP via expanding $\mathcal{M}$, Φ_G, and ψ_i simultaneously and thus translates the sub-P4 to a quadratic programming problem. The expansion relies on the concept of the *progression* of property formula.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

Algorithm 1 P4Solver

Input: MDP $\mathcal{M}$, goal formula Φ_G, preference formula $\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$
Output: An optimal policy π^* or "unsatisfiable"

```
π* = ∅
for i = 1 to k do
    π* = PolFinder(M, Φ_G, ψ_i)
    if π* ≠ ∅ then
        return π*
return unsatisfiable
```

Definition 5. *For given property formula φ, $s \in S$ and $a \in Act_\natural$, the* progression $\mathbb{P}^s_a(\varphi)$ *of φ from s through a is defined as:*

$$\mathbb{P}^s_a(p) = \begin{cases} \mathbf{True} & \text{if } s \models p \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\mathbb{P}^s_a(\neg\varphi) = \neg\mathbb{P}^s_a(\varphi)$$

$$\mathbb{P}^s_a(\varphi_1 \wedge \varphi_2) = \mathbb{P}^s_a(\varphi_1) \wedge \mathbb{P}^s_a(\varphi_2)$$

$$\mathbb{P}^s_a(\mathbf{final}(\varphi)) = \begin{cases} \mathbf{final}(\varphi) & \text{if } a \neq \natural \\ \mathbf{True} & \text{if } a = \natural \text{ and } s \models \mathbb{P}^s_\natural(\varphi) \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\mathbb{P}^s_a(\mathbf{X}(\varphi)) = \begin{cases} \varphi & \text{if } a \neq \natural \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\mathbb{P}^s_a(\mathbf{U}(\varphi_1, \varphi_2)) = \begin{cases} \mathbb{P}^s_a(\varphi_2) \vee (\mathbb{P}^s_a(\varphi_1) \wedge \mathbf{U}(\varphi_1, \varphi_2)) & \text{if } a \neq \natural \\ \mathbb{P}^s_\natural(\varphi_2) & \text{otherwise} \end{cases}$$

$$\mathbb{P}^s_a(\mathbf{occ}(a')) = \begin{cases} \mathbf{True} & \text{if } a' = a \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\mathbb{P}^s_a(\forall x(\varphi)) = \bigwedge_{v \in \mathrm{dom}(x)} \mathbb{P}^s_a(\varphi[x/v]).$$

Intuitively, if a path is defined as $\rho = sas_1 \ldots$ and we want to verify whether $\rho \models \varphi$, we can inductively add the information of each state and action until we reach the end of ρ to make the verification concrete. $\mathbb{P}^s_a(\varphi)$ represents what property remains to be verified after having visited state s and performed action a. As an example, consider the formula $\mathbf{F}(\mathbf{occ}(\mathtt{1})) = \mathbf{U}(\mathbf{True}, \mathbf{occ}(\mathtt{1}))$ and $\mathbb{P}^s_{\mathtt{n}}(\mathbf{U}(\mathbf{True}, \mathbf{occ}(\mathtt{1})))$, representing the fact that from s the robot performed action $\mathtt{n}$. By definition, we have that $\mathbb{P}^s_{\mathtt{n}}(\mathbf{U}(\mathbf{True}, \mathbf{occ}(\mathtt{1}))) = \mathbb{P}^s_{\mathtt{n}}(\mathbf{occ}(\mathtt{1})) \vee (\mathbb{P}^s_{\mathtt{n}}(\mathbf{True}) \wedge \mathbf{U}(\mathbf{True}, \mathbf{occ}(\mathtt{1}))) = \mathbf{False} \vee (\mathbf{True} \wedge \mathbf{U}(\mathbf{True}, \mathbf{occ}(\mathtt{1}))) = \mathbf{U}(\mathbf{True}, \mathbf{occ}(\mathtt{1}))$. The following theorem shows that verifying $\rho \models \varphi$ is equivalent to verifying $s_1 \ldots \models \mathbb{P}^s_a(\varphi)$.

Theorem 1. *Given a finite path ρ and a property formula φ,*

$$\rho \models \varphi \Leftrightarrow \begin{cases} \rho \models \mathbb{P}^{first(\rho)}_\natural(\varphi) & \text{if } |\rho| = 0, \\ suffix(\rho, 1) \models \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\varphi) & \text{otherwise.} \end{cases}$$

Proof. The proof is by induction on the length of the path. For the base case, consider a path $\rho \in Paths^*_{\mathcal{M}}$ such that $|\rho| = 0$, i.e., $\rho = s_0$. We now show that $s_0 \models \varphi \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi)$ by induction on the structure of the formula φ.

There are two base cases, namely $\varphi = p$ for some $p \in \Sigma$ and $\varphi = \mathbf{occ}(a)$ for some $a \in Act$. Consider the former case, i.e., $\varphi = p$ for some $p \in \Sigma$; we have that $s_0 \models \varphi \Leftrightarrow s_0 \models p$. By definition of the progression function $\mathbb{P}^{s_0}_\natural(\cdot)$, we have that $\mathbb{P}^{s_0}_\natural(p) = \mathbf{True} \Leftrightarrow s_0 \models p$, which means that $s_0 \models \mathbb{P}^{s_0}_\natural(p) \Leftrightarrow s_0 \models p$, as required.

Consider now the latter base case, i.e., $\varphi = \mathbf{occ}(a)$ for some $a \in Act$; we have that $\mathbb{P}^{s_0}_\natural(\mathbf{occ}(a)) = \mathbf{True} \Leftrightarrow a = \natural$. By the semantics of property formulas, we have that $s_0 \models \mathbf{occ}(a) \Leftrightarrow a = \natural$, so this means that $s_0 \models \mathbb{P}^{s_0}_\natural(\mathbf{occ}(a)) \Leftrightarrow s_0 \models \mathbf{occ}(a)$, as required.

This completes the analysis of the base cases; for the inductive step we assume by inductive hypothesis that, given a formula φ, $s_0 \models \varphi' \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi')$ for each instantiated subformula φ' of φ, that is, either φ' is a subformula of φ or, if $\varphi = \forall x(\psi)$, then $\varphi' = \psi[x/v]$ and $v \in \mathrm{dom}(x)$.

If $\varphi = \neg\varphi'$, by semantics of property formulas and the definition of $\mathbb{P}^{s_0}_\natural(\cdot)$, $s_0 \models \varphi \Leftrightarrow s_0 \not\models \varphi' \Leftrightarrow s_0 \not\models \mathbb{P}^{s_0}_\natural(\varphi') \Leftrightarrow s_0 \models \neg\mathbb{P}^{s_0}_\natural(\varphi') \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\neg\varphi') \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi)$.

If $\varphi = \varphi_1 \wedge \varphi_2$, by the induction hypothesis on φ_1 and φ_2, we have $s_0 \models \varphi_1 \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_1)$ and $s_0 \models \varphi_2 \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_2)$. Thus $s_0 \models \varphi \Leftrightarrow s_0 \models \varphi_1 \wedge \varphi_2 \Leftrightarrow s_0 \models \varphi_1 \bigwedge s_0 \models \varphi_2 \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_1) \bigwedge s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_2) \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_1) \wedge \mathbb{P}^{s_0}_\natural(\varphi_2) \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_1 \wedge \varphi_2) \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi)$.

If $\varphi = \mathbf{final}(\varphi')$, we have that $s_0 = last(s_0)$ and $s_0 \models \mathbf{final}(\varphi') \Leftrightarrow s_0 \models \varphi' \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi')$. By the definition of progression, $\mathbb{P}^{s_0}_\natural(\mathbf{final}(\varphi')) = \mathbf{True} \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi')$, thus $s_0 \models \mathbb{P}^{s_0}_\natural(\mathbf{final}(\varphi')) \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi') \Leftrightarrow s_0 \models \mathbf{final}(\varphi')$.

If $\varphi = \mathbf{X}(\varphi')$, since $|s_0| = 0$, it follows immediately that $s_0 \models \mathbf{X}(\varphi') \Leftrightarrow s_0 \models \mathbf{False}$ for each formula φ'. From definition of progression, we have that $\mathbb{P}^{s_0}_\natural(\mathbf{X}(\varphi')) = \mathbf{False}$, therefore $s_0 \models \mathbf{X}(\varphi') \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\mathbf{X}(\varphi'))$.

If $\varphi = \mathbf{U}(\varphi_1, \varphi_2)$, by induction hypothesis, we have $s_0 \models \varphi_2 \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_2)$. By definition of progression, we have that $\mathbb{P}^{s_0}_\natural(\mathbf{U}(\varphi_1, \varphi_2)) = \mathbb{P}^{s_0}_\natural(\varphi_2)$. Therefore we have $s_0 \models \varphi \Leftrightarrow s_0 \models \mathbf{U}(\varphi_1, \varphi_2) \Leftrightarrow s_0 \models \varphi_2$ or $s_0 \models \varphi_1 \wedge \mathbf{X}(\mathbf{U}(\varphi_1, \varphi_2)) \Leftrightarrow s_0 \models \varphi_2 \bigvee s_0 \models \varphi_1 \wedge \mathbf{False} \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi_2) \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\mathbf{U}(\varphi_1, \varphi_2)) \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi)$.

If $\varphi = \forall x(\varphi')$, the proof is similar to the case $\varphi = \varphi_1 \wedge \varphi_2$.

Now the base case $s_0 \models \varphi \Leftrightarrow s_0 \models \mathbb{P}^{s_0}_\natural(\varphi)$ of the induction has been completed. For the induction step, we need to prove that for each finite path ρ such that $|\rho| > 0$, $\rho \models \varphi \Leftrightarrow suffix(\rho, 1) \models \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\varphi)$. As before, this is done by induction on the structure of the formula φ.

For any φ such that $len(\varphi) = 0$, by definition, we have that $\varphi = p$ for some $p \in \Sigma$, or that $\varphi = \mathbf{occ}(a)$ for some $a \in Act$. If $\varphi = p$ for some $p \in \Sigma$, we have $\rho \models p \Leftrightarrow first(\rho) \models p \Leftrightarrow \mathbb{P}^{first(\rho)}_{action(\rho,1)}(p) = \mathbf{True} \Leftrightarrow suffix(\rho, 1) \models \mathbb{P}^{first(\rho)}_{action(\rho,1)}(p)$. If $\varphi = \mathbf{occ}(a)$ for some $a \in Act$, we have $\rho \models \mathbf{occ}(a) \Leftrightarrow action(\rho, 1) = a \Leftrightarrow \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\mathbf{occ}(a)) = \mathbf{True} \Leftrightarrow suffix(\rho, 1) \models \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\mathbf{occ}(a))$.

We assume that for any property formula ψ that $len(\psi) \leq n$, $\rho \models \psi \Leftrightarrow suffix(\rho, 1) \models \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\psi)$. For any property formula φ such that $len(\varphi) = n + 1$, we prove $\rho \models \varphi \Leftrightarrow suffix(\rho, 1) \models \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\varphi)$. For reading convenience, we define $s_0 \equiv first(\rho)$, $a_1 \equiv action(\rho, 1)$ and $\rho_1 \equiv suffix(\rho, 1)$, so that $\rho = s_0 a_1 \rho_1$. Then the assertion we need to prove is simplified as $\rho \models \varphi \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\varphi)$. Notice that we always have $a_1 \neq \natural$.

If $\varphi = \neg\psi$, it is easy to see $len(\psi) = n$. Thus $\rho \models \neg\psi \Leftrightarrow \rho \not\models \psi \Leftrightarrow \rho_1 \not\models \mathbb{P}^{s_0}_{a_1}(\psi) \Leftrightarrow \rho_1 \models \neg\mathbb{P}^{s_0}_{a_1}(\psi) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\neg\psi)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

If $\varphi = \psi_1 \wedge \psi_2$, $len(\psi_1) \leq n$, and $len(\psi_2) \leq n$, then $\rho \models \psi_1 \wedge \psi_2 \Leftrightarrow \rho \models \psi_1 \wedge \rho \models \psi_2 \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_1) \wedge \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_2) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_1) \wedge \mathbb{P}^{s_0}_{a_1}(\psi_2) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_1 \wedge \psi_2)$.

If $\varphi = \mathbf{final}(\psi)$, $len(\psi) = n$. According to the fact that $\rho \models \mathbf{final}(\psi) \Leftrightarrow last(\rho) \models \psi$ and $last(\rho) = last(\rho_1)$, we have $last(\rho) \models \psi \Leftrightarrow last(\rho_1) \models \psi \Leftrightarrow \rho_1 \models \mathbf{final}(\psi)$. Since $a_1 \neq \natural$, we have $\mathbb{P}^{s_0}_{a_1}(\mathbf{final}(\psi)) = \mathbf{final}(\psi)$ by the definition of progression. Therefore, $\rho \models \mathbf{final}(\psi) \Leftrightarrow \rho_1 \models \mathbf{final}(\psi) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\mathbf{final}(\psi))$.

If $\varphi = \mathbf{X}(\psi)$, in the proof we do not limit the length of ψ to be smaller than $n+1$. For ψ with arbitrary length, because $|\rho| > 0$, we can immediately get $\rho \models \mathbf{X}(\psi) \Leftrightarrow \rho_1 \models \psi$. Since $a_1 \neq \natural$, by the definition of progression, we have $\mathbb{P}^{s_0}_{a_1}(\mathbf{X}(\psi)) = \psi$. Thus $\rho \models \mathbf{X}(\psi) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi)$.

If $\varphi = \mathbf{U}(\psi_1, \psi_2)$, $len(\psi_1) \leq n$ and $len(\psi_2) \leq n$. We can decompose $\mathbf{U}(\psi_1, \psi_2)$ as $\mathbf{U}(\psi_1, \psi_2) = \psi_2 \vee (\psi_1 \wedge \mathbf{X}(\mathbf{U}(\psi_1, \psi_2)))$. Thus we have $\rho \models \mathbf{U}(\psi_1, \psi_2) \Leftrightarrow \rho \models \psi_2 \vee (\psi_1 \wedge \mathbf{X}(\mathbf{U}(\psi_1, \psi_2))) \Leftrightarrow \rho \models \psi_2 \vee (\rho \models \psi_1 \wedge \rho \models \mathbf{X}(\mathbf{U}(\psi_1, \psi_2))) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_2) \vee (\rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_1) \wedge \mathbf{U}(\psi_1, \psi_2)) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\psi_2) \vee (\mathbb{P}^{s_0}_{a_1}(\psi_1) \wedge \mathbf{U}(\psi_1, \psi_2)) \Leftrightarrow \rho_1 \models \mathbb{P}^{s_0}_{a_1}(\mathbf{U}(\psi_1, \psi_2))$.

If $\varphi = \forall x(\psi)$, the proof is similar to case $\varphi = \psi_1 \wedge \psi_2$. □

In most realistic problems, when a system reaches again the same state s with the same task to be performed, it is reasonable to expect that the policy guides the system to repeat again the previous actions to complete the task; for tasks modeled as bounded properties, the bound decreases between two visits of s, so it is reasonable to expect changes in the policy. Consider again the rail robot example. Whenever the robot wants to go from area 2 to area 3, it always chooses to execute `n` and does not need to try `l`. Such requirement is especially important for composed systems with coactive agents. For example, we design policies for robots cooperating to make cars in a plant: robots should operate so to fulfill their specific tasks, thus robots should not behave arbitrarily. We call such policies *property-dependent*, and for sub-P4 we focus on finding the property-dependent policies. The class of property-dependent policies is an extension of memoryless policies, and a subclass of history-dependent policies. To formally define property-dependent policies, we need to extend progression over states to a relative concept of *progression function* over finite paths.

Definition 6. *The* progression function *over $Paths^*$ is defined recursively as: if $|\rho| = 0$, $\mathtt{Prog}(\rho, \varphi) = \varphi$; otherwise* $\mathtt{Prog}(\rho, \varphi) = \mathtt{Prog}(suffix(\rho, 1), \mathbb{P}^{first(\rho)}_{action(\rho,1)}(\varphi))$.

It is easy to see that the progression function $\mathtt{Prog}(\rho, \varphi)$ is an extension of progression $\mathbb{P}^s_a(\varphi)$. It progresses φ with respect to sequence of states and actions of ρ until the last state of ρ is reached. Now we formalize property-dependent policies as follows.

Definition 7. *Given the sub-*P4 $(\mathcal{M}, \Phi_G, \psi)$*, a policy π is a* property-dependent policy *if for each $\rho_1, \rho_2 \in Paths^*$ such that $last(\rho_1) = last(\rho_2)$, it is $\pi(\rho_1) = \pi(\rho_2)$ whenever $\mathtt{Prog}(\rho_1, \varphi) = \mathtt{Prog}(\rho_2, \varphi)$, given $\psi = \mathcal{P}_J(\varphi)$.*

We now recall the definition of *quadratically constrained programming problem* (QCPP) from operations research. Then we show how to express a sub-P4 as a QCPP one to find a best property-dependent policy.

Definition 8. *A* quadratically constrained programming problem *(*QCPP*) with n variables is a problem of the form*

$$X^T Q_i X + q_i^T X + r_i \leq 0 \quad \forall i = 1, \ldots, m$$
$$AX = b$$

where $X \in \mathbb{R}^{n \times 1}$ is the vector of optimization variables and $Q_i \in \mathbb{R}^{n \times n}$, $q_i \in \mathbb{R}^{n \times 1}$, $r_i \in \mathbb{R}$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^{k \times 1}$ are the known coefficients.

As an example of QCPP, consider for instance the problem:

$$X^T \mathbf{1}_i X - 1 \leq 0 \quad \forall i = 1, \ldots, n$$
$$-X^T \mathbf{1}_i X + 1 \leq 0 \quad \forall i = 1, \ldots, n$$

where $\mathbf{1}_i$ is the zero matrix with 1 in position (i, i). Each solution X is such that X_i is either -1 or 1 and it is used in the encoding of the MAX-CUT graph problem as a quadratically constrained quadratic programming problem, i.e., a QCPP extended with a quadratic objective function.

Before encoding the sub-P4 $(\mathcal{M}, \mathcal{P}_{J_g}(\mathbf{final}(\gamma)), \mathcal{P}_{J_i}(\varphi_i))$ as a QCPP, we denote by $p^{\varphi}_{s,a}$ the probability of choosing the action $a \in Act_{\natural}(s)$ in state s when the progressed preference formula is φ (which corresponds to the property-dependent policy we are looking for), by $\theta_{s,\varphi,\eta}$ the probability to satisfy the goal property formula η with respect to the progressed preference formula φ from state s, and by $\mu_{s,\varphi}$ the probability to satisfy the progressed preference formula φ from state s. To satisfy the preference formula and goal formula, for $\bar{s}$ the initial state of $\mathcal{M}$, we have the constraints

$$\theta_{\bar{s},\varphi_i,\mathbf{final}(\gamma)} \in J_g \qquad \text{and} \qquad \mu_{\bar{s},\varphi_i} \in J_i.$$

Then we expand each $\theta_{s,\varphi,\eta}$ and $\mu_{s,\varphi}$ as

$$\begin{cases} \theta_{s,\varphi,\eta} = \sum_{a \in Act_{\natural}(s)} p^{\mathbb{P}^s_a(\varphi)}_{s,a} \cdot \sum_{s' \in S} \mathrm{P}(s,a)(s') \cdot \theta_{s',\mathbb{P}^s_a(\varphi),\mathbb{P}^s_a(\eta)} \\ \mu_{s,\varphi} = \sum_{a \in Act_{\natural}(s)} p^{\mathbb{P}^s_a(\varphi)}_{s,a} \cdot \sum_{s' \in S} \mathrm{P}(s,a)(s') \cdot \mu_{s',\mathbb{P}^s_a(\varphi)} \\ \theta_{s,\varphi,\mathbf{True}} = 1,\ \theta_{s,\varphi,\mathbf{False}} = 0,\ \mu_{s,\mathbf{True}} = 1,\ \mu_{s,\mathbf{False}} = 0 \end{cases} \tag{1}$$

where we set $\mathrm{P}(s, \natural)(s') = 1$ for $s = s'$ and $\mathrm{P}(s, \natural)(s') = 0$ for $s \neq s'$ to make the notation compact. This expansion is used by the procedure $PolFinder$.

As an example of the above expansion, consider state s and formula $\mathbf{F}(\mathbf{occ}(\mathtt{1}))$ we have seen below Definition 5. Then we have $p^{\mathbb{P}^s_1(\mathbf{F}(\mathbf{occ}(\mathtt{1})))}_{s,\mathtt{1}} = p^{\mathbf{True}}_{s,\mathtt{1}}$ while $p^{\mathbb{P}^s_a(\mathbf{F}(\mathbf{occ}(\mathtt{1})))}_{s,a} = p^{\mathbf{F}(\mathbf{occ}(\mathtt{1}))}_{s,a}$ for each $a \neq \mathtt{1}$. Hence, we have the equality $\mu_{s,\mathbf{F}(\mathbf{occ}(\mathtt{1}))} = \left(p^{\mathbf{True}}_{s,\mathtt{1}} \cdot \sum_{s' \in S} \mathrm{P}(s,l)(s') \cdot \mu_{s',\mathbf{True}}\right) + \sum_{a \in Act_{\natural}(s) \setminus \{\mathtt{1}\}} p^{\mathbf{F}(\mathbf{occ}(\mathtt{1}))}_{s,a} \cdot \sum_{s' \in S} \mathrm{P}(s,a)(s') \cdot \mu_{s',\mathbf{F}(\mathbf{occ}(\mathtt{1}))}$.

$PolFinder$, from the initial state $\bar{s}$ of $\mathcal{M}$ and input formulas $\mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ and $\mathcal{P}_{J_i}(\varphi_i)$, explores the state space of $\mathcal{M}$ while progressing both $\mathbf{final}(\gamma)$ and φ_i. For each newly reached tuple of state and progressed formulas η and φ (from the initial $\mathbf{final}(\gamma)$ and φ_i, respectively), it generates the corresponding instance for $\theta_{s,\varphi,\eta}$ and $\mu_{s,\varphi}$ according to Equation (1). In the bounded version of $PolFinder$, we also consider the number of past actions in the definition of Equation (1), by means of a counter decorating $\theta_{s,\varphi,\eta}$ and $\mu_{s,\varphi}$ that is decreased in the right-hand side of Equation (1).

It is easy to see that if we choose as X the list of all variables for probabilities $\theta_{s,\varphi,\eta}$, $\mu_{s,\varphi}$, and $p^{\varphi}_{s,a}$ appearing in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

the progress shown in Equation (1), then the constraints corresponding to Equation (1) for each state s and progressed formulas η and φ and the constraints for the probability values (like $p^{\mathbb{P}^s_a(\varphi)}_{s,a} \geq 0$ and $\sum_{a\in Act(s)} p^{\mathbb{P}^s_a(\varphi)}_{s,a} = 1$) can be re-written to a quadratic and linear form of X, respectively. Thus a sub-P4 can be encoded to a QCPP problem.

When solving sub-P4, we construct the equations iteratively for each $\mu_{s,\varphi}$ and with the form above until the (simplified) progression for φ becomes **True** or **False**, so we can definitely know the probability of the formula to be 1 or 0, respectively, and similarly for $\theta_{s,\varphi,\eta}$. If there exists a solution to the corresponding set of formulas to the sub-P4, the values $p^{\varphi}_{s,a}$ induce a solution of P4. So P4 can be converted into a set of quadratic equations whose solutions, if any, are the solutions for P4.

Clearly, the correctness of P4Solver relies on the termination of $PolFinder$. Since we consider only finite MDPs and a finite set of atomic propositions Σ, $PolFinder$ terminates for sure, since there are only a finite number of combinations of state s and progressed formulas η and φ for which generate new variables $\theta_{s,\varphi,\eta}$ and $\mu_{s,\varphi}$. As complexity of $PolFinder$, for an MDP $\mathcal{M}$ and a preference property formula φ, we have that first $PolFinder$ generates the QCPP, which is in PSPACE, similarly to [7]. Then it needs to solve it, which is in general NP-hard since the NP-complete 0-1 integer programming [33] can be reduced to QCPP. Note that the goal formula η adds only a constant factor to the complexity, since the progression of $\mathbf{final}(\gamma)$ is either $\mathbf{final}(\gamma)$ itself, **True**, or **False** (cf. Definition 5).

Theorem 2. *Given a* P4 *instance* $(\mathcal{M}, \Phi_G, \Phi_P)$*, it has an optimal property-dependent policy* π *making* ψ_j *in* Φ_P *the earliest probabilistic property formula to be satisfied if and only if* P4Solver *returns a policy* π^* *also making* ψ_j *in* Φ_P *the earliest probabilistic property formula to be satisfied.*

Proof. The key to prove the correctness of P4Solver is to prove the correctness of $PolFinder$. Suppose that Φ_G is of the form $\Phi_G = \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ where γ is a property formula.

For the first step, given an MDP $\mathcal{M} = (S, \Sigma, L, Act, \bar{s}, \mathrm{P})$ and two property formulas ψ and $\mathbf{final}(\gamma)$, we construct the product of these three components as the new MDP

$$\mathcal{M}_{\otimes} = (S_{\otimes}, \Sigma_{\otimes}, L_{\otimes}, Act_{\otimes}, \bar{s}_{\otimes}, \mathrm{P}_{\otimes})$$

according to the following rules.

1) $\bar{s}_{\otimes} = (\bar{s}, \psi, \mathbf{final}(\gamma)) \in S_{\otimes}$.
2) $\Sigma_{\otimes} = \Sigma$.
3) $Act_{\otimes} = Act$.
4) For each $(s, \varphi, \eta) \in S_{\otimes}$, it is $L_{\otimes}((s, \varphi, \eta)) = L(s)$.
5) For each $(s, \varphi, \eta) \in S_{\otimes}$, it is

$$Act_{\otimes}((s, \varphi, \eta)) = \begin{cases} \emptyset & \text{if } \eta \in \{\mathbf{True}, \mathbf{False}\}, \\ Act(s) & \text{otherwise.} \end{cases}$$

6) For each state $(s, \varphi, \eta) \in S_{\otimes}$ and action $a \in Act(s)$, if $\mathrm{P}(s, a)(s') > 0$, then $(s', \mathbb{P}^s_a(\varphi), \mathbb{P}^s_a(\eta)) \in S_{\otimes}$ and $\mathrm{P}_{\otimes}((s, \varphi, \eta), a)((s', \mathbb{P}^s_a(\varphi), \mathbb{P}^s_a(\eta))) = \mathrm{P}(s, a)(s')$.

From the construction above and the definition of progression function over finite paths, we can see that each state of $\mathcal{M}_{\otimes}$ is of the form

$$(last(\rho), \mathtt{Prog}(\rho, \psi), \mathtt{Prog}(\rho, \mathbf{final}(\gamma)))$$

for $\rho \in Paths^*$ of $\mathcal{M}$. We assert that $\mathcal{M}_{\otimes}$ is finite, since S and the closure of subformulas of ψ and $\mathbf{final}(\gamma)$ are finite.

With the proof of Theorem 1, we can see that (1) is the Bellman equation of $\mathcal{M}_{\otimes}$, where $\mu_{s,\varphi}$ represents the probability that the state (s, φ, η) satisfies the property formula φ, and $\theta_{s,\varphi,\eta}$ represents the probability that the state (s, φ, η) satisfies the goal formula η. For arbitrary property-dependent policy π of $\mathcal{M}$ and property formula Δ, under the construction above, we can contrive a corresponding history-independent policy $\pi_{\otimes}$ of $\mathcal{M}_{\otimes}$ as

$$\pi_{\otimes}(last(\rho), \mathtt{Prog}(\rho, \psi), \mathtt{Prog}(\rho, \mathbf{final}(\gamma))) = \pi(\rho) \tag{2}$$

for all $\rho \in Paths^*$, such that

$$\begin{aligned} &\mu_{\pi,\bar{s}}(\{\, \rho \in Paths^*_{\mathcal{M}} \mid \rho \models \Delta \,\}) \\ &= \mu_{\pi_{\otimes},(\bar{s},\psi,\mathbf{final}(\gamma))}(\{\, \rho_{\otimes} \in Paths^*_{\mathcal{M}_{\otimes}} \mid \rho_{\otimes} \models \Delta \,\}). \end{aligned} \tag{3}$$

On the other hand, because $\mathcal{M}_{\otimes}$ is a simulation of parsing the satisfaction of ψ and $\mathbf{final}(\gamma)$ in $\mathcal{M}$, each history-independent policy $\pi_{\otimes}$ for $\mathcal{M}_{\otimes}$ is a frame of a policy π for $\mathcal{M}$ constructed via (2), making (3) naturally hold. We only need to prove that such a policy π is property-dependent.

Let $\rho_1, \rho_2 \in Paths^*$ be two finite paths such that $last(\rho_1) = last(\rho_2)$ and $\mathtt{Prog}(\rho_1, \psi) = \mathtt{Prog}(\rho_2, \psi)$, so to satisfy the conditions of property-dependent policy in Definition 7. From the semantics of progression function, we have that $\mathtt{Prog}(\rho, \mathbf{final}(\gamma)) = \mathtt{Prog}(last(\rho), \gamma)$, indicating that $\mathtt{Prog}(\rho_1, \mathbf{final}(\gamma)) = \mathtt{Prog}(\rho_2, \mathbf{final}(\gamma))$. Since $\pi_{\otimes}$ is a history-independent policy for $\mathcal{M}_{\otimes}$, we can immediately have that $\pi_{\otimes}(last(\rho_1), \mathtt{Prog}(\rho_1, \psi), \mathtt{Prog}(\rho_1, \mathbf{final}(\gamma))) = \pi_{\otimes}(last(\rho_2), \mathtt{Prog}(\rho_2, \psi), \mathtt{Prog}(\rho_2, \mathbf{final}(\gamma)))$. Thus, from the construction of π according to (2), we have that $\pi(\rho_1) = \pi(\rho_2)$, showing that π is property-dependent.

Now the relationship between $\mathcal{M}$ and $\mathcal{M}_{\otimes}$ is clear: $\mathcal{M}_{\otimes}$ allows us to compute the satisfaction probability of both ψ and $\mathbf{final}(\gamma)$ in $\mathcal{M}$ under the same policy $\pi_{\otimes}$. By means of Equation (2), we derive the corresponding policy for $\mathcal{M}$. Thus the correctness of $PolFinder$ (and hence of P4Solver in finding an optimal policy) is guaranteed. □

6 A Model Checking-Based Planner

P4Solver transforms an instance of P4 to a QCPP. Unfortunately, P4Solver does not scale for the following reasons. Firstly, P4Solver has to expand the model starting from the initial state several times until all possible needed combinations of reached states and progressed goal and preference formulas have been considered. Secondly, the policy is obtained from the solution of the generated QCPP that is known to be NP-hard in general, and the size of the generated QCPP depends on the expansion. As we will see in Section 7, P4Solver already fails on rather small case studies. In this section, we present a multi-objective model checking based algorithm which improves the efficiency of solving P4 instances.

Consider P4 with an MDP $\mathcal{M} = (S, \Sigma, L, Act, \bar{s}, \mathrm{P})$, a policy π, a goal formula $\Phi_G = \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ where γ is a property formula, and a probabilistic property formula ψ_i from a preference formula $\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$. Some differences exist between P4 and classical probability model checking problem. As a planning problem defined

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

over finite paths, P4 allows $\mathbf{final}(\cdot)$ to describe the characteristics of the terminating states. Besides, $\mathbf{occ}(\cdot)$ is also not defined in classical model checking problem. In this section we explain how we encode termination and action occurrence in given P4 into the MDP, and how we encode the property formulas appearing in Φ_P and Φ_G into standard PLTL ones, so that the encoded P4 is equivalent to its origin.

6.1 Encoding of Termination

According to the semantics of $\bar{s} \models^{\pi}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$, the goal is satisfied if the probability of the finite paths ending with a state satisfying γ belongs to J_g; similarly, if $\mathbf{final}(\gamma')$ appears as a subformula of ψ_i, then it holds only on finite paths whose last state satisfies γ'.

Classical probabilistic model checking usually considers infinite paths, however P4 is defined for finite paths. Moreover, classical PLTL does not include the $\mathbf{final}(\cdot)$ operator, so PLTL model checkers do not support it natively. To use classical model checkers, we encode the termination into the MDP itself by enriching the given MDP with the information about termination. Intuitively, for each state s, we create a corresponding state $s_\perp$, and copy the label of s for $s_\perp$. Finally, we add a new transition from s to $s_\perp$ with $\perp$ action such that $(s, \perp, \delta_{s_\perp})$ and add a self-loop $(s_\perp, \perp, \delta_{s_\perp})$ to represent explicitly the choice of stopping. In this way, once $\perp$ is chosen, we freeze the status of the MDP so that the satisfaction of the formulas can be decided.

Formally, we consider the MDP $\mathcal{M}_{\zeta_\perp} = \zeta_\perp(\mathcal{M}) = (S', \Sigma', L', Act', \bar{s}, \mathrm{P}')$ where:

- $S' = S \cup \{ s_\perp \mid s \in S \}$,
- $\Sigma' = \Sigma \cup \{\perp\}$,
- $L' = L \cup \{ (s_\perp, L(s) \cup \{\perp\}) \mid s \in S \}$,
- $Act' = Act \cup \{\perp\}$, and
- $\mathrm{P}' = \mathrm{P} \cup \{ (s, \perp, \delta_{s_\perp}), (s_\perp, \perp, \delta_{s_\perp}) \mid s \in S \}$.

Note that when we represent the MDP symbolically, such as a factored MDP or by means of RDDL [50], the above construction reduces to the following one: we add a new Boolean variable `running` initialized to true and an action $\perp$ that sets `running` to false with probability 1. We then modify the original transitions by enabling them only when `running` is true.

6.2 Encoding of Action Occurrence

Besides the $\mathbf{final}(\cdot)$ operator, classical PLTL does not include the $\mathbf{occ}(\cdot)$ operator that holds whenever the first action of the considered path is the required one. We encode the information needed by $\mathbf{occ}(\cdot)$ by storing the action in the states reached by performing the transition. Let A be the set of actions a such that $\mathbf{occ}(a)$ occurs in ψ_i. We encode the occurrence of each action $a \in A$ by taking a copy s_a of the states, labeling s_a with the label of s extended with a (that now is also an atomic proposition), and modifying each transition (s, a, μ) so that it leaves s and reaches each state t_a with probability $\mu(t)$. Note that we keep the original initial state $\bar{s}$ and we modify its transitions only with respect to the reached states.

Formally, we consider the MDP $\mathcal{M}_{\zeta_A} = \zeta_A(\mathcal{M}) = (S', \Sigma', L', Act, \bar{s}, \mathrm{P}')$ constructed as follows:

- $S' = S \cup \{ s_a \mid s \in S, a \in A \}$,
- $\Sigma' = \Sigma \cup A$,
- $L' = L \cup \{ (s_a, L(s) \cup \{a\}) \mid s \in S, a \in A \}$, and
- $\mathrm{P}' = \{ (s, b, \mu) \in \mathrm{P} \mid b \notin A \} \cup \{ (s, a, \mu_{\zeta_a}) \mid (s, a, \mu) \in \mathrm{P}, a \in A \} \cup \{ (s_a, b, \mu) \mid (s, b, \mu) \in \mathrm{P}, a \in A, b \notin A \} \cup \{ (s_{a'}, a, \mu_{\zeta_a}) \mid (s, a, \mu) \in \mathrm{P}, a, a' \in A \}$,

where for each $\mu \in \mathit{Dist}(S)$ and $a \in A$, $\mu_{\zeta_a} \in \mathit{Dist}(S')$ is defined for each $s' \in S'$ as follows:

$$\mu_{\zeta_a}(s') = \begin{cases} \mu(t) & \text{if } s' = t_a, \\ 0 & \text{otherwise.} \end{cases}$$

Similarly to the encoding of termination, the above construction can be obtained symbolically as follows: we add a new variable `act` ranging over $A \cup \{\varepsilon\}$ with initial value ε and we modify each transition with action a so that it also sets `act` to a if $a \in A$, to ε otherwise.

6.3 Encoding of Both Termination and Action Occurrence

Since we are interested in both termination and action occurrence, we can apply in sequence $\zeta_A(\cdot)$ and then $\zeta_\perp(\cdot)$. In this way we manage correctly the labels associated to the states: suppose that we first apply $\zeta_A(\cdot)$ and then $\zeta_\perp(\cdot)$; consider a state s_a of $\mathcal{M}_{\zeta_A}$: its label is $L(s) \cup \{a\}$. When we apply $\zeta_\perp(\cdot)$, its counterpart is $s_{a\perp}$ with label $(L(s) \cup \{a\}) \cup \{\perp\}$, i.e., we have both the label of the last action (a) and the information that we stopped ($\perp$), so we keep the information about the last action we performed before stopping, since $\perp$ is not an action of $\mathcal{M}_{\zeta_A}$. This information is fundamental when we want to check $\mathbf{G}(\mathbf{occ}(a))$ which holds on a finite path ρ only when all its actions are a.

If we apply $\zeta_\perp(\cdot)$ before $\zeta_A(\cdot)$, the stopping states would never have the information about the last actual action we performed, so the resulting MDP is not the expected frozen status of the original MDP. This will lead to problems with the evaluation of the encoded LTL formulas. As we will see below, the encoding of $\mathbf{occ}(a)$ is $\mathbf{X}(a)$ and $\mathbf{G}(\mathbf{X}(a))$ is not satisfied by the infinite path $\rho \perp s \perp s \ldots$ where s is the copy of $\mathit{last}(\rho)$ according to $\zeta_\perp(\cdot)$, since $a \notin L(s)$.

6.4 Encoding of Probabilistic Property Formulas as PLTL Formulas

To complete the encoding of P4 as a classical model checking problem, we still have to encode the probabilistic property formulas underlying the preference formulas as ordinary PLTL formulas. The main operators we have to consider are $\mathbf{occ}(\cdot)$, $\mathbf{final}(\cdot)$ and $\forall x(\cdot)$: according to their semantics, $\mathbf{occ}(a)$ holds if a is the next action we perform, so we encode it as $\mathbf{X}(a)$; $\mathbf{final}(\varphi)$ holds if φ holds when we stop, so we encode it as $\mathbf{F}(\perp \wedge \varphi')$ where φ' is obtained from φ by replacing each temporal operator with its semantic value on the last state of the path, i.e., $\mathbf{X}(\cdot)$ by $\mathbf{False}$ and $\mathbf{U}(\cdot, \psi)$ by ψ'; and $\forall x(\varphi)$ is trivial since it is essentially a shortcut for $\bigwedge_{v \in \mathrm{dom}(x)} \varphi[x/v]$. Formally, we encode a probabilistic property formula $\mathcal{P}_J(\varphi)$ as a PLTL formula $\mathcal{P}_J(\varphi')$ by means of the function $\xi_{A,\perp} \colon \mathcal{P}_J(\varphi) \mapsto \mathcal{P}_J(\xi_{A,\perp}(\varphi, \mathbf{True}))$ where $\xi_{A,\perp}(\varphi, c)$, with c being a Boolean, is defined inductively as follows:

$$\xi_{A,\perp}(p, c) = p$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

$$\begin{aligned}
\xi_{A,\perp}(\neg\varphi, c) &= \neg\xi_{A,\perp}(\varphi, c)\\
\xi_{A,\perp}(\varphi \wedge \psi, c) &= \xi_{A,\perp}(\varphi, c) \wedge \xi_{A,\perp}(\psi, c)\\
\xi_{A,\perp}(\mathbf{final}(\varphi), c) &= \mathbf{F}(\perp \wedge \xi_{A,\perp}(\varphi, \mathbf{False}))\\
\xi_{A,\perp}(\mathbf{X}(\varphi), c) &= c \wedge \mathbf{X}(\xi_{A,\perp}(\varphi, c))\\
\xi_{A,\perp}(\mathbf{U}(\varphi, \psi), \mathbf{False}) &= \xi_{A,\perp}(\psi, \mathbf{False})\\
\xi_{A,\perp}(\mathbf{U}(\varphi, \psi), \mathbf{True}) &= \mathbf{U}(\xi_{A,\perp}(\varphi, \mathbf{True}), \xi_{A,\perp}(\psi, \mathbf{True}))\\
\xi_{A,\perp}(\mathbf{occ}(a), c) &= c \wedge \mathbf{X}(a)\\
\xi_{A,\perp}(\forall x(\varphi), c) &= \forall x(\xi_{A,\perp}(\varphi, c)).
\end{aligned}$$

The parameter c of $\xi_{A,\perp}(\cdot, c)$ is used to keep track of the behavior of the temporal operators when they occur inside the $\mathbf{final}(\cdot)$ operator: since $\mathbf{final}(\varphi)$ holds whenever φ holds on stopping, $\mathbf{X}(\cdot)$ is never satisfied (there is no next state) and $\mathbf{U}(\varphi, \psi)$ holds only if ψ already holds. The extension to preference formulas and P4 is straightforward.

We remark that bounded properties can be encoded as well: $\mathbf{U}^{\leq i}(\varphi, \psi)$ requires that ψ holds within i time steps for the first time and φ holds at every state before. We can inductively encode $\mathbf{U}^{\leq i}(\varphi, \psi)$ as a formula involving the $\mathbf{X}(\cdot)$ operator only with the observation that $\mathbf{U}^{\leq 0}(\varphi, \psi) = \psi$ and $\mathbf{U}^{\leq i}(\varphi, \psi) = \psi \vee (\varphi \wedge \mathbf{X}(\mathbf{U}^{\leq i-1}(\varphi, \psi)))$ for $i \geq 1$.

6.5 Equivalence of P4 and its Encoding

Consider the P4 instance $(\mathcal{M}, \Phi_G, \Phi_P)$; in the following, to simplify the notation, we write $\mathcal{M}_{A,\perp}$ for $\zeta_\perp(\zeta_A(\mathcal{M}))$ and $\varphi_{A,\perp}$ for $\xi_{A,\perp}(\varphi)$. It is easy to show that P4 has an optimal solution if and only if its encoding has an optimal solution, where the earliest satisfied preference is the same. The proof is based on the construction of the policy for the encoding of P4 given the optimal solution for P4 and vice-versa. Two important properties are essential for the proof. The first one, stated as Lemma 3 in Appendix A, is that for each $\rho \in Paths^*_{\mathcal{M}}$ and each property formula φ, we have that $\rho \models \varphi$ if and only if $\rho_{A,\perp} \models \xi_{A,\perp}(\varphi, \mathbf{True})$. The second one, stated as Corollaries 2 and 3 in Appendix A, is that there exists a policy $\pi_{A,\perp}$ for $\mathcal{M}_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$ and $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi'_{A,\perp}$ if and only if there exists a policy π for $\mathcal{M}$ so that $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$ and $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi'$. By means of these two properties, we can obtain the following theorem.

Theorem 3. *Let $(\mathcal{M}, \Phi_G, \Phi_P)$ be an instance of P4. Its encoded P4 $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$ has an optimal policy $\pi^*_{A,\perp}$ making $\psi_{j\,A,\perp}$ in $\Phi_{PA,\perp}$ the earliest probabilistic property formula to be satisfied if and only if $(\mathcal{M}, \Phi_G, \Phi_P)$ has an optimal policy π^* also making ψ_j in Φ_P the earliest probabilistic property formula to be satisfied.*

Proof. We first show that if $(\mathcal{M}, \Phi_G, \Phi_P)$ has an optimal policy, so does $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$.

Let π^* be an optimal policy for $(\mathcal{M}, \Phi_G, \Phi_P)$, where $\Phi_G = \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ and $\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$ is a preference formula. By definition of optimal policy, π^* satisfies that $\bar{s} \models^{\pi^*}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$, $\bar{s} \models^{\pi^*}_{\mathcal{M}} \psi_i$, and for each $0 < j < i$ and each solution π, $\bar{s} \not\models^{\pi}_{\mathcal{M}} \psi_j$. By Corollary 2 in Appendix A, there exists a policy $\pi^*_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi^*_{A,\perp}}_{\mathcal{M}_{A,\perp}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))_{A,\perp}$ and $\bar{s}_{A,\perp} \models^{\pi^*_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{i\,A,\perp}$, thus $\pi^*_{A,\perp}$ is a solution for $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$.

We claim that $\pi^*_{A,\perp}$ is an optimal solution satisfying $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{i\,A,\perp}$ but not satisfying $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{j\,A,\perp}$ for each $0 < j < i$. Suppose, for the sake of contradiction, that there exists a policy $\pi'_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))_{A,\perp}$ and $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{j\,A,\perp}$ for some $0 < j < i$. This implies, by Corollary 3 in Appendix A, that there exists a policy π' such that $\bar{s} \models^{\pi'}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ and $\bar{s} \models^{\pi'}_{\mathcal{M}} \psi_j$, that is, π' is a solution for $(\mathcal{M}, \Phi_G, \Phi_P)$ such that $\bar{s} \models^{\pi}_{\mathcal{M}} \psi_j$. This contradicts the assumption that π^* is an optimal solution for $(\mathcal{M}, \Phi_G, \Phi_P)$, thus $\pi^*_{A,\perp}$ is indeed an optimal solution for $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$, as required.

Now we show that if $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$ has an optimal policy, so does $(\mathcal{M}, \Phi_G, \Phi_P)$, both satisfying the same corresponding probabilistic property formulas.

For $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$, let $\pi^*_{A,\perp}$ be an optimal policy. By definition of optimal policy, $\pi^*_{A,\perp}$ satisfies that $\bar{s}_{A,\perp} \models^{\pi^*_{A,\perp}}_{\mathcal{M}_{A,\perp}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))_{A,\perp}$, $\bar{s}_{A,\perp} \models^{\pi^*_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{i\,A,\perp}$, and for each $0 < j < i$ and each solution $\pi_{A,\perp}$, $\bar{s}_{A,\perp} \not\models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{j\,A,\perp}$. By Corollary 3, there exists a policy π^* such that $\bar{s} \models^{\pi^*}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ and $\bar{s} \models^{\pi^*}_{\mathcal{M}} \psi_i$, thus π^* is a solution for $(\mathcal{M}, \Phi_G, \Phi_P)$.

We claim that π^* is optimal by satisfying $\bar{s} \models^{\pi'}_{\mathcal{M}} \psi_i$ as the earliest probabilistic property formula. Suppose, for the sake of contradiction, that there exists a policy π' such that $\bar{s} \models^{\pi'}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ and $\bar{s} \models^{\pi'}_{\mathcal{M}} \psi_j$ for some $0 < j < i$. This implies, by Corollary 2, that there exists a policy $\pi'_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))_{A,\perp}$ and $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{j\,A,\perp}$, that is, $\pi'_{A,\perp}$ is a solution for $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$ such that $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \psi_{j\,A,\perp}$. This contradicts the assumption that $\pi^*_{A,\perp}$ is an optimal solution for $(\mathcal{M}_{A,\perp}, \Phi_{GA,\perp}, \Phi_{PA,\perp})$, thus π^* is indeed an optimal solution for $(\mathcal{M}, \Phi_G, \Phi_P)$, as required. □

6.6 Model Checking Based P4 Planner: P4Solver$_{MO}$

Now we present the planning algorithm P4Solver$_{MO}$ for P4. As shown in Algorithm 2, P4Solver$_{MO}$ takes inspiration from the P4Solver planner shown in Algorithm 1: given the MDP model $\mathcal{M}$, the preference formula Φ_P, and the goal formula Φ_G, P4Solver$_{MO}$ parses Φ_P to the form $\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$ and then calls a procedure $PolFinder_{MO}(\mathcal{M}, \Phi_G, \psi_i)$ that tries to find a policy for $\mathcal{M}$ satisfying both ψ_i and Φ_G, for increasing values of i until a solution is found, if any.

Algorithm 2 P4Solver$_{MO}$

Input: MDP model $\mathcal{M}$, goal formula Φ_G, preference formula $\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$
Output: An optimal solution π^* or "unsatisfiable"

```
1: π* = ∅
2: for i = 1 to k do
3:     π* = PolFinder_MO(M, Φ_G, ψ_i)
4:     if π* ≠ ∅ then
5:         return π*
6: return unsatisfiable
```

The only difference between P4Solver and P4Solver$_{MO}$ is the call to the $PolFinder_{MO}$ procedure. Instead of constructing a QCPP, $PolFinder_{MO}$ performs multi-objective model checking [22], [24], [42] of $\Phi_{GA,\perp}$ and $\psi_{i\,A,\perp}$ on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

$\mathcal{M}_{A,\perp}$. In fact, the aim of the multi-objective model checking of a set of probabilistic LTL formulas $\{\varphi_1, \ldots, \varphi_n\}$ is to find a policy π such that $\bar{s} \models^{\pi} \varphi_i$ for each $i \in \{1, \ldots, n\}$; this is exactly the kind of problems $PolFinder_{MO}$ needs to solve.

Algorithm 3 $PolFinder_{MO}$

Input: MDP model $\mathcal{M}$, goal formula φ, probabilistic property formula ψ
Output: A policy π or $\emptyset$

1: $A = \{\, a \in Act \mid \mathbf{occ}(a)$ occurs in $\psi \,\}$
2: $\psi_{A,\perp} = \xi_{A,\perp}(\psi)$
3: $\varphi_{A,\perp} = \xi_{A,\perp}(\varphi)$
4: $\mathcal{M}_{A,\perp} = \zeta_{\perp}(\zeta_A(\mathcal{M}))$
5: **return** $MOSolver(\mathcal{M}_{A,\perp}, \{\varphi_{A,\perp}, \psi_{A,\perp}\})$

The description of $PolFinder_{MO}$ is given in Algorithm 3: it first extracts the actions we are interested in from the probabilistic property formula ψ coming from the preference formula Φ_P. Note that there is no need to extract also the actions occurring in the goal formula $\varphi = \Phi_G$, since each $\mathbf{occ}(a)$ occurring in φ is replaced by $\xi_{A,\perp}$ by **False**; afterwards it encodes φ, ψ and $\mathcal{M}$ as $\varphi_{A,\perp}$, $\psi_{A,\perp}$ and $\mathcal{M}_{A,\perp}$, respectively; finally it calls the multi-objective model checker $MOSolver$ that returns either a policy satisfying both $\varphi_{A,\perp}$ and $\psi_{A,\perp}$, or $\emptyset$ denoting the unsatisfiability of the two formulas.

7 Algorithm Implementations and Experimental Results

In this section we present the experimental evaluation of the proposed P4 solving techniques on five case studies: the rail robot described as running example, the dinner domain from [7] and three domains (Crossing, Navigation, and Reconnaissance) taken from the discrete track in the International Probabilistic Planner Competitions (IPPC) held in ICAPS'11 and ICAPS'14.

7.1 Implementation of P4Solver

We implemented P4Solver in Scala and delegated to Z3 [19] the evaluation of the generated QCPP. We represent each state of the MDP by the atomic propositions it is labelled with. For instance, in the rail robot example, the state depicted in Figure 1 is represented by the atomic propositions $robotAt(1)$, $carryBox(1)$, $boxAt(2,6)$, $boxAt(3,21)$, and $mode(\texttt{c})$, provided that the robot is in control mode.

An action a is symbolically encoded as a possibly empty list of variables, a precondition and the effects. The variables bind the condition and the effects to the state s enabling the action. The precondition is simply a property formula that has to be satisfied by the state s (or, more precisely, by the path $\rho = s$) in order to have a enabled in s. The effect of a is a probability distribution mapping each target state occurring with non-zero probability to the corresponding probability value. The target states are encoded by a pair (R, A) of sets of atomic propositions: those not holding anymore (R), to be removed, and those now holding (A), to be added. As a concrete example, the action $\texttt{d}(b,p)$ models the drop of box b at position p: the precondition is $mode(\texttt{a}) \wedge robotAt(p) \wedge carryBox(b) \wedge \neg\exists b'(boxAt(b',p))$; the effects are $(\{carryBox(b), mode(\texttt{a})\}, \{mode(\texttt{c}), boxAt(b,p)\})$ with probability 0.95 and $(\{mode(\texttt{a})\}, \{mode(\texttt{c})\})$ with probability 0.05.

The preference and goal formulas are encoded according to the grammars in Definitions 2 and 3, and the probability intervals by the corresponding bounds.

Each state, action, and formula is uniquely identified by a number that is used to generate the variables for the SMT solver: for instance, the policy's choice of action 4 in state 37 is represented by the variable `sched_s37_a4` while `sched_s37_stop` stands for the choice of stopping in state 37. We encode the SMT problem by the SMT-LIB format [5], thus we can replace Z3 with any other solver supporting such a format. Moreover, it is easy to change the output format to support solvers like Redlog and Mathematica.

7.2 Implementation of P4Solver$_{MO}$

We have implemented our version of $PolFinder_{MO}$ as well as of $MOSolver$ in our probabilistic model checker EPMC[1], the successor of IscasMc [28], [29]. The MDP model is given in the PRISM language [41], where each state is represented by the actual value of the variables defining the state space. Each variable is either an integer belonging to a given (finite) interval, or a Boolean. For example, the state space of the rail robot domain can be represented by means of five variables, namely `mode`, `robotAt`, `carryBox`, `box0At`, and `box1At`. For instance, `robotAt` is defined in the interval $[0, N-1]$ while `carryBox` in the interval $[-1, B-1]$, where the value -1 is used to encode the fact that no box is carried. Transitions are written in a guarded command style format, i.e., they are of the form $[action]\ g \rightarrow p_1 : update_1 + \cdots + p_n : update_n$, where g is a Boolean expression on the state variables, p_i is a probability value so that $\sum_{i=1}^{n} p_i = 1$, and $update_i$ specifies how variables are changed in the next state. A transition is enabled at a state s when the guard g is true in s. For instance, the action of dropping the carried box 0 is written as follows: $[dropB0]$ $(\texttt{mode} = \texttt{action})\&(\texttt{robotAt} \mathrel{!}= \texttt{box1At})\&(\texttt{carryBox} = 0) \rightarrow 0.95 : (\texttt{carryBox}' = -1)\&(\texttt{box0At}' = \texttt{robotAt})\&(\texttt{mode}' = \texttt{control}) + 0.05 : (\texttt{mode}' = \texttt{control})$.

As for P4Solver, the preference and goal formulas are encoded according to the grammars in Definitions 2 and 3, and the probability intervals by the corresponding bounds. The only difference is that for checking whether there exists a policy satisfying two encoded formulas φ_1 and φ_2, the multi-objective model checker expects as input the property $\texttt{multi}(\varphi_1, \varphi_2)$ instead of $\{\varphi_1, \varphi_2\}$ as written in Algorithm 2.

The implementation of the multi-objective solver is as follows. First, each LTL property is converted to an individual nondeterministic Büchi automaton, as usually done for (probabilistic) LTL model checking. We then build a product of the MDP model under consideration with an on-the-fly construction of the deterministic Rabin automaton corresponding to each Büchi automaton, so to obtain a product MDP with accepting conditions; the on-the-fly construction allows for the generation of the part of the Rabin automaton that is actually required for the analysis.

1. EPMC is publicly available at https://github.com/ISCAS-PMC/ePMC

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

TABLE 1
Computation time for the Rail Robot domain

N	B	t_{P4Solver} (s) $\varphi_1/\varphi_2/\varphi_3/\varphi_4$	$t_{\text{P4Solver}_{MO}}$ (s) $\varphi_1/\varphi_2/\varphi_3/\varphi_4$	Result $\varphi_1/\varphi_2/\varphi_3/\varphi_4$
5	2	1/1/-z3to-/1	1/1/1/1	✓/✓/✓/✓
6	2	2/53/-z3to-/1	1/1/1/1	✓/✓/✓/✓
7	2	4/3/-z3to-/4	1/1/1/1	✓/✓/✓/✓
10	2	-z3to-/-z3to/-z3mo-/-z3mo-	1/1/1/1	✓/✓/✓/✓
20	2	-z3mo-	3/2/3/3	✓/✓/✓/✓
30	2	-gto-	5/5/5/10	✓/✓/✓/✓
40	2	-gto-	12/12/12/29	✓/✓/✓/✓
50	2	-gto-	31/31/31/90	✓/✓/✓/✓

Once the product MDP is built, we consider a weighting (probability distribution) of the properties. Using a value iteration algorithm, we find the optimal policy for the weighted probability to fulfill the properties and we compute the individual probabilities to fulfill each property using the policy just computed. We solve a linear program based on these values and on the probability bounds of the properties so as to obtain an improved weighting. We repeat these two steps until no further improvements are possible.

7.3 Experimental Results

To evaluate the performance of the two proposed planners P4Solver and P4Solver_{MO}, we have considered five different case studies: the rail robot introduced in Section 2 and the dinner domain [7], as well as three domains taken from the discrete track IPPC competitions in ICAPS'11[2] and ICAPS'14[3]. We have varied the size of the instances to check the scalability of the two solving algorithms.

We have run P4Solver and P4Solver_{MO} on a single core of a 3.6GHz Intel® Core™ i7-4790 with 16GB of RAM of which 12GB assigned to the tool. The results are shown in Tables from 1 to 5. The columns "t_{P4Solver} (s)" and "$t_{\text{P4Solver}_{MO}}$ (s)" of Table 1 show the overall times in seconds spent by P4Solver and P4Solver_{MO} respectively for corresponding preference formulas, while column "Result" shows whether the problem is satisfiable (✓) or not (✗). To run the experiments, we imposed different timeouts: 1800 seconds, i.e., 30 minutes, for the generation of the programming problem (P4Solver only); 86400 seconds, i.e., 24 hours, for the computation of the policy (by P4Solver_{MO} or by Z3 for P4Solver). If a tool failed the computation, we indicate the cause (timeout or memory out) in the tables by marking the corresponding entry with -to- or -mo-, respectively. For P4Solver we have also indicated in what stage it fails: when generating the programming problem (denoted by -g-) or computing the policy (denoted by -z3-).

7.3.1 The Rail Robot Domain

For the case of the rail robot, we considered a scenario with $B = 2$ boxes and $N \in \{5, 6, 7, 10, 20, 30, 40, 50\}$ positions. We used $\Phi_G = \mathcal{P}_{[1,1]}(\mathbf{final}(\forall i(boxAt(i, i))))$ as the goal formula, requiring that with probability 1 each box i is in the corresponding area i on termination. As preference formulas, we considered $\Phi_i = \varphi_i \ll \mathcal{P}_{[1,1]}(\mathbf{True})$ for the following four formulas φ_i:

$$\varphi_1 = \mathcal{P}_{[1,1]}(\mathbf{F}(\exists B(\exists P(\mathbf{occ}(\mathtt{p}(B, P)))))),$$

2. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/
3. https://ssanner.github.io/IPPC_2014/

TABLE 2
Computation time for the Dinner domain

Type	Version	t_{P4Solver} (s) $\varphi_1/\varphi_2/\varphi_3/\varphi_4$	$t_{\text{P4Solver}_{MO}}$ (s) $\varphi_1/\varphi_2/\varphi_3/\varphi_4$	Result $\varphi_1/\varphi_2/\varphi_3/\varphi_4$
np	reduced	1/1/-gmo-/1	1/1/1/1	✓/✓/✓/✓
pr	reduced	1/1/-gmo-/1	1/1/1/1	✓/✓/✓/✗
np	complete	-gto-	64/224/227/205	✓/✓/✓/✓
pr	complete	-gto-	72/830/202/188	✓/✓/✓/✓

$$\varphi_2 = \mathcal{P}_{[1,1]}(\mathbf{F}(\exists B(\exists P(\mathbf{occ}(\mathtt{d}(B, P)))))),$$
$$\varphi_3 = \mathcal{P}_{[1,1]}(\mathbf{F}(\exists B(\exists P(\mathbf{occ}(\mathtt{p}(B, P)))))), \text{ and}$$
$$\varphi_4 = \mathcal{P}_{[1,1]}(\mathbf{F}(\exists P(\mathbf{occ}(\mathtt{d}(1, P))))).$$

Note that Φ_1 and Φ_3 are actually the same formula, but we evaluate them from different initial states. The initial state relative to preference Φ_4 has $boxAt(1, 1)$, so the preference requires to drop the box 1 even if this action is not needed to satisfy the goal. The comparisons of the two algorithms are shown in Table 1.

As we can see, the experiments show how P4Solver_{MO} outperforms P4Solver as soon as the scale of the model starts to grow. For values of N at most 10, P4Solver_{MO} takes almost no time, while P4Solver already requires a long time (for φ_2 when $N = 6$) or even goes timeout (in all cases for property φ_3). In particular, P4Solver constantly fails to return a useful outcome for $N = 8$, and fails by memory out for $N = 9$. For P4Solver, according to the reduction procedure from P4 to QCPP, the number of variables in the reduced QCPP problem is quite large even for small cases. It increases exponentially with the number of subformulas of the property formula, and linearly with the number of states in the given MDP. P4Solver_{MO}, on the other hand, constructs new MDP and property formulas separately, so that the scale of encoded MDP remains conservative. Besides the efficiency of solving QCPP problems with Z3 in P4Solver is not comparable to the multi-objective model checking algorithm in P4Solver_{MO}. Thus P4Solver_{MO} naturally drastically simplified the computation of solution to P4 and outperforms P4Solver.

7.3.2 The Dinner Domain

For the case of the dinner domain, shown in Table 2, we considered two different versions: the full version as presented in [7] and the simplified version used in [43] where we have omitted some of dishes and places so to reduce the size of the problem. Moreover, for each of the version, we considered the original non-probabilistic ("np") version (i.e., each action leads to a unique successor) as well as a probabilistic ("pr") version in which each action leads to the same successor as in the non-probabilistic version with probability 0.95, while with the remaining probability 0.05 it causes no change in the state.

We used $\Phi_G = \mathcal{P}_{[0.75,1]}(\mathbf{final}(at(home) \wedge sated \wedge kitchenClean))$ as the goal formula, requiring to be at home, sated, and with the kitchen clean with probability at least 75%. We considered the following four formulas

$$\varphi_1 = \mathcal{P}_{[1,1]}(\vartheta),$$
$$\varphi_2 = \mathcal{P}_{[1,1]}(\vartheta \wedge \mathbf{F}(\iota)),$$
$$\varphi_3 = \mathcal{P}_{[1,1]}(\mathbf{F}(\vartheta \wedge \iota)), \text{ and}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

TABLE 3
Computation time for the Navigation domain

ROWS/COLS	$t_{\text{P4Solver}_{MO}}$ (s) $\varphi_1/\varphi_2/\varphi_3$	Result $\varphi_1/\varphi_2/\varphi_3$
50/500	32/15/29	✗/✓/✗
50/1000	101/48/93	✗/✓/✗
50/1500	304/120/129	✗/✓/✗
50/2000	354/170/341	✗/✓/✗
50/2500	382/221/414	✗/✓/✗
50/3000	1 154/457/461	✗/✓/✗
50/3500	1 256/545/1 131	✗/✓/✗
50/4000	1 356/630/1 286	✗/✓/✗
50/4500	1 422/716/1 428	✗/✓/✗
50/5000	1 442/820/1 545	✗/✓/✗

TABLE 4
Computation time for the Reconnaissance domain

GRID_SIZE	$t_{\text{P4Solver}_{MO}}$ (s) $\varphi_1/\varphi_2/\varphi_3$	Result $\varphi_1/\varphi_2/\varphi_3$
10	5/4/4	✓/✗/✗
20	15/14/13	✓/✗/✗
30	38/36/34	✓/✗/✗
40	101/91/86	✓/✗/✗
50	313/261/101	✓/✗/✗
60	306/323/294	✓/✗/✗
70	1 091/931/287	✓/✗/✗
80	1 159/1 083/957	✓/✗/✗
90	3 725/1 091/1 026	✓/✗/✗
100	4 236/3 450/914	✓/✗/✗

$$\varphi_4 = \mathcal{P}_{[1,1]}(\mathbf{G}(\forall L(\mathbf{occ}(drive(home, L)) \Rightarrow \imath)))$$

as the preference formulas, where

$$\imath = \mathbf{X}(at(italianRestaurant)),$$
$$\vartheta = \exists L(\mathbf{F}(\mathbf{occ}(eat)(pizza, L)),$$
$$\imath = \mathbf{occ}(drive(home, italianRestaurant)).$$

As we can see from Table 2, similarly to the robot domain shown in Table 1, we have again that, except for the very small cases where they are comparable, P4Solver_{MO} outperforms P4Solver. P4Solver fails in several cases already during the generation of the programming problem: in the robot case already when $N > 20$ and in the complete dinner case. The other failures are due to that Z3 taking too much time or memory.

Notice that in both rail robot and dinner domain, the performance of the current P4Solver seems degraded compared to the results in [43]. The results for P4Solver in Table 1 are remarkably higher than the ones for the corresponding experiments in [43, Table 1]. This is caused by the different formulation of Equation (1), as explained in Section 5, which induces a QCPP instance that is more challenging for the SMT solver to decide. In [43] we actually computed a history-independent policy, while here we look for a property-dependent policy, which takes into consideration not only the current state, but also the information about the progressed preference formula.

7.3.3 The IPPC Domains

For the models taken from the IPPC competitions, namely the Navigation, Reconnaissance, and Crossing traffic domains, shown in Tables 3, 4, and 5,[4] respectively, the goal formulas and the preference properties φ_i are as follows: the goal formula requires to be in a specific cell on the grid, and this has to happen with a probability in the interval $[0.75, 1]$. The preference formulas ask to reach or avoid specific places or to avoid specific (sequences of) actions, as well as to avoid specific actions in specific places; this has to happen with probability 1 (i.e., in $[1, 1]$).

The tables show that the performance of P4Solver_{MO} is naturally influenced by the size of the analyzed MDP: the larger the MDP is, the longer it takes to complete the task. As we can see from the experiments, P4Solver_{MO} is able to cope with MDPs whose size ranges over few million states

4. Domains available as RDDL model files in https://github.com/ssanner/rddlsim

TABLE 5
Computation time for the Crossing traffic domain

ROWS/COLS	$t_{\text{P4Solver}_{MO}}$ (s) $\varphi_1/\varphi_2/\varphi_3$	Result $\varphi_1/\varphi_2/\varphi_3$
2/2	1/1/1	✓/✗/✓
2/3	1/2/2	✓/✗/✓
2/4	3/3/4	✓/✗/✓
2/5	11/11/12	✓/✗/✓
3/2	2/2/2	✓/✗/✓
3/3	5/5/7	✓/✗/✓
3/4	90/84/92	✓/✗/✓
3/5	4 372/3 939/-mo-	✓/✗/-mo-
4/2	6/4/5	✓/✗/✗
4/3	111/103/132	✓/✗/✓
4/4	-mo-	-mo-
4/5	-mo-	-mo-

and transitions; the size of the MDP is mainly limited by the explicit state-space representation that is used to solve the multi-objective problem. This becomes clear from the Crossing traffic domain (Table 5), where P4Solver_{MO} goes out of memory on the larger instances, which involve tens to hundreds of millions of states and transitions (cf. Table 6, bottom part).

Finally, in Table 6 we report the size of the MDPs used for our experimental evaluation. Column "$|S|/|\text{P}|$" refers to the original MDP $\mathcal{M} = (S, \Sigma, L, Act, \bar{s}, \text{P})$ where $|\text{P}| = \sum_{s \in S} |Act(s)|$; similarly, "$|S_{A,\perp}|/|\text{P}_{A,\perp}|$" refers to the largest MDP obtained by using the encoding functions ζ_A and $\zeta_\perp$: $\zeta_\perp$ introduces $|S|$ new states and $2|S|$ new transitions while the contribution of ζ_A depends on the size of A, the number of transitions with action in A, and the number of states such transitions lead to.

8 Discussion

In this section we first show some difficulties to reduce P4 to optimization problem and to multi-objective planning problem with reward structure. Then we discuss how to improve the performance of P4Solver and P4Solver_{MO}, and how to extend their scope.

8.1 Discussion about Other Reductions of P4

Intuitively, we could consider other approaches to solve P4, for instance the classical probabilistic model checking or the ones adopted by other planners working with deterministic models: planners like MIPS-BDD [20] and HPLAN-P [4] first translate the LTL formula to an automaton and then find the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

TABLE 6
Model dimensions

Model (parameters)	Original model $\lvert S\rvert/\lvert \mathrm{P}\rvert$	Encoded model $\lvert S_{A,\perp}\rvert/\lvert \mathrm{P}_{A,\perp}\rvert$
Robot (5)	380/610	920/1 690
Robot (6)	624/996	1 488/2 724
Robot (7)	952/1 512	2 072/3 752
Robot (10)	2 560/4 020	5 480/9 860
Robot (20)	18 320/28 240	38 160/67 920
Robot (30)	59 240/90 660	122 040/216 180
Robot (40)	137 440/209 280	281 120/496 640
Robot (50)	264 800/402 100	539 400/951 300
Dinner-np (reduced)	64/296	196/665
Dinner-pr (reduced)	64/296	320/1 136
Dinner-np (complete)	98 304/1 089 536	357 632/2 360 640
Dinner-pr (complete)	98 304/1 089 536	524 288/3 686 400
Navigation (50/500)	49 000/290 806	195 900/777 220
Navigation (50/1000)	98 000/581 806	391 900/1 555 220
Navigation (50/1500)	147 000/872 806	587 900/2 333 220
Navigation (50/2000)	196 000/1 163 806	783 900/3 111 220
Navigation (50/2500)	245 000/1 454 806	979 900/3 889 220
Navigation (50/3000)	294 000/1 745 806	1 175 900/4 667 220
Navigation (50/3500)	343 000/2 036 806	1 371 900/5 445 220
Navigation (50/4000)	392 000/2 327 806	1 567 900/6 223 220
Navigation (50/4500)	441 000/2 618 806	1 763 900/7 001 220
Navigation (50/5000)	490 000/2 909 806	1 959 900/7 779 220
Reconnaissance (10)	14 400/56 412	29 448/89 505
Reconnaissance (20)	57 600/223 452	115 848/342 945
Reconnaissance (30)	129 600/505 692	259 848/769 185
Reconnaissance (40)	230 400/903 132	461 448/1 368 225
Reconnaissance (50)	360 000/1 415 772	720 648/2 140 065
Reconnaissance (60)	518 400/2 043 612	1 037 448/3 084 705
Reconnaissance (70)	705 600/2 786 652	1 411 848/4 202 145
Reconnaissance (80)	921 600/3 644 892	1 843 848/5 492 385
Reconnaissance (90)	1 166 400/4 618 332	2 333 448/6 955 425
Reconnaissance (100)	1 440 000/5 706 972	2 880 648/8 591 265
Crossing traffic (2/2)	256 / 4 608	868 / 8 964
Crossing traffic (2/3)	1 536 / 31 744	5 140 / 59 509
Crossing traffic (2/4)	8 196 / 172 032	27 264 / 319 616
Crossing traffic (2/5)	40 960 / 786 432	135 872 / 1 478 848
Crossing traffic (3/2)	1 280 / 45 312	4 320 / 86 496
Crossing traffic (3/3)	15 328 / 563 168	51 008 / 1 052 480
Crossing traffic (3/4)	163 296 / 6 107 616	540 032 / 11 303 296
Crossing traffic (3/5)	1 631 584 / $61.7 \cdot 10^6$	5 376 000 / $113.5 \cdot 10^6$
Crossing traffic (4/2)	6 144 / 432 128	20 672 / 817 344
Crossing traffic (4/3)	146 944 / 10 731 008	487 168 / 19 853 056
Crossing traffic (4/4)	$3.1 \cdot 10^6$ / $232.5 \cdot 10^6$	$10.3 \cdot 10^6$ / $425.9 \cdot 10^6$
Crossing traffic (4/5)	$62.4 \cdot 10^6$ / $4.7 \cdot 10^9$	$204.9 \cdot 10^6$ / $8.5 \cdot 10^9$

plan in the product. The main problem of this approach is that it works for a single formula while in P4 we have to consider two formulas with side constraints: 1) the property formula φ has to be satisfied before the goal formula γ; 2) the desired probability intervals for both preference and goal formulas have to be matched under the same policy; and 3) the policy is not required to maximize/minimize the probability of the formulas, it just has to be in the required interval. While it is possible to manage constraint 1, by carefully merging γ and φ into ψ, the constraint 2 is much more difficult to achieve; also constraint 3 is challenging: in the product, the wanted policy may choose actions that do not maximize/minimize the local satisfaction probability value, so local optimization (used e.g. for solving Bellman equations) can not be safely used to derive a global policy.

Another possibility would be to encode the formulas as reward structures and then perform multi-objective planning [13], [45], [46], [48] on the resulting models with existing planners. However, obtaining this encoding turns out to be challenging. Within the scope of our knowledge, we did not find a generic encoding between qualitative planning problems and quantitative planning problems. Besides, as mentioned previously, in P4 we are not optimizing some goal, but looking for a specific value that is intended to stand for the probability of the corresponding formula.

8.2 Discussion about Improvements and Extensions

A possible improvement for $\mathrm{P4Solver}_{MO}$ would be to avoid the encoding of actions (as requested by the translation of the $\mathbf{occ}(\cdot)$ operator) by directly supporting the occurrence of actions, since this would remove a possibly large growth of the number of states and transitions. Directly supporting actions (i.e., by considering the $\mathbf{occ}(\cdot)$ operator as a basic formula of LTL) would require the appropriate extensions of the theoretical and implementation aspects $\mathrm{P4Solver}_{MO}$ is built on (e.g., by encoding $\mathbf{occ}(\cdot)$ directly in the Büchi and Rabin automata equivalent to the LTL formula).

Since P4 has an intrinsic finite trace semantics, as given by the $\mathbf{final}(\cdot)$ in the goal formula, another possible improvement is to directly use LTL over finite traces (LTL_f) instead of standard LTL over infinite traces. LTL_f synthesis and planning have several points in common, with the former that can be seen as a generalization of the latter (see, e.g., [10], [12], [16], [17], [18], [57]). Moreover, several work has been done on LTL_f synthesis with assumptions, such as [1], [2], [11], which is related to preference planning. The use of native finite trace-based methods is likely to improve the running time of $\mathrm{P4Solver}_{MO}$, since it would avoid to have to encode termination (cf. Section 6.1). However, to the best of our knowledge, there is no algorithm known in literature about probabilistic LTL_f and multi-objective LTL_f model checking. While extending automata-based algorithms to probabilistic LTL_f model checking seems a feasible task, the multi-objective counterpart is likely to be more demanding.

A possible extension for P4 would be to consider quantitative properties, i.e., formulas of the form $\mathcal{P}_{max=?}(\varphi)$ and $\mathcal{P}_{min=?}(\varphi)$, which require to maximize or minimize the probability of satisfying φ. Multi-objective model checking supports such formulas, by means of Pareto-curve computation, but it does not scale well in this scenario. Such extensions are clearly worthwhile to be investigated but they are beyond the scope of this work.

Another possible extension for P4 would be to enrich the expressiveness of preference formulas. As in [7], each probabilistic property formula can be decorated with a priority value. Thus it will be easier to dynamically update the preference formula. Moreover, in such extension, if two probabilistic property formulas have the same value, they have to be checked simultaneously. This is applicable in both P4Solver and $\mathrm{P4Solver}_{MO}$. For P4Solver, in the proof of Theorem 2, we can add new dimensions to the state space of the MDP to represent the progression of property formulas. Therefore, the construction of Equation (1) needs to be modified accordingly, so that it simulates the progression of the original MDP $\mathcal{M}$. Thus we can still contrive the conditions for a corresponding QCPP problem in a similar manner to Equation (1), which means such extension can be implemented in P4Solver. For $\mathrm{P4Solver}_{MO}$, satisfaction verification for more probabilistic property formulas makes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

no difference to current encoding procedure. We only need to encode more formulas and add the corresponding conditions to the multi-objective model checking problem.

9 Conclusion

In this paper we have proposed a framework for probabilistic preference-based planning problem on MDPs. Our language can express rich properties, and for solving P4 we have designed two planners P4Solver and P4Solver$_{MO}$. P4Solver reduces P4 to a quadratically constrained programming problem and solves it with SMT solver. To improve computational efficiency and scalability, we have also proposed another planner for P4 based on multi-objective model checking. We have shown how the proposed planner P4Solver$_{MO}$ outperforms P4Solver, with the ability to manage much larger case studies. We have also discussed the non-applicability of other well known techniques due to the specific requirements of P4.

Appendix A
The Relevant Details for the Proof of Theorem 3

Here we provide the lemmas and propositions used in proving Theorem 3. In the following, for a given $A \subseteq Act$, we assume to have the MDP $\mathcal{M}_{A,\bot} = \zeta_\bot(\zeta_A(\mathcal{M})) = (S_{A,\bot}, L_{A,\bot}, Act_{A,\bot}, \bar{s}_{A,\bot}, \mathrm{P}_{A,\bot})$; we refer directly to its components by their symbol: for instance, we use $S_{A,\bot}$ to refer to the set of states of $\mathcal{M}_{A,\bot}$. We extend the notion of encoding to paths.

Notation 1. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and $\rho \in Paths^*_{\mathcal{M}}$. We define $\rho_{A,\bot}$ to be $\rho' \bot s_\bot$ where $s = last(\rho')$ and $\rho' = \varrho_{A,\bot}(\rho)$ is defined recursively as follows:*

$$\varrho_{A,\bot}(\rho) = \begin{cases} s & \text{if } \rho = s \in S, \\ \varrho_{A,\bot}(\rho')as & \text{if } \rho = \rho' as \text{ and } a \notin A, \\ \varrho_{A,\bot}(\rho')as_a & \text{if } \rho = \rho' as \text{ and } a \in A. \end{cases}$$

We first present some preliminary result about the satisfaction of the LTL formula φ by $\mathcal{M}$ and the satisfaction of $\xi_{A,\bot}(\varphi)$ by $\mathcal{M}_{A,\bot}$. The following two lemmas establish the relationship between finite paths in $Paths^*_{\mathcal{M}}$ and in $Paths^*_{\mathcal{M}_{A,\bot}}$. It is to worthy to recall that $\natural$ is a meta-action denoting termination, while $\bot$ in $\mathcal{M}_{A,\bot}$ is a normal action.

Lemma 1. *Let $\mathcal{M}$ be an MDP and $A \subseteq Act$ be a set of actions. For each $\rho \in Paths^*_{\mathcal{M}}$, we have that $\rho_{A,\bot} \in Paths^*_{\mathcal{M}_{A,\bot}}$. Moreover, $|\rho_{A,\bot}| = |\rho| + 1$.*

Proof. We prove by induction on the length of ρ that $\varrho_{A,\bot}(\rho) \in Paths^*_{\mathcal{M}_{A,\bot}}$ and that $|\varrho_{A,\bot}(\rho)| = |\rho|$. According to this result, since $\rho_{A,\bot}$ is $\rho' \bot s_\bot$ where $s = last(\rho')$ and $\rho' = \varrho_{A,\bot}(\rho)$, it can be immediately derived that $|\rho_{A,\bot}| = |\rho| + 1$ and that $\rho' \bot s_\bot \in Paths^*_{\mathcal{M}_{A,\bot}}$ since $\mathrm{P}_{A,\bot}(s, \bot) = \delta_{s_\bot}$ and $\delta_{s_\bot}(s_\bot) = 1 > 0$ as required.

Suppose that $|\rho| = 0$. it follows that $\rho = s$ for some state $s \in S$. Thus, by definition of $\varrho_{A,\bot}(\rho)$, we have that $\varrho_{A,\bot}(\rho) = s \in Paths^*_{\mathcal{M}_{A,\bot}}$. Moreover, $|\rho| = 0 = |\varrho_{A,\bot}(\rho)|$.

Suppose that $|\rho| > 0$, which means that $\rho = \rho' as$ for some finite path $\rho' \in Paths^*_{\mathcal{M}}$, action $a \in Act$, and state $s \in S$. By definition of $\varrho_{A,\bot}(\rho)$, it follows that $\varrho_{A,\bot}(\rho) = \varrho_{A,\bot}(\rho')as'$ where $s' = s$ if $a \notin A$ or $s' = s_a$ if $a \in A$. By inductive hypothesis, $\varrho_{A,\bot}(\rho') \in Paths^*_{\mathcal{M}_{A,\bot}}$. According to the construction of $\mathcal{M}_{A,\bot}$, if $a \in A$, we have $\mathrm{P}_{A,\bot}(last(\varrho_{A,\bot}(\rho')), a) = \mu_a$ with $\mu_a(s') > 0$ since $s' = s_a$, $\mathrm{P}(last(\rho'), a) = \mu$, and $\mu(s) > 0$. Otherwise, if $a \notin A$, we have $\mathrm{P}_{A,\bot}(last(\varrho_{A,\bot}(\rho')), a) = \mu$ with $\mu(s') > 0$ since $s' = s$, $\mathrm{P}(last(\rho'), a) = \mu$, and $\mu(s) > 0$. Thus $\varrho_{A,\bot}(\rho) = \varrho_{A,\bot}(\rho')as'$ is a legal path in $\mathcal{M}_{A,\bot}$, therefore $\varrho_{A,\bot}(\rho) \in Paths^*_{\mathcal{M}_{A,\bot}}$. Moreover, $|\rho| = |\rho'| + 1 = |\varrho_{A,\bot}(\rho')| + 1 = |\varrho_{A,\bot}(\rho)|$ since $|\rho'| = |\varrho_{A,\bot}(\rho')|$ by inductive hypothesis. □

Lemma 2. *Let $\mathcal{M}$ be an MDP and $A \subseteq Act$ be a set of actions. For each $\rho' \in Paths^*_{\mathcal{M}_{A,\bot}}$ with $first(\rho') \in S$, there exists $\rho \in Paths^*_{\mathcal{M}}$ such that $\rho' = \varrho_{A,\bot}(\rho)(\bot s_\bot)^k$ where $k \in \mathbb{N}$ and $s = last(\varrho_{A,\bot}(\rho))$.*

Proof. The proof is by induction of the length of ρ': for the base case, suppose that $|\rho'| = 0$; this means that $\rho' = t \in S$, thus by taking $\rho = t$ and $k = 0$, we have that $\varrho_{A,\bot}(\rho)(\bot t_\bot)^k = t(\bot t_\bot)^0 = t = \rho'$ as required.

For the induction step, suppose that $|\rho'| > 0$ which implies that $\rho' = \rho''' as'$ for some $a' \in Act_{A,\bot}$ and $s' \in S_{A,\bot}$. Thus there are two cases: either $a \neq \bot$ or $a = \bot$.

Consider the case $a \neq \bot$, i.e., $a \in Act_{A,\bot} \setminus \{\bot\} = Act$: since $\rho' \in Paths^*_{\mathcal{M}_{A,\bot}}$, it follows that $\mathrm{P}_{A,\bot}(last(\rho''), a) = \mu'$ with $\mu'(s') > 0$. Here are two possibilities, $a \in A$ or $a \notin A$ If $a \neq \bot$ and $a \in A$, then $s' = t_a$ for some $t \in S$ and we have $\mathrm{P}(u, a) = \mu$ such that $last(\rho'') \in \{ u, u_b \mid b \in A \}$ and $\mu' = \mu_a$. By induction hypothesis, there exists $\rho'' \in Paths^*_{\mathcal{M}}$ such that $\rho''' = \varrho_{A,\bot}(\rho'')(\bot s''_\bot)^{k''}$ where $k'' \in \mathbb{N}$ and $s'' = last(\varrho_{A,\bot}(\rho''))$. Note that it must be that $k'' = 0$ since by definition of $\mathrm{P}_{A,\bot}$, no action except $\bot$ can follow $\bot$ in a path. Now, consider $\rho = \rho'' at$ and $k = 0$: we have that $\varrho_{A,\bot}(\rho)(\bot s'_\bot)^k = \varrho_{A,\bot}(\rho'' at)(\bot s'_\bot)^0 = \varrho_{A,\bot}(\rho'' at) = \varrho_{A,\bot}(\rho'')at_a = \rho'''at_a = \rho'''as' = \rho'$, as required. If $a \neq \bot$ and $a \notin A$, we have that $s' \in S$ and $\mathrm{P}(u, a) = \mu$ such that $last(\rho'') \in \{ u, u_b \mid b \in A \}$ and $\mu' = \mu$. As in the previous case, by induction hypothesis, there exists $\rho'' \in Paths^*_{\mathcal{M}}$ such that $\rho''' = \varrho_{A,\bot}(\rho'')(\bot s''_\bot)^{k''}$ where $k'' \in \mathbb{N}$ and $s'' = last(\varrho_{A,\bot}(\rho''))$. Note that again it must be that $k'' = 0$ since by definition of $\mathrm{P}_{A,\bot}$, no action except $\bot$ can follow $\bot$ in a path. Now, consider $\rho = \rho'' as'$ and $k = 0$: we have that $\varrho_{A,\bot}(\rho)(\bot s'_\bot)^k = \varrho_{A,\bot}(\rho'' as')(\bot s'_\bot)^0 = \varrho_{A,\bot}(\rho'' as') = \varrho_{A,\bot}(\rho'')as' = \rho'''as' = \rho'$, as required.

Suppose that $a = \bot$ which means that $\rho' = \rho''' \bot s'_\bot$ where either $last(\rho''') = s'_\bot$, or $last(\rho''') = s'$ for some $s' \in \{ s, s_b \mid s \in S, b \in A \}$.

In the former case, by induction hypothesis, there exists ρ'' such that $\rho''' = \varrho_{A,\bot}(\rho'')(\bot s''_\bot)^{k''}$ where $k'' \in \mathbb{N}$ and $s'' = last(\varrho_{A,\bot}(\rho''))$. Since $last(\rho''') = s'_\bot$, $Act(s'_\bot) = \{\bot\}$, and $\mathrm{P}_{A,\bot}(s'_\bot, \bot) = \delta_{s'_\bot}$, we have that $k'' > 0$ and $s''_\bot = s'_\bot$. Now, consider $\rho = \rho''$ and $k = k'' + 1$: we have that $\varrho_{A,\bot}(\rho)(\bot s'_\bot)^k = \varrho_{A,\bot}(\rho'')(\bot s'_\bot)^{k''+1} = \varrho_{A,\bot}(\rho'')(\bot s'_\bot)^{k''} \bot s'_\bot = \rho''' \bot s'_\bot = \rho'$, as required.

Consider now the latter case where $last(\rho''') = s'$ for some $s' \in \{ s, s_b \mid s \in S, b \in A \}$. By induction hypothesis, there exists ρ'' such that $\rho''' = \varrho_{A,\bot}(\rho'')(\bot s''_\bot)^{k''}$ where $k'' \in \mathbb{N}$ and $s'' = last(\varrho_{A,\bot}(\rho''))$. Note that it must be that $k'' = 0$ since by definition of $\mathrm{P}_{A,\bot}$, states of the set $\{ s, s_b \mid s \in S, b \in A \}$ can never be reached by a $\bot$ action and no action

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

except $\bot$ can follow $\bot$ in a path. Now, consider $\rho = \rho''$ and $k = 1$: we have that $\varrho_{A,\bot}(\rho)(\bot s'_\bot)^k = \varrho_{A,\bot}(\rho'')(\bot s'_\bot)^1 = \varrho_{A,\bot}(\rho'')\bot s'_\bot = \rho'''\bot s'_\bot = \rho'$, as required.

This completes the proof that for each $\rho' \in Paths^*_{\mathcal{M}_{A,\bot}}$ with $first(\rho') \in S$, there exists $\rho \in Paths^*_{\mathcal{M}}$ such that $\rho' = \varrho_{A,\bot}(\rho)(\bot s_\bot)^k$ where $k \in \mathbb{N}$ and $s = last(\varrho_{A,\bot}(\rho))$. □

For a property formula φ, the encoded LTL formula $\xi_{A,\bot}(\varphi, \mathbf{True})$ is not equivalent to φ, but the paths of $\mathcal{M}_{A,\bot}$ that are the counterpart of paths of $\mathcal{M}$ satisfying φ also satisfy $\xi_{A,\bot}(\varphi, \mathbf{True})$.

Lemma 3. *Let $\mathcal{M}$ be an MDP and $A \subseteq Act$ be a set of actions. For each $\rho \in Paths^*_{\mathcal{M}}$ and each property formula φ, we have that $\rho \models \varphi$ if and only if $\rho_{A,\bot} \models \xi_{A,\bot}(\varphi, \mathbf{True})$.*

Proof. The proof is by induction of the semantics of $\rho \models \varphi$:

1) For the case $\varphi = p \in \Sigma$, $\xi_{A,\bot}(p, \mathbf{True}) = p$. Thus
$$\begin{aligned} & \rho \models p \\ \iff\ & p \in L(first(\rho)) \\ \iff\ & p \in L_{A,\bot}(first(\rho)) \\ \iff\ & \rho_{A,\bot} \models p \\ \iff\ & \rho_{A,\bot} \models \xi_{A,\bot}(p, \mathbf{True}) \end{aligned}$$
2) For the case $\varphi = \neg\psi$, $\rho \models \neg\psi \iff \rho \not\models \psi$. By inductive hypothesis, $\rho \not\models \psi \iff \rho_{A,\bot} \not\models \xi_{A,\bot}(\psi, \mathbf{True})$. Since $\xi_{A,\bot}(\neg\psi, \mathbf{True}) = \neg\xi_{A,\bot}(\psi, \mathbf{True})$, we have
$$\begin{aligned} & \rho \models \neg\psi \\ \iff\ & \rho_{A,\bot} \not\models \xi_{A,\bot}(\psi, \mathbf{True}) \\ \iff\ & \rho_{A,\bot} \models \xi_{A,\bot}(\neg\psi, \mathbf{True}) \end{aligned}$$
3) For the case $\varphi = \psi_1 \wedge \psi_1$, $\rho \models \psi_1 \wedge \psi_1 \iff \rho \models \psi_1 \wedge \rho \models \psi_2$ by definition. From the inductive hypothesis, $\rho \models \psi_i \iff \rho_{A,\bot} \models \xi_{A,\bot}(\psi_i, \mathbf{True})$ for $i = 1, 2$. Thus
$$\begin{aligned} & \rho \models \psi_1 \wedge \psi_1 \\ \iff\ & \rho_{A,\bot} \models \xi_{A,\bot}(\psi_1, \mathbf{True}) \wedge \xi_{A,\bot}(\psi_2, \mathbf{True}) \\ \iff\ & \rho_{A,\bot} \models \xi_{A,\bot}(\psi_1 \wedge \psi_2, \mathbf{True}). \end{aligned}$$
4) For the case $\varphi = \mathbf{X}(\psi)$, if $|\rho| = 0$, $\rho \models \mathbf{X}(\psi) \iff \rho \models \mathbf{False}$, and if $|\rho| > 0$, we have
$$\begin{aligned} & \rho \models \mathbf{X}(\psi) \\ \iff\ & suffix(\rho, 1) \models \psi \\ \iff\ & suffix(\rho_{A,\bot}, 1) \models \xi_{A,\bot}(\psi, \mathbf{True}) \\ \iff\ & suffix(\rho_{A,\bot}, 1) \models \mathbf{True} \wedge \xi_{A,\bot}(\psi, \mathbf{True}) \\ \iff\ & \rho_{A,\bot} \models \xi_{A,\bot}(\mathbf{X}(\psi), \mathbf{True}). \end{aligned}$$
5) For the case $\varphi = \mathbf{U}(\psi_1, \psi_2)$, since $\rho \models \mathbf{U}(\psi_1, \psi_2) \iff \rho \models \psi_2 \vee \rho \models \psi_1 \wedge \mathbf{X}(\mathbf{U}(\psi_1, \psi_2))$, proof of this case is a trivial composition of the proofs of the cases $\varphi = \neg\psi$, $\varphi = \mathbf{X}(\psi)$ and $\varphi = \psi_1 \wedge \psi_1$.
6) For the case $\varphi = \mathbf{occ}(a)$, according to the definition, $\rho \models \mathbf{occ}(a) \iff action(\rho, 1) = a$, i.e., the first action of ρ is a. Since $b \in A$ occurs if and only if action b appears in the scope of a formula of type $\mathbf{occ}(\cdot)$, we have that $\rho_{A,\bot}$ satisfies that $action(\rho_{A,\bot}, 1) = a$ and its second state (i.e., $first(suffix(\rho_{A,\bot}, 1))$) is of the form s_a with $a \in L_{A,\bot}(s_a)$ iff $a \in A$. Together with the fact that $\xi_{A,\bot}(\mathbf{occ}(a), \mathbf{True}) = \mathbf{True} \wedge \mathbf{X}(a)$, we have
$$\begin{aligned} & \rho \models \mathbf{occ}(a) \\ \iff\ & a \in L_{A,\bot}(first(suffix(\rho_{A,\bot}, 1))) \\ \iff\ & suffix(\rho_{A,\bot}, 1) \models a \\ \iff\ & \rho_{A,\bot} \models \mathbf{X}(a) \\ \iff\ & \rho_{A,\bot} \models \mathbf{True} \wedge \mathbf{X}(a) \\ \iff\ & \rho_{A,\bot} \models \xi_{A,\bot}(\mathbf{occ}(a), \mathbf{True}). \end{aligned}$$
7) For the case $\varphi = \forall x(\psi)$, the proof is a natural extension of the case $\varphi = \psi_1 \wedge \psi_2$.
8) For the case $\varphi = \mathbf{final}(\psi)$, we have $\rho \models \mathbf{final}(\psi) \iff last(\rho) \models \psi$. Now we consider $\rho_{A,\bot}$. By definition, a state s in $\rho_{A,\bot}$ satisfies $\bot \in L_{A,\bot}(s)$ iff $s = last(\rho_{A,\bot})$. Let t be the second last state in $\rho_{A,\bot}$. By definition, we have that $L_{A,\bot}(t) = L_{A,\bot}(last(\rho_{A,\bot}))\setminus\{\bot\}$, indicating that all the properties except $\bot$ satisfied by $last(\rho_{A,\bot})$ is also already satisfied by t. This means that for the segment $t\bot last(\rho_{A,\bot})$, $last(\rho) \models \psi \iff t\bot last(\rho_{A,\bot}) \models \mathbf{F}(\bot \wedge \xi_{A,\bot}(\psi, \mathbf{False}))$. Since $\xi_{A,\bot}(\mathbf{final}(\psi), \mathbf{True}) = \mathbf{F}(\bot \wedge \xi_{A,\bot}(\psi, \mathbf{False}))$ we have
$$\begin{aligned} & \rho_{A,\bot} \models \xi_{A,\bot}(\mathbf{final}(\psi)) \\ \iff\ & \rho_{A,\bot} \models \mathbf{F}(\bot \wedge \xi_{A,\bot}(\psi, \mathbf{False})) \\ \iff\ & t\bot last(\rho_{A,\bot}) \models \mathbf{F}(\bot \wedge \xi_{A,\bot}(\psi, \mathbf{False})) \\ \iff\ & last(\rho) \models \psi \\ \iff\ & \rho \models \mathbf{final}(\psi). \end{aligned}$$

This completes the proof that $\rho \models \varphi$ if and only if $\rho_{A,\bot} \models \xi_{A,\bot}(\varphi, \mathbf{True})$. □

Corollary 1. *Let $\mathcal{M}$ be an MDP and $A \subseteq Act$ be a set of actions. For each $\rho' \in Paths^*_{\mathcal{M}_{A,\bot}}$ with $first(\rho') \in S$ and each property formula φ, there exists $\rho \in Paths^*_{\mathcal{M}}$ such that $\rho \models \varphi$ if and only if $\rho' \models \xi_{A,\bot}(\varphi, \mathbf{True})$, and $\rho' = \varrho_{A,\bot}(\rho)(\bot s_\bot)^k$ where $k \in \mathbb{N}$ and $s_\bot = last(\rho_{A,\bot})$.*

Proof. Based on the discussion for the case $\varphi = \mathbf{final}(\psi)$ in Lemma 3, Corollary 1 is a trivial extension of Lemma 3. □

From the above two lemmas, we prove the equivalence of the verifications on $\mathcal{M}$ and $\mathcal{M}_{A,\bot}$. Now assuming that φ is a probabilistic property formula, we prove that if we have a policy π in $\mathcal{M}$ such that $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$, we can contrive a policy $\pi_{A,\bot}$ in $\mathcal{M}_{A,\bot}$ such that $\bar{s}_{A,\bot} \models^{\pi_{A,\bot}}_{\mathcal{M}_{A,\bot}} \varphi_{A,\bot}$, and vice-versa. Note that this result can be extended to the cases with two or more property formulas, indicating the correctness of $\mathrm{P4Solver}_{MO}$. We prove the first direction as follows.

Lemma 4. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and π be a policy for $\mathcal{M}$. There exists a policy $\pi_{A,\bot}$ such that for each $\rho \in Paths^*_{\mathcal{M}}$ and $s \in S$, $\mu_{\pi,s}(\rho) = \mu_{\pi_{A,\bot},s}(\rho_{A,\bot})$.*

Proof. For each $\rho' \in Paths^*_{\mathcal{M}_{A,\bot}}$ and each $a \in Act_{A,\bot}$, we define the policy $\pi_{A,\bot}(\rho')(a)$ as follows:
$$\pi_{A,\bot}(\rho')(a) = \begin{cases} \pi(\rho)(a) & \text{if } \rho' = \xi_{A,\bot}(\rho) \text{ and } a \in Act; \\ \pi(\rho)(\bot) & \text{if } \rho' = \xi_{A,\bot}(\rho) \text{ and } a = \bot; \\ 0 & \text{otherwise.} \end{cases}$$
By Lemma 1 we have that $\rho_{A,\bot} \in Paths^*_{\mathcal{M}_{A,\bot}}$, thus also $\xi_{A,\bot}(\rho) \in Paths^*_{\mathcal{M}_{A,\bot}}$, hence $\pi_{A,\bot}(\rho')$ is well defined.

By induction on the length of $\rho = s_0 a_1 s_1 \dots a_n s_n$, we show that $\mu_{\pi,s}(C_\rho) = \mu_{\pi_{A,\bot},s}(C_{\xi_{A,\bot}(\rho)})$, where C_ρ is the set of all paths having ρ as prefix and $\mu_{\pi,s}(C_\rho) = \delta_s(s_0) \cdot \prod_{i=1}^{n}(\pi(\rho_{i-1})(a_i) \cdot \mathrm{P}(s_{i-1}, a_i)(s_i))$, where $\rho_j =$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

$s_0 a_1 s_1 \dots a_j s_j$. Note that $\mu_{\pi,s}(\rho) = \mu_{\pi,s}(C_\rho) \cdot \pi(\rho)(\natural)$. Suppose that $|\rho| = 0$; it follows that $\rho = t$ for some state $t \in S$, thus $\mu_{\pi,s}(C_\rho) = 1 = \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)})$ if $t = s$, and $\mu_{\pi,s}(C_\rho) = 0 = \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)})$ if $t \neq s$.

Suppose that $|\rho| > 0$. Thus $\rho = \rho' a s$ for some finite path $\rho' \in Paths^*_{\mathcal{M}}$, action $a \in Act$, and state $s \in S$. By definition of $\xi_{A,\perp}(\rho)$, it follows that $\xi_{A,\perp}(\rho) = \xi_{A,\perp}(\rho') a t'$ where $t' = t$ if $a \notin A$ or $t' = t_a$ if $a \in A$. From this, we derive the following sequence of equalities (comments refer to the previous equality):

$$\begin{aligned}
&\mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho')}) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho'))(a) \\
&\qquad \cdot \mathrm{P}_{A,\perp}(last(\xi_{A,\perp}(\rho')), a)(t') \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho')}) \cdot \pi(\rho')(a) \cdot \mathrm{P}(last(\rho'), a)(t) \\
&= \mu_{\pi,s}(C_{\rho'}) \cdot \pi(\rho')(a) \cdot \mathrm{P}(last(\rho'), a)(t) \\
&= \mu_{\pi,s}(C_\rho).
\end{aligned}$$

Since $\mathrm{P}_{A,\perp}(last(\xi_{A,\perp}(\rho)), \perp) = \delta_{last(\xi_{A,\perp}(\rho))_\perp}$, we have

$$\begin{aligned}
&\mu_{\pi_{A,\perp},s}(\rho_{A,\perp}) \\
&= \mu_{\pi_{A,\perp},s}(\xi_{A,\perp}(\rho) \perp last(\xi_{A,\perp}(\rho))_\perp) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho))(\perp) \\
&\qquad \cdot \mathrm{P}_{A,\perp}(last(\xi_{A,\perp}(\rho)), \perp)(last(\xi_{A,\perp}(\rho))_\perp) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho))(\perp) \cdot 1 \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \cdot \pi(\rho)(\perp) \\
&= \mu_{\pi,s}(C_\rho) \cdot \pi(\rho)(\perp) \\
&= \mu_{\pi,s}(\rho).
\end{aligned}$$

□

Proposition 1. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and $\varphi = \mathcal{P}_J(\psi)$ be a PLTL formula. If there exists a policy π for $\mathcal{M}$ such that $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$, then there exists a policy $\pi_{A,\perp}$ for $\mathcal{M}_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$.*

Proof. The result follows easily from the previous lemmas: let $\pi_{A,\perp}$ be the policy constructed by Lemma 4 for π. Then

$$\begin{aligned}
&\bar{s} \models^{\pi}_{\mathcal{M}} \varphi \\
\iff &\mu_{\pi,\bar{s}}(\{\, \rho \in Paths^*_{\mathcal{M}} \mid \rho \models \psi \,\}) \in J \\
\iff &\sum_{\rho \in \{\, \rho' \in Paths^*_{\mathcal{M}} \mid \rho' \models \psi \,\}} \mu_{\pi,\bar{s}}(\rho) \in J \\
\overset{\dagger}{\iff} &\sum_{\rho_{A,\perp} \in \Delta} \mu_{\pi_{A,\perp},\bar{s}_{A,\perp}}(\rho_{A,\perp}) \in J \\
\iff &\mu_{\pi_{A,\perp},\bar{s}_{A,\perp}}(\Delta) \in J \\
\iff &\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \xi_{A,\perp}\varphi
\end{aligned}$$

where $\Delta = \{\, \rho_{A,\perp} \in Paths^*_{\mathcal{M}_{A,\perp}} \mid \rho_{A,\perp} \models \xi_{A,\perp}(\psi, \mathbf{True}) \,\}$. The equivalence † is justified by Lemma 4 with respect to the probability of each path, by Lemma 3 with respect to the satisfaction of the formula by each path, and by Lemmas 1 and 2 for the correspondence between paths. □

Corollary 2. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and φ, φ' be two PLTL formulas. If there exists a policy π for $\mathcal{M}$ such that $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$ and $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi'$, then there exists a policy $\pi_{A,\perp}$ for $\mathcal{M}_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$ and $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi'_{A,\perp}$.*

Proof. The result is immediate by the way the policy $\pi_{A,\perp}$ is constructed in the proof of Lemma 4 providing the policy for Proposition 1: it does not depend on the PLTL formula φ but it depends only on π. This implies that when Proposition 1 is used to derive $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$ from $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$ and used again to derive $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi'_{A,\perp}$ from $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi'$, we actually have $\pi'_{A,\perp} = \pi_{A,\perp}$, as required. □

To complete the proof, now we turn to the second direction using the similar techniques.

Lemma 5. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and $\pi_{A,\perp}$ be a policy for $\mathcal{M}_{A,\perp}$. There exists a policy π such that for each $\rho \in Paths^*_{\mathcal{M}}$ and $s \in S$, $\mu_{\pi,s}(\rho) = \mu_{\pi_{A,\perp},s}(\{\, \rho_{A,\perp}(\perp t_\perp)^k \mid k \in \mathbb{N}, t_\perp = last(\rho_{A,\perp}) \,\})$.*

Proof. For each $\rho \in Paths^*_{\mathcal{M}}$ and each $a \in Act$, we define the policy $\pi(\rho)(a)$ as follows: if $last(\xi_{A,\perp}(\rho)) = t_b \in \{\, s_b \mid b \in A \,\}$, $a \in Act(t)$ or $last(\xi_{A,\perp}(\rho)) = t \in S$, $a \in Act(t)$, $\pi(\rho)(a) = \pi_{A,\perp}(\xi_{A,\perp}(\rho))(a)$; otherwise $\pi(\rho)(a) = 0$.

We now show by induction on the length of ρ that $\mu_{\pi,s}(C_\rho) = \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)})$. Suppose that $|\rho| = 0$: this means that $\rho = t$ for some $t \in S$; we have that $\mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) = \mu_{\pi_{A,\perp},s}(C_t) = \delta_s(t) = \mu_{\pi,s}(C_t) = \mu_{\pi,s}(C_\rho)$. Suppose that $|\rho| > 0$: in this case we have that $\rho = \rho' a t$ for some action a and state t such that $\mathrm{P}(last(\rho'), a) = \mu$ and $\mu(t) > 0$. Together with definition of π and $\mathrm{P}_{A,\perp}$ and inductive hypothesis, we then have

$$\begin{aligned}
&\mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho' a t)}) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho')}) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho'))(a) \\
&\qquad \cdot \mathrm{P}_{A,\perp}(last(\xi_{A,\perp}(\rho')), a)(last(\xi_{A,\perp}(\rho))) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho')}) \cdot \pi(\rho')(a) \cdot \mathrm{P}(last(\rho'), a)(t) \\
&= \mu_{\pi,s}(C_{\rho'}) \cdot \pi(\rho')(last(\rho'), a, \mu) \cdot \mu(t) \\
&= \mu_{\pi,s}(C_\rho).
\end{aligned}$$

By definition of $\mathrm{P}_{A,\perp}$, it is now immediate to see that $\{\, \rho_{A,\perp}(\perp t_\perp)^k \mid k \in \mathbb{N}, t_\perp = last(\rho_{A,\perp}) \,\} = C_{\rho_{A,\perp}}$. Together with $\mathrm{P}_{A,\perp}(last(\xi_{A,\perp}(\rho)), \perp)(last(\xi_{A,\perp}(\rho))_\perp) = \delta_{last(\xi_{A,\perp}(\rho))_\perp}$ and the previous result stating $\mu_{\pi,s}(C_\rho) = \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)})$, we have

$$\begin{aligned}
&\mu_{\pi_{A,\perp},s}(\{\, \rho_{A,\perp}(\perp t_\perp)^k \mid k \in \mathbb{N}, t_\perp = last(\rho_{A,\perp}) \,\}) \\
&= \mu_{\pi_{A,\perp},s}(C_{\rho_{A,\perp}}) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho) \perp last(\xi_{A,\perp}(\rho))_\perp}) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho))(\perp) \\
&\qquad \cdot \mathrm{P}_{A,\perp}(last(\xi_{A,\perp}(\rho)), \perp)(last(\xi_{A,\perp}(\rho))_\perp) \\
&= \mu_{\pi_{A,\perp},s}(C_{\xi_{A,\perp}(\rho)}) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho))(\perp) \cdot 1 \\
&= \mu_{\pi,s}(C_\rho) \cdot \pi_{A,\perp}(\xi_{A,\perp}(\rho))(\perp) \\
&= \mu_{\pi,s}(C_\rho) \cdot \pi(\rho)(\perp) \\
&= \mu_{\pi,s}(\rho).
\end{aligned}$$

□

Proposition 2. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and $\varphi = \mathcal{P}_J(\psi)$ be a PLTL formula. If there exists a policy $\pi_{A,\perp}$ for $\mathcal{M}_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$, then there exists a policy π for $\mathcal{M}$ such that $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$.*

Proof. The result follows easily from the previous lemmas: let π be the policy constructed by Lemma 5 for $\pi_{A,\perp}$. Then,

$$\begin{aligned}
&\bar{s} \models^{\pi}_{\mathcal{M}} \varphi \\
\iff &\mu_{\pi,\bar{s}}(\{\, \rho \in Paths^*_{\mathcal{M}} \mid \rho \models \psi \,\}) \in J
\end{aligned}$$

$$\iff \sum_{\rho\in\Delta_1} \mu_{\pi,\bar{s}}(\rho) \in J$$

$$\overset{\dagger}{\iff} \sum_{\rho\in\Delta_1} \sum_{\rho'_{A,\perp}\in\{\,\rho_{A,\perp}(\perp s_\perp)^k \mid k\in\mathbb{N}\,\}} \mu_{\pi_{A,\perp},\bar{s}_{A,\perp}}(\rho'_{A,\perp}) \in J$$

$$\overset{\ddagger}{\iff} \sum_{\rho_{A,\perp}\in\Delta_2} \mu_{\pi_{A,\perp},\bar{s}_{A,\perp}}(\rho_{A,\perp}) \in J$$

$$\iff \mu_{\pi_{A,\perp},\bar{s}_{A,\perp}}(\Delta_2) \in J$$

$$\iff \bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$$

where $\Delta_1 = \{\,\rho' \in Paths^*_{\mathcal{M}} \mid \rho' \models \psi\,\}$ and $\Delta_2 = \{\,\rho'_{A,\perp} \in Paths^*_{\mathcal{M}_{A,\perp}} \mid \rho'_{A,\perp} \models \xi_{A,\perp}(\psi, \mathbf{True})\,\}$. The equivalence † is justified by Lemma 5 with respect to the probability of each path; the equivalence ‡ is justified by Corollary 1 with respect to the satisfaction of the formula by each path, by Lemmas 1 and 2 for the correspondence between paths, and by the fact that for each $\rho, \rho' \in Paths^*_{\mathcal{M}}$, if $\rho \neq \rho'$, then $\{\,\rho_{A,\perp}(\perp s_\perp)^k \mid k \in \mathbb{N}\,\} \cap \{\,\rho'_{A,\perp}(\perp s_\perp)^k \mid k \in \mathbb{N}\,\} = \emptyset$. □

Corollary 3. *Let $\mathcal{M}$ be an MDP, $A \subseteq Act$ be a set of actions, and φ, φ' be two PLTL formulas. If there exists a policy $\pi_{A,\perp}$ for $\mathcal{M}_{A,\perp}$ such that $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$ and $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi'_{A,\perp}$, then there exists a policy π for $\mathcal{M}$ so that $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$ and $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi'$.*

Proof. The result is immediate by the way the policy $\pi_{A,\perp}$ is constructed in the proof of Proposition 2: it does not depend on the PLTL formula φ but it depends only on π. This implies that when Proposition 2 is used to derive $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi$ from $\bar{s}_{A,\perp} \models^{\pi_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi_{A,\perp}$ and used again to derive $\bar{s} \models^{\pi}_{\mathcal{M}} \varphi'$ from $\bar{s}_{A,\perp} \models^{\pi'_{A,\perp}}_{\mathcal{M}_{A,\perp}} \varphi'_{A,\perp}$, we actually have $\pi' = \pi$, as required. □

ACKNOWLEDGMENTS

The authors would like to show sincere gratitude to the editor and the anonymous reviewers for their numerous detailed suggestions and comments on improving the quality of this paper.

REFERENCES

[1] B. Aminof, G. De Giacomo, A. Murano, and S. Rubin. Synthesis under assumptions. In *KR*, pages 615–616, 2018.
[2] B. Aminof, G. De Giacomo, A. Murano, and S. Rubin. Planning under LTL environment specifications. In *ICAPS*, pages 31–39, 2019.
[3] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191, 2000.
[4] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173:593–618, 2009.
[5] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB standard: Version 2.0. In *SMT*, 2010.
[6] R. Bellman. A Markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.
[7] M. Bienvenu, C. Fritz, and S. A. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175:1308–1345, 2011.
[8] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
[9] G. Brewka, S. Benferhat, and D. L. Berre. Qualitative Choice Logic. *Artificial Intelligence*, 157:203–237, 2004.
[10] A. Camacho, J. A. Baier, C. J. Muise, and S. A. McIlraith. Finite LTL synthesis as planning. In *ICAPS*, pages 29–38, 2018.
[11] A. Camacho, M. Bienvenu, and S. A. McIlraith. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, pages 454–463, 2018.
[12] A. Camacho, E. Triantafillou, C. J. Muise, J. A. Baier, and S. A. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*, pages 3716–3724, 2017.
[13] K. Chatterjee, R. Majumdar, and T. A. Henzinger. Markov decision processes with multiple objectives. In *STACS*, pages 325–336, 2006.
[14] S. Coste-Marquis, J. Lang, P. Liberatore and P. Marquis. Expressive power and succinctness of propositional languages for preference representation. In *KR*, pages 203–212, 2004.
[15] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson. MPDM: multipolicy decision-making in dynamic, uncertain environments for autonomous driving. In *ICRA*, pages 1670–1677, 2015.
[16] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.
[17] G. De Giacomo and M. Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015.
[18] G. De Giacomo and S. Rubin. Automata-theoretic foundations of FOND planning for LTL_f/LDL_f goals. In *IJCAI*, pages 4729–4735, 2018.
[19] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
[20] S. Edelkamp. Optimal symbolic PDDL3 planning with MIPS-BDD. In *5th International Planning Competition Booklet (IPC-2006)*, pages 31–33, 2006.
[21] S. D. Eppinger. A planning method for integration of large-scale engineering systems. In *ICED*, volume 97, pages 1–6, 1997.
[22] K. Etessami, M. Kwiatkowska, M. Y. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4(8):1–21, 2008.
[23] Y. Feng, Z. Zhong, J. Li, W. Fan, and S. C. Feng. Closed-Loop Production Lines With Geometric Reliability Machines: Modeling, Analysis, and Application. *IEEE Robotics and Automation Letters*, 3(2):704–711, 2018.
[24] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS*, pages 112–127, 2011.
[25] M. Y. R. Gadelha, E. Steffinlongo, L. C. Cordeiro, B. Fischer, and D. A. Nicole. SMT-based refutation of spurious bug reports in the clang static analyzer. In *ICSE*, pages 11–14, 2019.
[26] R. A. Howard. Dynamic Programming and Markov Processes. Mathematical Gazette, 3(358):120, 1960.
[27] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3: The language of the fifth international planning competition. Technical report, University of Brescia, 2005.
[28] E. M. Hahn, G. Li, S. Schewe, A. Turrini, and L. Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR*, pages 354–367, 2015.
[29] E. M. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang. ISCASMC: A web-based probabilistic model checker. In *FM*, pages 312–317, 2014.
[30] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
[31] V. Hasu and H. Koivo. Automatic maintenance route planning of large-scale sensor networks. In *CISMA*, pages 18–23, 2009.
[32] J. Hoffmann, H. Hermanns, M. Klauck, M. Steinmetz, E. Karpas, and D. Magazzeni. Let's learn their language? A case for planning with automata-network languages from model checking. In *AAAI*, pages 13569–13575, 2020.
[33] R. M. Karp. Reducibility among combinatorial problems. In *CCC*, pages 85–113, 1972.
[34] T. Keller and M. Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *ICAPS*, pages 135–143, 2013.
[35] J. Kim, C. J. Banks, and J. A. Shah. Collaborative planning with encoding of users' high-level strategies. In *AAAI*, pages 955–962, 2017.
[36] M. Klauck, M. Steinmetz, J. Hoffmann, and H. Hermanns. Bridging the Gap Between Probabilistic Model Checking and Probabilistic Planning: Survey, Compilations, and Empirical Comparison. In *Journal of Artificial Intelligence Research*, 68:247–310, 2020.
[37] A. Kolobov, Mausam, and D. S. Weld. LRTDP versus UCT for online probabilistic planning. In *AAAI*, pages 1786–1792, 2012.
[38] L. Kocsis, and C. Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, pages 282–293, 2006.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3024215, IEEE Transactions on Software Engineering

[39] D. Kroening, A. Groce, and E. Clarke. Counterexample guided abstraction refinement via program execution. In *ICFEM*, pages 224–238, 2004.
[40] J. Kvarnström and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):119–169, 2000.
[41] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.
[42] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Compositional probabilistic verification through multi-objective model checking. *Information and Computation*, 232:38–65, 2013.
[43] M. Li, Z. She, A. Turrini, and L. Zhang. Preference planning for Markov decision processes. In *AAAI*, pages 3313–3319, 2015.
[44] J. Mendes-Moreira, L. Moreira-Matias, J. Gama, and J. Freire de Sousa. Validating the coverage of bus schedules: A machine learning approach. *Information Sciences*, 293:299–313, 2015.
[45] A. Mouaddib. Multi-objective decision-theoretic path planning. In *ICRA*, pages 2814–2819, 2004.
[46] W. Ogryczak, P. Perny, and P. Weng. A compromise programming approach to multiobjective Markov decision processes. *International Journal of Information Technology and Decision Making*, 12(5):1021–1054, 2013.
[47] J. Olhager and A. Feldmann. Distribution of manufacturing strategy decision-making in multi-plant networks. *International Journal of Production Research*, 56(1-2):692–708, 2018.
[48] P. Perny, P. Weng, J. Goldsmith, and J. P. Hanna. Approximation of Lorenz-optimal solutions in multiobjective Markov decision processes. In *AAAI*, pages 92–94, 2013.
[49] A. Resano Lázaro and C. J. Luis pérez. Analysis of an automobile assembly line as a network of closed loops working in both, stationary and transitory regimes. *International Journal of Production Research*, 46(17):4803–4825, 2008.
[50] S. Sanner. Relational dynamic influence diagram language (RDDL): Language description. Unpublished manuscript, Australian National University, 2010.
[51] W. Schwarting, J. Alonso-Mora, and D. Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, 2018.
[52] S. Sohrabi, J. A. Baier, and S. A. McIlraith. HTN planning with preferences. In *IJCAI*, pages 1790–1797, 2009.
[53] T. C. Son, and E. Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.
[54] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen. The structure and value of modularity in software design. In *ESEC/SIGSOFT FSE*, pages 99–108, 2001.
[55] F. Teichteil-Königsbuch. Path-constrained Markov decision processes: bridging the gap between probabilistic model-checking and decision-theoretic planning. In *ECAI*, pages 774–749, 2012.
[56] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff'94*, pages 238–266, 1995.
[57] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi. Symbolic LTL_f synthesis. In *IJCAI*, pages 1362–1369, 2017.

Meilun Li received the B.Eng. degree in school of automation science and electrical engineering from Beihang University, Beijing, China, in 2012. He is currently pursuing the Ph.D. degree in applied mathematics from Beihang University, Beijing, China. He has been a visiting student at Carl von Ossietzky University Oldenburg, Germany, in 2017-2018, under supervision of Prof. Martin Fränzle. His research interests include reachable set computation for dynamical systems and preference planning for Markov decision processes.

Andrea Turrini received his M.Sc and Ph.D. in Computer Science from the University of Verona, Italy. He was first a Postdoctoral Researcher in Verona and then at Saarland University, Saarbrücken, Germany. He then moved to the Institute of Software, Chinese Academy of Sciences, Beijing, China as Associate Research Professor in the group of Lijun Zhang. His research interests include the formal analysis of probabilistic and quantum systems, with a particular focus on system minimization and verification of omega regular properties. He is also interested in omega regular automata, including learning-based algorithms for automata operations.

Ernst Moritz Hahn has received a Ph.D. in 2013 from Saarland University. After his dissertation, he became research assistant at the group of automated verification at the University of Oxford in the VERIWARE project, and has then worked as associate professor in the Institute of Software, Chinese Academy of Sciences (ISCAS). After taking a sabbatical at Saarland University, in 2017 he started working as a Marie Sklodowska Curie Fellow at University of Liverpool, then joined Queen's University Belfast as a Lecturer in Computer Science and before moving to University of Twente. His research area is located in the area of formal methods. There, he works in probabilistic model checking, which is the automatic analysis of properties (given e.g. as omega-regular languages) of stochastic models (such as for instance Markov decision processes).

Zhikun She received the B.Sc. degree in Computational Mathematics and the Ph.D. degree in Mathematics from Peking University, Beijing, China, in 1999 and 2005, respectively. He is a Professor and Vice-Dean in the School of Mathematics and Systems Science, Beihang University, Beijing, China. From January 2004 to December 2006, he was a Postdoctoral Researcher in the MPI für Informatik. His research interests include theory, methodologies, and applications of safety verification and stability analysis of hybrid systems. He has been the Principal Investigator of more than ten research projects and published more than 60 research papers. Dr. She received the first class of Natural Science Prize of the Ministry of Education of China in 2013 and the National Science Fund for Excellent Young Scholars of China in 2014.

Lijun Zhang is a research professor at State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences (ISCAS). He is also the director of the Sino-Europe Joint Institute of Dependable and Smart Software at the Institute of Intelligent Software in Guangzhou. Before this he was a postdoctoral researcher at University of Oxford, and then an associate professor at Language-Based Technology section, DTU Compute, Technical University of Denmark. He gained a Diploma Degree and a PhD (Dr. Ing.) at Saarland University. His research interests include: probabilistic model checking, simulation reduction and decision algorithms, abstraction and model checking, learning algorithms, and verification of deep neural networks. He is leading the development of several tools including PRODeep for verifying neural networks, ROLL for learning automata, and the model checker ePMC, previously known as IscasMC.