Utrecht University

Artificial Intelligence Master Thesis

r-PLBP: Temporal Logic for Reasoning about Safety and Rewards of Bounded Policies under Uncertainty

By Sterre Lutz (7767927)

Daily Supervisor:	Nima Motamed, Utrecht University
Supervisor:	Dr. Dragan Doder, Utrecht University
Second Examiner:	Dr. Natasha Alechina, Utrecht University

September 12, 2023

Abstract

Temporal logics can be used to reason about how environments change over time. When environments have an element of uncertainty, agents with the power to make decisions, or an opportunity for gaining rewards, reasoning about such factors requires a temporal logic with the vocabulary to address them.

We propose rPLBP, a reward-based extension of Probabilistic Logic of Bounded Policies that can reason about the rewards gained by an agent in a probabilistic environment. This logic includes expressions to reason about rewards accumulated along a finite *path*, as well as the expected reward given a finite *strategy* (a sort of playbook describing which actions the agent will take for a finite number of future steps). We show that the model checking problem for rPLBP is PSPACE-complete. Moreover, we prove that the satisfiability problem for rPLBP is in 2EXPSPACE.

We explore applications of rPLBP in the field of safe reinforcement learning. In shielded learning, safety constraints are used to block unsafe actions during the learning phase of an agent. Unlike the temporal logics usually used for such constraints – e.g. Linear Temporal Logic and Probabilistic Computation Tree Logic – the logic rPLBP allows us to express to what extent safety guidelines may be relaxed based on expected reward.

Contents

Chapter 1

Introduction

Reinforcement learning (RL) is a machine learning technique where desirable behaviour of an agent that is learning is rewarded [Kaelbling et al., 1996]. Given that the agents are tasked with maximising the total reward, this *reinforces* desirable behaviour, hence the name. It can be compared to how humans, but especially children learn: through experience. You touch the hot oven and your finger hurts? Then you are unlikely to do it again. While generally an effective method of learning, there are scenarios where imposing constraints on behaviour while learning is beneficial, e.g. when behaviour can cause financial or physical harm in the real world.

The need for RL algorithms that can guarantee or promote safety during the learning phase, has led to an impulse in research on *safe* RL [Garcıa and Fernández, 2015]. A promising technique within this field is shielded learning [Odriozola-Olalde et al., 2023]. In shielded learning, safety constraints that are specified in advance are used to verify which actions are safe to take. If the learning agent then proposes an unsafe action (according to the constraints), the shield blocks the action and forces the agent to choose a different action.

To specify the safety constraints, most shielding techiques make use of some temporal logic, a language that can express how properties change over time. Most commonly used is the language LTL [Pnueli, 1977], which allows the engineer to assert requirements on all future paths that can result from the action under review [Alshiekh et al., 2018]. However, probabilistic shielding using PCTL specifications – a probabilistic language for expressing that a certain property must hold with at least a certain probability – has also proven to be feasible and effective on a set of benchmark RL tasks [Jansen et al., 2020].

In this thesis, we explore whether it is theoretically possible and useful to practice shielding with specifications written in a more expressive temporal logic that can reason about policies and rewards. For example, consider an environment where it is dangerous for the agent to be near the many pits in the ground, but many of the goal states (and therefore the rewarding states) are close to some of those pits. A safety requirement that simply states to always keep away from any pit may be too restrictive to be able to successfully solve the task. Rather, we could express that the agent may come closer to the pit, but *only* when the expected reward of the policy is high enough.

In order to express such *reward-constrained requirements*, a specification logic that has the 'vocabulary' to assert properties concerning both policies and rewards is required. However, the shield must still be able to check the safety specifications for any action within an amount of time and space comparable to that of currently used shielding languages. To this end, we introduce the logic rPLBP, an extension of Probabilistic Logic of Bounded Policies (PLBP) [Motamed et al., 2023]. While rPLBP extends PLBP with policy formulas and reward operators and increases the expressivity significantly, we prove that the model-checking and satisfiability problems remain in the same complexity classes (PSPACE-complete and 2EXPSPACE respectively). Since the commonly used LTL also has PSPACE model checking, we conclude that reward-contingent shielding with rPLBP is feasible. Moreover, we propose theoretical examples of how rPLBP safety requirements may be formulated to improve the balance between safety and performance in RL tasks.

In summary, the aim of this thesis is to further the field of safe RL by presenting and analysing a new

specification language that can be used to balance the safety and rewards of policies of RL agents. This thesis is structured as follows. Chapter 2 provides the necessary background on temporal logic and RL, culminating in an analysis of the requirements for a shielding specification language. Chapter 3 presents rPLBP, introducing and justifying new operators and concepts, and concluding with the complete syntax and semantics. Chapters 4 and 5 contain all proofs pertaining to the complexity of the model-checking and satisfiability problems, respectively. Chapter 6 illustrates theoretical examples of specifying shielding requirements using rPLBP in real RL task environments. Chapter 7 summarises the thesis results, discusses potential objections, and suggests future research in this area.

Chapter 2

Background

In this chapter, we discuss the necessary background for this thesis. First, different ways of modelling environments that change over time are discussed. Second, a survey of temporal logics is provided, to give insight into existing ways to reason about (non-)stochastic environments. Third, a brief introduction to reinforcement learning and shielded learning is given, followed by an analysis of which characteristics of the discussed temporal logics are suitable to this application of shielded learning.

2.1 Modelling

The truth, or value, of any logical statement is evaluated against some model: an abstract specification of a (fictional) world. For propositional logic for example, the model is a specific truth assignment for a set of propositional variables. In the case of temporal logics, statements are made about environments that change throughout time. Therefore, the models for temporal logic need to encode information about the way the environment (and sometimes the agent's place in it) changes over time: the environment's dynamics. We want to do this in a way that abstracts away from the physical details as much as possible, so that the movements of a robot through a maze can be described with the same model as the movements of a submarine in the ocean. This makes the temporal logics widely applicable without the need for adapting the language of the logic or the models for individual environments.

2.1.1 Environments

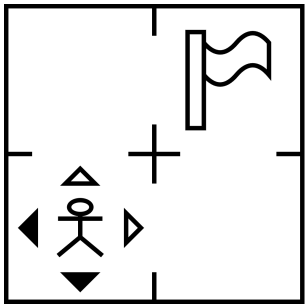
This section cover three different models, each of them gradually integrating more details about the underlying environment compared to the previous model: state transition systems, Markov chains, and Markov decision processes. As in [Motamed et al., 2023], we use the following notation: $f : X \rightharpoonup Y$ denotes a partial function from X to Y, where $f(x)\downarrow$ denotes that f is defined for element $x \in X$. For a set X we use the notation $X_x^{\leq k} = \{x_1 \dots x_n \mid x_1 = x \text{ and } 1 \leq n \leq k\}$ and $\Delta(X)$ for the set of all probability distributions $D : X \to [0, 1]$ where $D(x) > 0$ for finitely many x.

State Transition Systems

State transition systems describe an environment by defining states, each with its own truth assignment of propositions, and transitions between them. These transitions are non-deterministic, meaning that the model may give multiple possible transitions in each state, without offering information about probability distributions or the influence of the agent.

▶ **Definition 1 (State transition system).** Given an infinite set of propositional variables Prop, a *state transition system* over Prop is a tuple $\langle S, \longrightarrow, V \rangle$, with a non-empty set of *states* S, a *transition relation* between states $\longrightarrow \subseteq S \times S$, and a *valuation* function $V : S \to 2^{\mathsf{Prop}}$.

The valuation function describes which propositional variables hold in each state in S, while the transition relation shows which pairs of states can be transitioned between. Thus, if propositional variable p is in the set $V(s)$ then p holds in state s, and if $s \longrightarrow t$ then the system may transition from state s to t.



▶ **Figure 2.1:** Example environment used for illustrating different models: a 2 by 2 grid across which a robot can move. The top-right square contains a flag, which represents a target to reach.

▶ **Example 1 (Robot on a grid).** Suppose there is a grid of 2 by 2 squares, of which the top-right square contains a flag (visualised in Figure 2.1). In each time step, a robot is situated in one of these squares. From one time step to the next, the robot can move to an adjacent square or stay put. We can model this with propositional variables $\mathsf{Prop} = \{\mathsf{atLeft}, \mathsf{atBottom}, \mathsf{atFlag}\}$ denoting the robot being in the left column, the bottom row, and in the square with the flag respectively[1]. To create a state transition system over this Prop, we choose four states where each represents the robot being in one of the four squares: s_0 for the bottom-left square, s_1 for the top-left square, s_2 for the bottom-right square, and s_3 for the top-right square (with the flag). We can then define the state transition system as $M = \langle S, \longrightarrow, V \rangle$, where:

- $S = \{s_0, s_1, s_2, s_3\}$,
- $\longrightarrow = \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_1, s_1), (s_1, s_0), (s_1, s_3), (s_2, s_2), (s_2, s_0), (s_2, s_3), (s_3, s_3), (s_3, s_1), (s_3, s_2)\}$,
- $V(s_0) = \{\mathsf{atLeft}, \mathsf{atBottom}\}$, $V(s_1) = \{\mathsf{atLeft}\}$, $V(s_2) = \{\mathsf{atBottom}\}$, $V(s_3) = \{\mathsf{atFlag}\}$.

A diagram of M is given in Figure 2.2a.

State transition systems are useful models for temporal logics where we are only interested in what *may* happen, without knowledge of or interest in specifics about how likely certain paths are or how actions influence them.

Markov Chains

If we *are* interested in modelling the probability of transitioning between states, we can use Markov chains instead. Again, such a model contains a set of states with valuations, but now the transitions between states are all assigned a probability. A key property of any Markov model is that these probabilities only depend on the current state, not on the history.

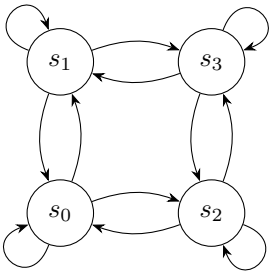
▶ **Definition 2 (Markov chain).** Given an infinite set of propositional variables Prop, a *Markov chain* over Prop is a tuple $\langle S, P, V \rangle$, where S is a non-empty set of states, $P : S \to \Delta(S)$ is a *probabilistic transition function*, and $V : S \to 2^{\mathsf{Prop}}$ is a valuation function.

In contrast to the transition relation from Definition 1 – which expressed only which transitions are possible – the probabilistic transition function of a Markov chain details the probability of transitioning between states: e.g. $P(s)(t) = \frac{1}{2}$ means that the probability of transitioning from state s to t is $\frac{1}{2}$.

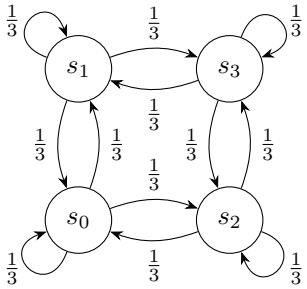
▶ **Example 2 (Robot on a grid, continued).** Suppose we know that the robot in our environment from Example 1 moves between squares with a certain probability. For example, with a probability of $\frac{1}{3}$ it remains in the current square, and it moves to both adjacent squares with a probability of $\frac{1}{3}$ respectively. Using the same Prop as we did in the Example 1, we can specify a Markov chain model $M = \langle S, P, V \rangle$ that describes this probabilistic environment, with:

- $S = \{s_0, s_1, s_2, s_3\}$;

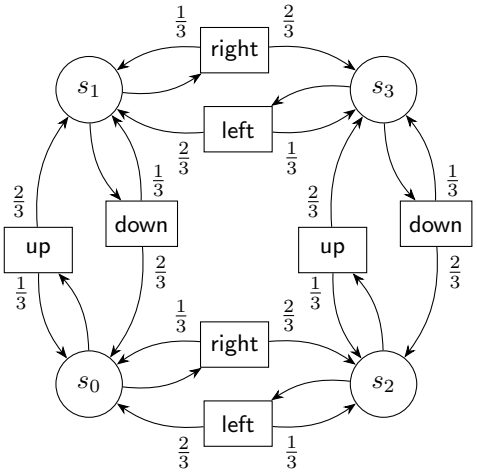
[1] Note that we do not require propositional variables for the robot being in the right column and top row, since in this can be expressed by $\neg\mathsf{atLeft}$ and $\neg\mathsf{atBottom}$ respectively. In the top-right square, for example, we have that $\neg\mathsf{atLeft} \wedge \neg\mathsf{atBottom}$ holds.



(a) State transition system of a robot on a 2-by-2 grid, as specified in Example 1. In any square, the robot can move into directly adjacent squares, or remain in the same place.



(b) Markov chain of a robot on a 2-by-2 grid, as specified in Example 2. The robot remains in the current square with a probability of $\frac{1}{3}$ and moves to either adjacent square with a probability of $\frac{1}{3}$ respectively.



(c) Markov decision process of a robot on a 2-by-2 grid, as specified in Example 4. In any square, the robot can choose to move toward one of the two adjacent squares. This action succeeds (i.e. the robot moves in the intended direction) with a probability of $\frac{2}{3}$. This action fails (i.e. the robot remains in the current square) with a probability of $\frac{1}{3}$.

▶ **Figure 2.2:** The dynamics of a robot moving on a 2-by-2 grid modelled with different levels of abstraction: a state transitions system, a Markov chain, and a Markov decision process. In each model, the four states s_0, s_1, s_2, and s_3 represent the robot being in the bottom-left, top-left, bottom-right, and top-right squares of the grid respectively.

- $P(s_0)(s_0) = \frac{1}{3}, P(s_0)(s_1) = \frac{1}{3}, P(s_0)(s_2) = \frac{1}{3}, P(s_0)(s_3) = 0$, etc.;
- $V(s_0) = \{\mathsf{atLeft}, \mathsf{atBottom}\}$, $V(s_1) = \{\mathsf{atLeft}\}$, $V(s_2) = \{\mathsf{atBottom}\}$, $V(s_3) = \{\mathsf{atFlag}\}$.

The Markov chain M is drawn in Figure 2.2b.

Using Markov chains to model environments where an agent has a choice between actions, is only possible when we have information about with which probability the agent chooses each action. Therefore, they are most suitable for modelling environments with very predictable agents or no agents at all.

Markov Decision Processes

For environments with an agent that can influence what happens in the environment, it is useful to model for each state what the possible actions of the agent are, and what effect they have on the probability of transitioning into another state. A Markov decision process (MDP) can model such an environment. They are similar to Markov chains, with the addition of actions that decide the probability distribution over the transitions out of a state.

We largely follow the definition of actions from [Motamed et al., 2023], since it allows us to give detailed information about when actions are allowed and what their effects can be.

▶ **Definition 3 (Action set).** Given an infinite set of propositional variables Prop, an *action set* is a finite set of actions $\mathcal{A}$, where each action $a \in \mathcal{A}$ is associated with a *precondition* pre_a, a conjunction of literals over Prop, and *postconditions* Post_a, a finite non-empty set of conjunctions of literals over Prop. An action set must satisfy the following requirements for all $a \in \mathcal{A}$:

(i) for all $\mathsf{post}_{a,i}, \mathsf{post}_{a,j} \in \mathsf{Post}_a$ such that $i \neq j$, the negation of at least one literal in $\mathsf{post}_{a,i}$ must be in $\mathsf{post}_{a,j}$;

(ii) for all $\mathsf{post}_{a,i} \in \mathsf{Post}_a$, there exists at least one action $a' \in \mathcal{A}$ such that for all literals in $\mathsf{post}_{a,i}$, its negation is not in $\mathsf{pre}_{a'}$.

A precondition dictates what must be satisfied before an action is performed, while the postconditions are all the possible outcomes of an action. Requirement (i) states that all postconditions of an action must be mutually inconsistent, and therefore refer to different outcomes. Requirement (ii) states that for each possible outcome of an action, there must always be an action that can be taken, thus preventing deadlocks.

▶ **Example 3 (Robot on a grid, continued).** Suppose that the robot in our example has four actions in total to perform within this environment: moving up, down, left, or right. We let $\mathcal{A} = \{\mathsf{up}, \mathsf{down}, \mathsf{left}, \mathsf{right}\}$. Suppose that the agent cannot choose an action that moves it off of the grid. For example, the action up is only executable when the robot is in the bottom row of the grid. Thus we define the preconditions as

$$\begin{aligned} \mathsf{pre}_{\mathsf{up}} &= \mathsf{atBottom}, \\ \mathsf{pre}_{\mathsf{down}} &= \neg\mathsf{atBottom}, \\ \mathsf{pre}_{\mathsf{right}} &= \mathsf{atLeft}, \\ \mathsf{pre}_{\mathsf{left}} &= \neg\mathsf{atLeft}. \end{aligned} \tag{2.1}$$

For any action, there are always two outcomes: either the robot has moved in the intended direction or remained in its place. This means that for the horizontal actions left and right, the two outcomes can be distinguished by whether atLeft holds, and for the vertical actions up and down, the two outcomes can be distinguished by whether $\mathsf{atBottom}$ holds. Thus, the postconditions can be defined as

$$\begin{aligned} \mathsf{Post}_{\mathsf{up}} &= \{\mathsf{atBottom}, \neg\mathsf{atBottom}\}, \\ \mathsf{Post}_{\mathsf{down}} &= \{\mathsf{atBottom}, \neg\mathsf{atBottom}\}, \\ \mathsf{Post}_{\mathsf{right}} &= \{\mathsf{atLeft}, \neg\mathsf{atLeft}\}, \\ \mathsf{Post}_{\mathsf{left}} &= \{\mathsf{atLeft}, \neg\mathsf{atLeft}\}. \end{aligned} \tag{2.2}$$

Note that it can easily be verified that $\mathcal{A}$ satisfies the requirements from Definition 3: all the postconditions of an action are mutually inconsistent and each postcondition is consistent with at least one action's precondition.

Now that we have a way to model the actions of an agent, we can proceed to modelling the effects of those actions on an environment. There are many different definitions of MDPs, depending on the application. One such definition, which is used throughout this thesis, is the following.

▶ **Definition 4 (Markov decision process).** Given an infinite set of propositional variables Prop and an action set $\mathcal{A}$, a *Markov decision process* (MDP) is a tuple $\langle S, P, V\rangle$, where S is a non-empty set of states, where $P : S \times \mathcal{A} \rightharpoonup \Delta(S)$ is a *partial probabilistic transition function*, and where $V : S \to 2^{\mathsf{Prop}}$ is a valuation function. An MDP must satisfy the following requirements:

(i) *(deadlock-free)* for all s in S there exists some a in $\mathcal{A}$ such that $M, s \vDash \mathsf{pre}_a$,

(ii) *(pre)* $P(s,a)\downarrow$ iff $s \vDash \mathsf{pre}_a$ for all $s \in S$ and $a \in \mathcal{A}$, and

(iii) *(post)* given an $s \in S$ and $a \in \mathcal{A}$ such that $P(s,a)\downarrow$, it holds that

(iii.i) for all $s' \in S$ where $P(s,a)(s') > 0$, there is exactly one $\mathsf{post}_{a,i} \in \mathsf{Post}_a$ such that $M, s' \vDash \mathsf{post}_{a,i}$, and

(iii.ii) for all $\mathsf{post}_{a,i} \in \mathsf{Post}_a$ there exists exactly one s' such that $P(s,a)(s') > 0$ and $M, s' \vDash \mathsf{post}_{a,i}$.

The probabilistic transition function of an MDP gives the probability distribution over all states in S that results from executing an action in a state. For example, $P(s,a)(t) = \frac{1}{2}$ denotes that executing action a in s results in state t with a probability of $\frac{1}{2}$. The function is partial since $P(s,a)$ is only defined (denoted $P(s,a)\downarrow$) if action a is possible/allowed in state s.

▶ **Example 4 (Robot on a grid, continued).** We can model our robot on a grid example as an MDP as well, using the action set $\mathcal{A}$ defined in Example 3. Suppose that in any square, the robot can choose to move towards either of the adjacent squares (e.g. in the bottom left square, the robot can choose between actions up and right). Moving to an adjacent space only succeeds with a probability of $\frac{2}{3}$. With a probability of $\frac{1}{3}$, the robot remains in the current space instead. For actions that move in the direction of the wall, the probability distribution function is not defined (e.g. moving left in the bottom left square). We can define an MDP $M = \langle S, P, V\rangle$ over $\mathcal{A}$ modelling this environment, where

- $S = \{s_0, s_1, s_2, s_3\}$;
- $P(s_0, \mathsf{up})(s_0) = \frac{1}{3}, P(s_0, \mathsf{up})(s_1) = \frac{2}{3}, P(s_0, \mathsf{right})(s_0) = \frac{1}{3}, P(s_0, \mathsf{right})(s_2) = \frac{2}{3}$, etc.;
- $V(s_0) = \{\mathsf{atLeft}, \mathsf{atBottom}\}$, $V(s_1) = \{\mathsf{atLeft}\}$, $V(s_2) = \{\mathsf{atBottom}\}$, $V(s_3) = \{\mathsf{atFlag}\}$.

The MDP M is given in Figure 2.2c. Note that this M satisfies the requirements from Definition 4. Firstly, for all states of M there is at least one action executable (satisfying requirement (i)). Secondly, for each action we have that it is executable in a state according to its precondition, exactly when the probability distribution function is defined for this state-action pair (satisfying requirement (ii)). Lastly, for all state-action pairs, each of the postconditions corresponds to exactly one outcome state, and vice-versa (satisfying requirement (iii)).

Markov decision process with rewards

There are several ways to add rewards to an MDP that is defined over a set of actions $\mathcal{A}$. In general, we define a reward function R that maps to values in $\mathbb{R}$ (where negative values represent costs), but the domain from which is mapped varies from application to application. If only the outcome of an action matters for the reward, for example, the function can be defined as $R : S \to \mathbb{R}$. However, in many environments, the state from which the action was taken matters as well. Moreover, sometimes the action itself also has an impact on the reward. To accommodate for all of these options, we define the reward function to depend on the origin and outcome node, as well as the action.

▶ **Definition 5 (Markov decision process with rewards).** Given an infinite set of propositional variables Prop and an action set $\mathcal{A}$, a *Markov decision process with rewards* is a tuple $\langle S, P, V, R\rangle$, where S is a non-empty set of states, $P : S \times \mathcal{A} \rightharpoonup \Delta(S)$ is a partial probabilistic transition function, $V : S \to 2^{\mathsf{Prop}}$ is a valuation function, and $R : S \times \mathcal{A} \times S \to \mathbb{R}$ is a *reward function*. When it is clear from context, an MDP with rewards is referred to as an MDP. An MDP with rewards must satisfy the following requirements:

(i') *(deadlock-free)* for all s in S there exists some a in $\mathcal{A}$ such that $M, s \vDash \mathsf{pre}_a$,

(ii') *(pre)* $P(s,a)\downarrow$ iff $s \vDash \mathsf{pre}_a$ for all $s \in S$ and $a \in \mathcal{A}$, and

(iii') *(post)* given an $s \in S$ and $a \in \mathcal{A}$ such that $P(s,a)\downarrow$, it holds that

(iii.i') for all $s' \in S$ where $P(s,a)(s') > 0$, there is exactly one $\mathsf{post}_{a,i} \in \mathsf{Post}_a$ such that $M, s' \vDash \mathsf{post}_{a,i}$, and

(iii.ii') for all $\mathsf{post}_{a,i} \in \mathsf{Post}_a$ there exists exactly one s' such that $P(s,a)(s') > 0$ and $M, s' \vDash \mathsf{post}_{a,i}$.

▶ **Example 5 (Robot on a grid, continued).** To give an example of an MDP with rewards on which an r-PLBP formula can be evaluated, we can reuse the MDP M constructed in Example 5 to model a robot on a grid and extend it with rewards. Suppose that the robot receives a reward of 10 whenever it successfully transitions onto the top-right square with the flag. We use the action set $\mathcal{A}$ from Example 3. We define an MDP with rewards as $M = \langle S, P, V, R\rangle$ where

- $S = \{s_0, s_1, s_2, s_3\}$;
- $P(s_0, \mathsf{up})(s_0) = \frac{1}{3}, P(s_0, \mathsf{up})(s_1) = \frac{2}{3}, P(s_0, \mathsf{right})(s_0) = \frac{1}{3}, P(s_0, \mathsf{right})(s_2) = \frac{2}{3}$, etc.;
- $V(s_0) = \{\mathsf{atLeft}, \mathsf{atBottom}\}$, $V(s_1) = \{\mathsf{atLeft}\}$, $V(s_2) = \{\mathsf{atBottom}\}$, $V(s_3) = \{\mathsf{atFlag}\}$;
- R : $R(s,a)(s_3) = 10$, for all $s \in S$ and $a \in \mathcal{A}$, and $R(s,a)(s') = 0$ for all $s \in S$, $a \in \mathcal{A}$, and $s' \in S \backslash \{s_3\}$.

Note that this M satisfies the requirements from Definition 5. Firstly, for all states of M there is at least one action executable (satisfying requirement (i)). Secondly, for each action we have that it is executable in a state according to its precondition, exactly when the probability distribution function is defined for this state-action pair (satisfying requirement (ii). Lastly, for all state-action pairs, each of the postconditions corresponds to exactly one outcome state, and vice-versa (satisfying requirement (iii)).

2.2 Temporal Logic

Temporal logics are logical languages used to reason formally about environments that change over time. In this section, we first look at a selection of basic languages to introduce the elementary concepts and operators used in the field of temporal logic. After this first introduction, we proceed to a survey of temporal logics that have means to reason about matters such as probability and reward.

2.2.1 Linear and branching time

We discuss four logics that can be considered to be at the core of the family of temporal logics. By explaining these, the most common temporal operators are explained, as well as the fundamental difference between linear and computation tree approaches, and the effect of bounding the expressions.

Linear-time Temporal Logic (LTL)

First introduced as a language for computer programs [Pnueli, 1977], Linear-time Temporal Logic (LTL[2]) is a logic that can express properties about a specific infinite *path* in a state transition system as introduced in Section 2.1.1. Formally, a path $\lambda = s_1 s_2 \ldots$, in $M = \langle S, \longrightarrow, V\rangle$ is a sequence of states from S, where $s_i \longrightarrow s_{i+1}$ for all i. We use the following conventional shorthand, given a path $\lambda = s_1, s_2, \ldots$:

- $\lambda[i] = s_i$ (i.e. the ith state of the path),
- $\lambda[i \ldots j] = s_i, s_{i+1}, \ldots, s_{j-1}, s_j$ (i.e. the sub-path of λ from $\lambda[i]$ until $\lambda[j]$), and
- $\lambda[i \ldots \infty] = s_i, s_{i+1}, \ldots$ (i.e. the tail of λ starting at $\lambda[i]$).

Given a set of atomic propositions Prop and an element $p \in \mathsf{Prop}$, the syntax of LTL is defined as

$$\gamma ::= p \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathsf{X}\gamma \mid \gamma \mathsf{U} \gamma.$$

[2] In some literature, LTL is referred to as PLTL, where indicating that it is at its core a propositional logic. Since we later introduce a probabilistic logic that uses this acronym, we use the acronym LTL here.

Temporal operators X ('next') and U ('until'), are usually used to define two additional operators: G ('always') and F ('finally'). The semantics of LTL is defined as

$$
\begin{array}{lll}
\lambda \vDash p & \text{iff} & p \in V(\lambda[1]); \\
\lambda \vDash \neg\gamma & \text{iff} & \lambda \nvDash \gamma; \\
\lambda \vDash \gamma_1 \wedge \gamma_2 & \text{iff} & \lambda \vDash \gamma_1 \text{ and } \lambda \vDash \gamma_2; \\
\lambda \vDash \mathsf{X}\gamma & \text{iff} & \lambda[2 \ldots \infty] \vDash \gamma; \\
\lambda \vDash \gamma_1 \mathsf{U} \gamma_2 & \text{iff} & \lambda[i \ldots \infty] \vDash \gamma_2 \text{ for some } i \geq 1, \text{ and } \lambda[j \ldots \infty] \vDash \gamma_1 \text{ for all } 1 \leq j < i; \\
\lambda \vDash \mathsf{F}\gamma & \text{iff} & \lambda[i \ldots \infty] \vDash \gamma \text{ for some } i \geq 1; \\
\lambda \vDash \mathsf{G}\gamma & \text{iff} & \lambda[i \ldots \infty] \vDash \gamma \text{ for all } i \geq 1.
\end{array}
$$

▶ **Example 6 (Robot on a grid, continued).** Now let us look at what LTL allows us to express about our example environment of a robot on a grid from section 2.3, modelled by state transition system $M = \langle S, \longrightarrow, V \rangle$ in Example 1. Suppose we wish to investigate the characteristics of the path that is taken when the robot repeatedly goes up and down from the starting state s_0: this gives us $\lambda = s_0 s_1 s_0 s_1 \ldots$, repeating infinitely. For this path, we can express (among many others) the following characteristics using LTL:

- $\lambda \vDash \mathsf{atBottom} \wedge \mathsf{atLeft}$ – in the first step of λ, the robot is in the bottom-left square;
- $\lambda \vDash \mathsf{F}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft})$ – at some point in λ, the robot is in the top-left square;
- $\lambda \vDash \mathsf{G}((\neg\mathsf{atBottom} \wedge \mathsf{atLeft}) \vee \mathsf{X}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft}))$ – at any step in λ, the robot is either currently in the top-left square, or it will be there in the next step;
- $\lambda \nvDash \mathsf{F}(\mathsf{atFlag})$ – it is *not* true for λ that the robot will be in the square with the flag at some point in time.

Linear-time Temporal Logic over Finite Traces (LTL_f)

In the previous section we have seen that LTL is evaluated over infinite paths. Describing those paths exactly can be difficult, especially if there is not an obvious pattern that can be described. Moreover, there are practical applications for which evaluation up to some finite bound is sufficiently powerful. To this end LTL_f is presented by [De Giacomo and Vardi, 2013]. It uses the same syntax as LTL: given a set of atomic propositions Prop and an element $p \in \mathsf{Prop}$, the syntax of LTL_f is defined as

$$\gamma ::= p \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathsf{X}\gamma \mid \gamma \mathsf{U} \gamma.$$

The semantics are expressed by [De Giacomo and Vardi, 2013] in terms of *finite* paths. For a finite path $\lambda = s_1 s_2 \ldots s_k$ the specific elements and sub-paths are denoted as before ($\lambda[i]$ and $\lambda[i \ldots j]$ respectively), but we add the notation $\mathsf{last}(\lambda) = k$.

Given a finite path λ in state transition system $M = \langle S, \longrightarrow, V \rangle$, the semantics of LTL_f are defined as

$$
\begin{array}{lll}
\lambda \vDash p & \text{iff} & p \in V(\lambda[1]); \\
\lambda \vDash \neg\gamma & \text{iff} & \lambda \nvDash \gamma; \\
\lambda \vDash \gamma_1 \wedge \gamma_2 & \text{iff} & \lambda \vDash \gamma_1 \text{ and } \lambda \vDash \gamma_2; \\
\lambda \vDash \mathsf{X}\gamma & \text{iff} & \lambda[2 \ldots \mathsf{last}(\lambda)] \vDash \gamma \text{ and } \mathsf{last}(\lambda) \neq 1; \\
\lambda \vDash \gamma_1 \mathsf{U} \gamma_2 & \text{iff} & \lambda[i \ldots \mathsf{last}(\lambda)] \vDash \gamma_2 \text{ for some } i \text{ where } 1 \leq i \leq \mathsf{last}(\lambda), \text{ and } \lambda[j \ldots \mathsf{last}(\lambda)] \vDash \gamma_1 \\
 & & \text{for all } j \text{ where } 1 \leq j < i; \\
\lambda \vDash \mathsf{F}\gamma & \text{iff} & \lambda[i \ldots \mathsf{last}(\lambda)] \vDash \gamma \text{ for some } i \text{ where } 1 \leq i \leq \mathsf{last}(\lambda); \\
\lambda \vDash \mathsf{G}\gamma & \text{iff} & \lambda[i \ldots \mathsf{last}(\lambda)] \vDash \gamma \text{ for all } i \text{ where } 1 \leq i \leq \mathsf{last}(\lambda).
\end{array}
$$

By using the term $\mathsf{last}(\lambda)$ as an upper bound in the semantics this way, the LTL semantics are adapted intuitively to only evaluate the truth of expressions up to and including the last state in the path. This does have the counter-intuitive consequence that tautologies like $p \vee \neg p$ are false in time steps beyond the length of the path: $s_a s_b \nvDash \mathsf{XX}(p \vee \neg p)$.

Computation Tree Logic (CTL and CTL*)

Both LTL and LTL_f are evaluated over single paths within state transition systems, making them *linear-time* logics. For some applications, these logics are not expressive enough: it might be that it is not fully in our control which path is taken. For example, in systems where the agent can make choices or where there is some randomness involved. In those cases, we would ideally make more general statements about what *may* happen in some paths or what *must* happen in all paths.

To that end, logics were introduced that are based on *branching-time*, which encompasses all possible paths that can branch from a specific state within such a system. Computation Tree Logic (CTL*) [Emerson and Halpern, 1986] is such a logic, which distinguishes between *path formulas* and *state formulas*, expressing properties of paths and states respectively. Part of the state formulas are the path quantifiers $\mathsf{E}\gamma$ and $\mathsf{A}\gamma$ expressing respectively the existence of a path where the path formula γ is true, or that γ is true for all paths.

The syntax of CTL* for state formulas φ and path formulas γ is defined as

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\gamma,$$

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathsf{X}\gamma \mid \gamma\mathsf{U}\gamma.$$

The universal path quantifier A is defined as the shorthand $\mathsf{A}\gamma \equiv \neg\mathsf{E}\neg\gamma$. The temporal operators G and F are also defined from X and U as before. Given a state transition system $M = \langle S, \longrightarrow, V\rangle$, and a state $s \in S$, the semantics of the state formulas in CTL* are defined as follows

$M, s \vDash p$	iff	$p \in V(s)$;
$M, s \vDash \neg\varphi$	iff	$M, s \nvDash \varphi$;
$M, s \vDash \varphi_1 \wedge \varphi_2$	iff	$M, s \vDash \varphi_1$ and $M, s \vDash \varphi_2$;
$M, s \vDash \mathsf{E}\gamma$	iff	there is a path λ in M s.t. $\lambda[1] = s$ and $M, \lambda \vDash \gamma$;
$M, s \vDash \mathsf{A}\gamma$	iff	for all paths λ in M where $\lambda[1] = s$, it holds that $M, \lambda \vDash \gamma$.

The semantics of the path formulas in CTL* are nearly identical to LTL, with the exception of a state formula occurring in a path formula. Given a path λ in a state transition system $M = \langle S, \longrightarrow, V\rangle$:

$M, \lambda \vDash \varphi$	iff	$M, \lambda[1] \vDash \varphi$;
$M, \lambda \vDash \neg\gamma$	iff	$M, \lambda \nvDash \gamma$;
$M, \lambda \vDash \gamma_1 \wedge \gamma_2$	iff	$M, \lambda \vDash \gamma_1$ and $M, \lambda \vDash \gamma_2$;
$M, \lambda \vDash \mathsf{X}\gamma$	iff	$M, \lambda[2 \ldots \infty] \vDash \gamma$;
$M, \lambda \vDash \gamma_1\mathsf{U}\gamma_2$	iff	$M, \lambda[i \ldots \infty] \vDash \gamma_2$ for some $i \geq 1$, and $M, \lambda[j \ldots \infty] \vDash \gamma_1$ for all $1 \leq j < i$;
$M, \lambda \vDash \mathsf{F}\gamma$	iff	$M, \lambda[i \ldots \infty] \vDash \gamma$ for some $i \geq 1$;
$M, \lambda \vDash \mathsf{G}\gamma$	iff	$M, \lambda[i \ldots \infty] \vDash \gamma$ for all $i \geq 1$.

▶ **Example 7 (Robot on a grid, continued).** The language CTL* allows us to express more general statements about the state transition system M from Example 1 modelling a robot on a 2 by 2 grid. For this model and state s_0, we can express the following properties in CTL*:

- $M, s_0 \vDash \mathsf{atBottom} \wedge \mathsf{atLeft}$ – in the state s_0, the robot is in the bottom-left square;
- $M, s_0 \vDash \mathsf{EG}(\mathsf{atBottom} \wedge \mathsf{atLeft})$ – there exists a path starting in s_0, for which the robot is always in the bottom-left square;
- $M, s_0 \vDash \mathsf{AGE}((\neg\mathsf{atBottom} \wedge \mathsf{atLeft}) \vee \mathsf{X}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft}) \vee \mathsf{XX}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft}))$ – in any state the robot can reach from s_0, the robot is never more than two steps away from the top-left square;
- $M, s_0 \nvDash \mathsf{AF}(\mathsf{atFlag})$ – it is *not* true for all paths starting in s_0 that the robot will eventually reach the flag in the top-right square.

The "vanilla" version of Computation Tree Logic, CTL, restricts CTL* ("star") by demanding that each path quantifier is immediately followed by exactly one temporal operator [Clarke and Emerson, 1981]. For example, the expression $\mathsf{EXX}\gamma$ is allowed in CTL*, but not in CTL. While this limits the range of

expressions, it does make checking whether a formula holds for a particular model a less time-consuming task. Depending on the application, the restricted version may just be expressive enough, in which case it is preferable to use it in an application where formulas need to be verified. This trade-off is similar to the restriction of LTL to the finite LTL_f.

The semantics of CTL are identical to CTL* – though of course only defined for the syntactically valid parts of CTL* – and therefore omitted here. The example expressions from the CTL* section mostly apply to CTL as well. The notable exception is the third example:

$$M, s_0 \vDash \mathsf{AGE}((\neg\mathsf{atBottom} \wedge \mathsf{atLeft}) \vee \mathsf{X}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft}) \vee \mathsf{XX}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft}))$$

The stacking of the next operator in $\mathsf{XX}(\neg\mathsf{atBottom} \wedge \mathsf{atLeft})$ is not allowed, as well as the lack of temporal operator following the E operator. It is however important to note, that for any model M, state s, and CTL formula φ, it holds that $M, s \vDash \varphi$ in CTL iff $M, s \vDash \varphi$ in CTL*.

2.2.2 Probability

As explained in Section 2.1.1, there are cases where we have precise information about the probability of transitioning between states in our environment, either modelled with Markov chains or MDPs. If we are interested in reasoning about temporal properties in a probabilistic way, the CTL/CTL* operators that say that something must 'always', or 'never' occur are too simplistic. Rather, we would like to express (a bound on) the probability that something occurs. This is the reason probabilistic extensions of logics in the previous section were introduced. We first discuss a probabilistic extension of CTL/CTL*, PCTL/pCTL*, which reasons about Markov chains. Subsequently, we examine a of PCTL/pCTL*, based on MDPs, as well as a bounded variant of PCTL/pCTL*. Finally, we discuss PLBP, a logic with which we can reason about bounded policies.

Probabilistic Computation Tree Logic (PCTL and pCTL*)

As explained in Section 2.2.1, the branching-time logics CTL and CTL* allow us to express whether paths with certain temporal properties or patterns exist, or are even universal (true for all paths). The probabilistic extensions PCTL [Hansson and Jonsson, 1994] and pCTL* [Aziz et al., 1995] allow us to further specify the probability of such paths. For determining these probabilities, we need to know more about the probability of the transitions in the underlying system. Therefore, PCTL and pCTL* are evaluated against Markov chains.

Like CTL*, the syntax of pCTL* is made up of state formulas and path formulas. The existential path quantifier, however, is replaced by a set probability operators $\mathsf{P}_{\bowtie c}$ for $\bowtie \in \{<, >, =\}$ and $c \in [0, 1] \cap \mathbb{Q}$. The syntax of state formulas φ in pCTL* is defined as

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{P}_{\bowtie c}(\gamma),$$

where p is a propositional variable. The formula $\mathsf{P}_{\bowtie c}(\gamma)$ indicates that the probability of the path formula γ being true in the paths branching from the current state, is $\bowtie c$. The syntax of path formulas γ in pCTL* is defined as

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathsf{X}\gamma \mid \gamma\mathsf{U}\gamma.$$

The more restricted syntax of state formulas in PCTL is defined as

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{P}_{\bowtie c}(\mathsf{X}\varphi) \mid \mathsf{P}_{\bowtie c}(\varphi\mathsf{U}\varphi).$$

The semantics of pCTL* are identical to CTL*, with the addition of the probability operator. Suppose we have a Markov chain $M = \langle S, P, V\rangle$, some state $s \in S$, a path formula γ, a rational number c between 0 and 1, and the set Paths^M that contains all paths through M. The semantics of the probability operator is then defined as

$$M, s \vDash \mathsf{P}_{\bowtie c}(\gamma) \qquad \text{iff} \quad \mu^s(\{\lambda \in \mathsf{Paths}^M \mid \lambda[1] = s \wedge M, \lambda \vDash \gamma\}) \bowtie c;$$

This reads as: the combined probability in state s of all paths λ that start in s and for which the path formula γ hold, is lower or higher than c. For PCTL, the semantics are similar, except for the additional restrictions given by the temporal operator. For example,

$$M, s \vDash \mathsf{P}_{\bowtie c}(\mathsf{X}\varphi) \qquad \text{iff} \quad \mu^s(\lambda \in \mathsf{Paths}^M \mid \lambda[1] = s \wedge M, \lambda[2] \vDash \varphi) \bowtie c.$$

The other new operators are defined in a similar fashion and are omitted here.

▶ **Example 8 (Robot on a grid, continued).** Let us look at some examples of pCTL* statements about our robot on a grid example. For the Markov chain M in Figure 2.2b and state s_0:

- $M, s_0 \vDash \mathsf{P}_{>0.4}(\mathsf{X}(\mathsf{atBottom} \wedge \mathsf{atLeft}))$ – starting in s_0, the probability that the robot is in the bottom-left square in the next step, is more than 0.4.
- $M, s_0 \vDash \mathsf{P}_{>0}(\mathsf{G}(\mathsf{atBottom} \wedge \mathsf{atLeft}))$ – starting in s_0, the probability that the robot is always in the bottom-left square, is more than 0.
- $M, s_0 \nvDash \mathsf{P}_{>0.4}(\mathsf{XXX}(\mathsf{atBottom} \wedge \mathsf{atLeft}))$ – starting in s_0, it is *not* true that the probability that the robot is in the bottom-left square after three steps, is more than 0.4.

As you may notice when checking whether the statements in Example 8 indeed hold, these statements are much more difficult to verify than the examples of previous logics. Especially statements with a high *depth* (the amount of future steps the statement describes), cause trouble. This is why the restriction that disallows the 'stacking' of temporal operators (and thus limits the depth of formulas), makes PCTL less complex to verify.

Formally, model checking pCTL* is polynomial time in the size of the system, but exponential time in the size of the formula. In comparison, PCTL can be model-checked in polynomial time in the size of the system, but in linear time in the size of the formula. In essence, this means that increasing the size of the Markov chains (through the number of states and/or transitions) impacts the efficiency of PCTL and pCTL* in the same way, but increasing the size of the formulas negatively impacts the computation time of pCTL* much more drastically than that of PCTL.

PCTL and pCTL* with Non-Determinism (PCTL$_{nd}$ and pCTL$^*_{nd}$)

We have now seen how PCTL and pCTL* can express properties about purely probabilistic models (i.e. Markov chains). However, what if we cannot express a probability for every transition in our system? This can happen when the probability of transitioning depends on a free choice of an agent, or because we have only partial knowledge about certain probabilities. Moreover, we might not always *want* to express all probabilities, in order to reason about temporal properties regardless of some probabilities. To accommodate for all of these scenarios, a model is required that lies in between a Markov chain and an MDP: for each state, there is a set of possible probability distributions (instead of a single one like in Markov chains), without actions being explicitly specified.

To accommodate this, [Bianco and Alfaro, 1995] introduce adapted syntax and semantics for PCTL and pCTL* that allow for model checking on MDPs. They do not explicitly name their adaptations, but to distinguish them from the purely probabilistic versions we refer to them as PCTL$_{nd}$ and pCTL$^*_{nd}$. Reintroduced are the path quantifiers A and E, and their semantics are defined alongside the probability operators as follows. Given a model $M = \langle S, P, V \rangle$, a state $s \in S$ and the set $\mathsf{Paths}^M = \{s_1 s_2 \ldots \mid s_i \in S \text{ and } P(s_i)(s_{i+1}) > 0 \text{ for all } 1 \geq i\}$, the (partial) semantics of pCTL$^*_{nd}$ are:

$$
\begin{aligned}
M, s \vDash \mathsf{A}\gamma \quad & \text{iff} \quad \text{for all } \lambda \in \mathsf{Paths}^M \text{ where } \lambda[1] = s, \text{ it holds that } M, \lambda \vDash \gamma; \\
M, s \vDash \mathsf{E}\gamma \quad & \text{iff} \quad \text{for some } \lambda \in \mathsf{Paths}^M \text{ where } \lambda[1] = s, \text{ it holds that } M, \lambda \vDash \gamma; \\
M, s \vDash \mathsf{P}_{<c}(\gamma) \quad & \text{iff} \quad \mu_s^+(\{\lambda \in \mathsf{Paths}^M \mid \lambda[1] = s \wedge M, \lambda \vDash \gamma\}) < c; \\
M, s \vDash \mathsf{P}_{>c}(\gamma) \quad & \text{iff} \quad \mu_s^-(\{\lambda \in \mathsf{Paths}^M \mid \lambda[1] = s \wedge M, \lambda \vDash \gamma\}) > c;
\end{aligned}
$$

where the last two definitions read as: "the most probable path from s for which γ holds has a probability lower than c" and "the least probable path from s for which γ holds has a probability greater than c" respectively. The temporal operators and Boolean connectives are defined as usual. PCTL$_{nd}$ is again the result of a restriction on the stacking of temporal operators.

Perhaps surprisingly, the impact of allowing non-determinism impacts the time complexity of model checking only for pCTL*. As shown by [Bianco and Alfaro, 1995], model checking PCTL$_{nd}$ formulas on Markov chains is still polynomial in the size of the model and linear in the size of the formula. For pCTL$^*_{nd}$, the time complexity of model checking in the size of the system is also unchanged, while the complexity in terms of the formula size increases from single exponential time to double exponential time.

Bounded PCTL and pCTL* (PCTL$_b$ and pCTL$^*_b$)

For all probabilistic branching-time logics discussed so far, it is possible to perform model checking with varying degrees of complexity. However, an open question remains: is it possible to decide whether there is *any* model that can satisfy a given formula? Neither the existence of an algorithm for deciding this, nor a classification of how complex this problem is compared to other logics, has been found for previously discussed probabilistic variants PCTL, pCTL*, PCTL$_{nd}$ and pCTL$^*_{nd}$.

In [Chodil et al., 2022], the current landscape of satisfiability results for fragments of PCTL and pCTL* is sketched. While there is not yet any proof that the general satisfiability problem is decidable, the *bounded* satisfiability problem has been shown to be decidable. This version answers the question: is there a model of maximum size X, that satisfies the formula? This means that if we can identify for a formula what the size of the smallest model should be, we can decide the general satisfiability problem as well.

This is where the bounded variants PCTL and pCTL* come into play (PCTL$_b$ and pCTL$^*_b$). These fragments introduce finite bounds to all operators, limiting the amount of steps to which they apply: e.g. $M, s \vDash \mathsf{EG}^k(\varphi)$ if and only if there exists a path λ in M from state s such that for all i where $0 \leq i < k$, $\lambda[i \dots k] \vDash \varphi$. For these fragments, [Chodil et al., 2022] proves that if there exists any model that satisfies a formula, then there also exists a model that satisfies it with a given finite bound on its size. Thus, the satisfiability problem of PCTL$_b$ and pCTL$^*_b$ is decidable.

Probabilistic Logic of Bounded Policies (PLBP)

The next logic we discuss, Probabilistic Logic of Bounded Policies (PLBP) by [Motamed et al., 2023], borrows elements from many logics and fragments discussed before. It is a probabilistic logic with probability operators like pCTL*, but like the adaptation pCTL$^*_{nd}$ it introduces non-determinism in the underlying model in the form of actions. Like pCTL$^*_b$, its operators have a finite bound, and going a step further even, the path formulas are evaluated on finite traces, just like in LTL$_f$.

The logic PLBP is defined relative to an infinite set of propositional variables Prop and a set of actions $\mathcal{A}$ (with preconditions and postconditions according to Definition 3) that is fixed throughout. The models, MDPs, that the formulas of the logic are evaluated against, are defined over these Prop and $\mathcal{A}$. The logic distinguishes between state formulas φ and path formulas γ^k, where the semantics of the latter are only defined with respect to paths of at least $k+1$ steps. Given propositional variable $p \in \mathsf{Prop}$, action $a \in \mathcal{A}$, integer $k \geq 1$, $c \in \mathbb{Q} \cap [0,1]$, and $\bowtie \in \{<, >, =\}$, the syntax of PLBP is defined inductively by the grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond^k_{\bowtie c}(\gamma^{k+1}),$$

$$\gamma^1 ::= \varphi$$

$$\gamma^{k+1} ::= \varphi \mid \mathsf{do}_a \mid \gamma^{k+1} \wedge \gamma^{k+1} \mid \neg\gamma^{k+1} \mid \mathsf{X}\gamma^k.$$

The probabilistic formula $\Diamond^k_{\bowtie c}(\gamma)$ informally means that the agent can act in such a way for the next k steps, that γ is true with a probability $\bowtie c$. There is also a universal dual to the $\Diamond$ operator: $\Box^k_{\bowtie c}(\gamma)$, expressing informally that for *any* behaviour of the agent in the next k steps, γ is true with a probability $\bowtie c$. The other newly introduced expression, do_a, asserts that the first action taken by the agent is action a.

To formally describe the semantics of PLBP, the concept of the agent being able to "act in such a way that ...", needs a more precise definition. This is done by introducing *k-step policies*: a specification of how an agent will act in the next k steps.

▶ **Definition 6 (k-step policy).** Given an MDP $M = \langle S, P, V \rangle$ defined over the action set $\mathcal{A}$, an integer $k \geq 1$, and a state $s \in S$, a *k-step policy* from s in M is a function $\pi : S^{\leq k}_s \to \mathcal{A}$ where $P(s_k, \pi(s_1 \dots s_k))\downarrow$ for all $s_1 \dots s_k \in S^{\leq k}_s$. To make the initial state s explicit, we usually write π^s instead of π. We denote the set of all k-step policies in M as $\mathsf{Pol}^{M,k}$.

▶ **Definition 7 (Path).** Given an MDP $M = \langle S, P, V \rangle$ defined over the action set $\mathcal{A}$, a *path* λ in M is an alternating sequence of states $s \in S$ and actions $a \in \mathcal{A}$ of the form $s_1 a_1 \dots s_k a_k s_{k+1}$ such that $P(s_i, a_i)\downarrow$ for all $1 \leq i \leq k$. We denote the set of all paths in M as Paths^M. The set of all paths from a k-step

policy π^s is

$$\mathsf{Paths}(\pi^s) = \{s_1 a_1 \dots s_k a_k s_{k+1} \in \mathsf{Paths}^M \mid s_1 = s \text{ and } \pi^s(s_1 s_2 \dots s_i) = a_i \text{ for all } 1 \leq i \leq k\}. \quad (2.3)$$

▶ **Definition 8 (Path distribution).** The *path distribution* of a k-step policy π^s in an MDP $M = \langle S, P, V \rangle$, is the probability distribution $\mu_\pi^{M,s}$ over $\mathsf{Paths}(\pi^s)$ defined as

$$\mu_\pi^{M,s}(s_1 a_1 \dots s_k a_k s_{k+1}) = \prod_{i=1}^{k} P(s_i, a_i)(s_{i+1}) \quad (2.4)$$

As standard, this extends to sets of paths X as

$$\mu_\pi^{M,s}(X) = \sum_{\lambda \in X} \mu_\pi^{M,s}(\lambda). \quad (2.5)$$

We drop the subscript π and superscript M, s from μ, whenever the variables are clear from context.

Now, given an MDP M and a state s in it, the semantics of the state formulas are defined as

$M, s \vDash p$	iff	$s \in V(p)$;
$M, s \vDash \neg\varphi$	iff	$M, s \nvDash \varphi$;
$M, s \vDash \varphi_1 \wedge \varphi_2$	iff	$M, s \vDash \varphi_1$ and $M, s \vDash \varphi_2$;
$M, s \vDash \Diamond^k_{\bowtie c}(\gamma)$	iff	there exists a k-step policy π^s such that $\mu^s_\pi(\{\lambda \in \mathsf{Paths}(\pi^s) \mid M, \lambda \vDash \gamma\}) \bowtie c$.

The semantics of the path formulas are defined with respect to a model M and a path λ, as

$M, \lambda \vDash \mathsf{do}_a$	iff	$a_1 = a$.
$M, \lambda \vDash \varphi$	iff	$M, s_1 \vDash \varphi$;
$M, \lambda \vDash \mathsf{do}_a$	iff	$a_1 = a$.
$M, \lambda \vDash \neg\gamma$	iff	$M, \lambda \nvDash \gamma$;
$M, \lambda \vDash \gamma_1 \wedge \gamma_2$	iff	$M, \lambda \vDash \gamma_1$ and $M, \lambda \vDash \gamma_2$;
$M, \lambda \vDash \mathsf{X}\gamma$	iff	$M, s_2 a_2 \dots s_{k+1} \vDash \gamma$;

▶ **Example 9 (Robot on a grid, continued).** To illustrate the expressivity of PLBP, consider the following example sentences, where M is the MDP modelling a robot on a 2 by 2 grid from Example 4, which has the state s_0 representing the bottom-left square.

- $M, s_0 \vDash \Box^2_{\geq \frac{1}{9}}(\mathsf{XX}(\mathsf{atBottom} \wedge \mathsf{atLeft}))$ – for all 2-step policies, the probability of being in the bottom-left square after two steps, is at least $\frac{1}{9}$.
- $M, s_0 \nvDash \Diamond^2_{> \frac{4}{9}}(\mathsf{XX}(\mathsf{atFlag})$ – it is not possible in M to have a 2-step policy that leads to the flag after two steps with a probability higher than $\frac{4}{9}$ (the optimal policy leads to the top-right square with a probability of only $\frac{2}{3} \cdot \frac{2}{3} = \frac{4}{9}$).

In conclusion, PLBP allows us to reason about probabilistic environments with agents that can strategise and act non-deterministically. Moreover, the logic has a decidable model checking problem (in PSPACE) and satisfiability problem (in 2EXPSPACE).

2.2.3 Rewards

As explained in Section 2.1.1, there are cases where we have information about rewards that the agent gains (or in case of negative rewards, costs that it incurs) in the environment. These rewards often play a central role in the tasks performed in such environments, e.g. in the case of reinforcement learning as further explained in Section 2.3. Therefore, we wish to reason about these rewards. For example, how much is accumulated with a particular path? Or what is the expected reward of particular policy? The following sections show three distinct approaches to the task (PRCTL [Andova et al., 2004], rPCTL [Bacci et al., 2021] and MTL [Jamroga, 2008]), which have varying expressivity and as a result different complexity results for the model checking and satisfiability problems.

PRCTL

The logic PRCTL, by [Andova et al., 2004] is an extention of PCTL with four new reward-related operators. The logic is evaluated on a Markov chain M and a reward function $\rho : S \to \mathbb{R}_{\geq 0}$ that maps each state to a reward.

The first addition to PCTL is the long-run average operator, $\mathcal{E}_J^k(\varphi)$, which informally means that in the next k steps, the expected reward rate for states in which φ holds lies in the interval J. $\mathcal{E}_J(\varphi)$ is the unbounded counterpart of this operator. $\mathcal{C}_J^n(\varphi)$ means that for all φ-satisfying states in the next k steps, each of the individual rewards is within the bounds of J. The final operator, $\mathcal{Y}_J^n(\varphi)$ means that the accumulated reward of all φ-satisfying states in the next k states lies in the interval J.

An algorithm for model checking PRCTL is also provided by [Andova et al., 2004], with the time complexity being exponential in the number of states of the Markov chain. As PRCTL is an extension of PCTL, the satisfiability problem for PCTL reduces to the satisfiability of PRCTL. Since the decidability of the satisfiability problem of PCTL remains unproven, the decidability of the satisfiability problem for PRCTL is by extension an open problem as well.

rPCTL

Another probabilistic temporal logic that incorporates rewards is rPCTL, by [Bacci et al., 2021]. Like PRCTL, it is an extension from PCTL, but the underlying models are different here: it works with *interval* Markov chains, in which each probabilistic transition is represented by an interval rather than a single rational number.

In rPCTL, the 'next' and 'until' operators are given bounds on the accumulated reward: $\mathsf{X}_{\trianglelefteq k}(\varphi)$ and $\varphi_1 \mathsf{U}_{\trianglelefteq k}(\varphi_2)$, where $\trianglelefteq \in \{\leq, =, \geq\}$. These are both path formulas, with the informal meaning that if $\lambda \vDash \mathsf{X}_{\trianglelefteq k}(\varphi)$ for a path λ, the next state is reached with a reward $\trianglelefteq k$, and that φ holds in that next state. For the until operator, the informal meaning is that there is some state s_i in the path for which φ_2 holds, all steps before that φ_1 holds, and the accumulated reward up until s_i is $\trianglelefteq k$.

In contrast to the reward operators from PRCTL, which reason about expected reward from a specific state onwards, the rewards in rPTCL are only reasoned about in the context of specific paths. Essentially, PRCTL can answer the question: what is the expected reward of all future states where φ holds? In contrast, rPCTL answers the question: with what probability do we take a path with these temporal properties and this bound on the accumulated reward? While rPCTL does have a more restricted expressivity, its relative simplicity does result in a more intuitive language for those used to working with temporal logics. Since rPCTL is again an extension of PCTL, the decidability of the satisfiability problem for rPCTL is also an open problem.

MTL

The last probabilistic temporal logic with rewards we discuss, MTL, was designed by [Jamroga, 2008]. It is quite distinct from many previously discussed logics, especially since it is a multi-valued logic for which sentences in the logic can not just be evaluated to be true or false, to an element of a subset of real numbers. This is usually the range $[0, 1]$, where the elements 0 and 1 are selected to represent the classical false ($\bot$) and true ($\top$), and the complement of each element c, denoted $\bar{c}$, is defined as $1 - c$.

The syntax for the state formulas φ and path formulas γ, can be inductively defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \oplus \varphi \mid \varphi \preccurlyeq \varphi \mid \mathsf{E}\gamma \mid \mathsf{M}\gamma,$$

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \gamma \oplus \gamma \mid \gamma \preccurlyeq \gamma \mid \mathsf{X}\gamma \mid \gamma \mathsf{U} \gamma \mid \mathsf{m}\gamma.$$

The traditional Boolean operators in the syntax have the following informal semantics: $\neg\varphi$ gives the complement of φ and $\varphi \wedge \psi$ gives the minimum value of φ and ψ. The two operators $\oplus$ and $\preccurlyeq$ have the following informal semantics: $\varphi \oplus \psi$ gives the average of φ and ψ, while $\varphi \preccurlyeq \psi$ is true (i.e. 1) when φ is smaller or equal to ψ and false (i.e. 0) otherwise. The path quantifiers $\mathsf{E}\gamma$ and $\mathsf{A}\gamma$ still have essentially the same meaning as for CTL* when γ is evaluated as true or false. When γ yields a numerical value however, E acts as a maximiser (there exists a future where this maximum value of γ is attainable), while A acts as a minimizer (for all futures γ is at least this value). The new quantifier $\mathsf{M}\gamma$ represents the middle ground of the two: the expected value of γ of all possible paths (the average weighed by probability of each particular path). The final addition is $\mathsf{m}\gamma$, which represents the average reward along

one particular path. A notable missing element, that was present in all previous probabilistic logics, is a way of reasoning about probabilities. Although the underlying model's transitions are probabilistic, is not possible to reason about probability in an explicit way in MTL.

The logic MTL_1, can be seen as a single-agent extension to MTL (or – to reflect the lack of agency – MTL_0) and introduces actions into the underlying model, similar to how PCTL_b extends PCTL.

The syntax of MTL_1 is given by:

$$\vartheta ::= p \mid \neg\vartheta \mid \vartheta \wedge \vartheta \mid \vartheta \oplus \vartheta \mid \vartheta \preccurlyeq \vartheta \mid \langle a \rangle \varphi,$$

$$\varphi ::= \vartheta \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \oplus \varphi \mid \varphi \preccurlyeq \varphi \mid \mathsf{E}\gamma \mid \mathsf{M}\gamma,$$

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \gamma \oplus \gamma \mid \gamma \preccurlyeq \gamma \mid \mathsf{X}\gamma \mid \gamma \mathsf{U} \gamma \mid \mathsf{m}\gamma.$$

where the newly introduced $\langle a \rangle \gamma$ means that agent a has a policy with which they can enforce φ. Policies in MTL_1 are similar to those in PLBP, but they are memoryless: each decision in MTL_1 depends only on the current state, whereas in PLBP the agent can decide based on the last k visited states.

The model checking problem and satisfiability problem for MTL_1 have both not yet been shown to be decidable.

2.3 Reinforcement Learning

As was already touched upon briefly in Chapter 1, reinforcement learning (RL) is an artificial intelligence technique used to learn the optimal way to behave in an environment by trial-and-error: repeatedly running experiments and learning from each outcome [Kaelbling et al., 1996]. Traditionally, RL algorithms learn by optimising the sum of rewards that are collected throughout an experiment. For example, take our earlier example environment of a robot on a 2 by 2 grid as drawn in Figure 2.1, that is tasked with reaching a flag. By awarding points for reaching the square with the flag, we can let an RL algorithm 'hone in' on behaviour that leads to reaching it.

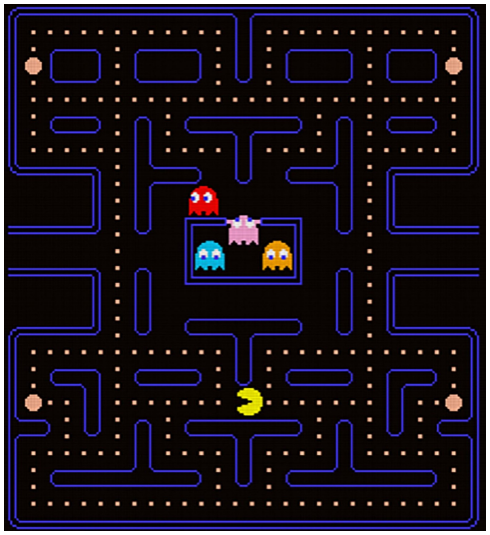
Usually, tasks involve more complex requirements or desiderata than simply reaching a target: e.g. avoiding certain harmful states. In typical RL methods, agents are trained to avoid dangerous situations by giving them negative rewards (i.e. costs). However, to learn that these unsafe states are undesirable, the agent still needs to experience them during training. This means that the training process can become expensive or harmful.

Take for example the training phase of a drone. Collision with another object can cause damage to the drone or the other object and must be avoided whenever possible. Unsafe states can then be defined as areas that are too close to a tree, or flight paths that cross the path of another drone. Instances involving physical agents like drones, which are susceptible to damage or even loss, are particularly evident in their need for enhanced safety measures during the training phase. However, it is important to note that RL methods for agents in virtual environments can also be adversely affected by unsafe behaviour during training.

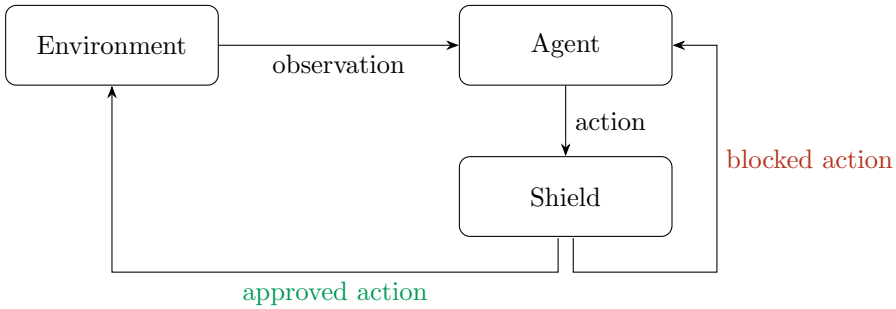
For instance, in a study by [Jansen et al., 2020], the PAC-MAN game (see Figure 2.3 for a screenshot of the game) is used as a motivating example for safe RL. In this game, the player is tasked with collecting small dots scattered throughout a maze to accumulate points. The objective is to collect all the dots, resulting in a victory. However, if the agent is captured by any of the ghosts patrolling the maze, the game ends and the player loses. During the initial iterations of the algorithm, the agent frequently falls prey to the ghosts, prompting numerous game restarts. This repetitive process negatively impacts the learning rate of the agent. Therefore, preventing unsafe behaviour during the training phase remains beneficial, despite the environment being virtual.

2.3.1 Shielded learning

Shielded learning [Alshiekh et al., 2018] (or simply *shielding*) is a method used during the training phase of RL to prevent unsafe behaviour by employing formal methods to verify the safety of chosen actions based on predefined requirements. Each action that is selected by the agent, is first assessed by the shield before execution. Actions deemed safe are forwarded to the environment, while unsafe actions are returned to the agent, who is tasked with choosing an alternative action. A diagram of this process is given in Figure 2.4.



▶ **Figure 2.3:** Screen capture of the PAC-MAN game, where the player (the yellow circle) should try to eat as many small yellow dots as possible, without being caught by one of the ghosts.

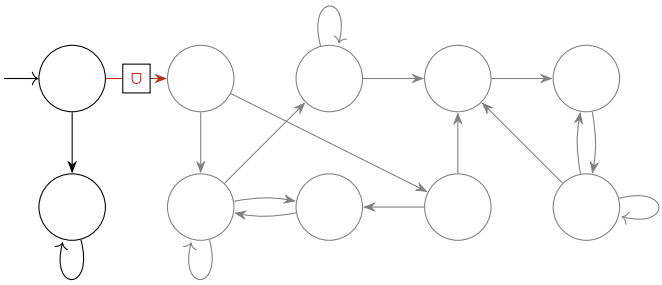


▶ **Figure 2.4:** Shielded reinforcement learning.

A distinction is made between *offline* shielding and *online* shielding [Könighofer et al., 2021]. In offline shielding, the impact of each state-action combination in the environment is evaluated in advance, and any combinations leading to unsafe states are blocked for the agent during the reinforcement learning (RL) algorithm's execution. On the other hand, online shielding verifies the safety of behaviour in real-time, considering only relevant actions at any given moment and using a finite horizon, such as 10 steps ahead, to reduce computational delays. In large and sparse MDPs (e.g. the example MDP in Figure 2.5), online shielding can significantly reduce the number of state-action pairs that are verified for safety, since unreachable states are never encountered nor checked.

The safety requirements are typically expressed using temporal logic like LTL or CTL, enabling restrictions such as "never be in a state where there is a ghost less than 6 steps away" in the context of a PAC-MAN environment. For some applications, such requirements are too restrictive on the ability of the agent to explore. This is where *probabilistic shielding* comes in [Jansen et al., 2020]. A probabilistic shield ensures that actions are only allowed whenever the probability of the safety requirements being satisfied is above some threshold $\delta \in [0, 1]$. The safety requirements can be written in the form $\mathsf{P}_{>\delta}(\gamma)$, where γ is an LTL formula. Thus, it can be considered to use a restricted form of pCTL* that does not allow for nested probability operators.

As discussed in Section 2.2.2, the logic PCTL cannot reason about particular policies or actions. Therefore, instead of determining the probability of satisfying the safety requirement exactly, this shield iterates over all possible policies and assesses whether the *maximal* probability is still below the set threshold. This means that only the safest future policy has to satisfy the constraint. An additional limitation of using PCTL (or LTL or CTL for that matter), is that it is not possible to reason about rewards in the safety specifications. This means we cannot express, for example, reward-contingent specifications that allow more risky behaviour in case a high reward is expected.



▶ **Figure 2.5:** Example MDP of a task where online shielding is less computationally demanding than offline shielding. The sparseness of the graph can render large parts of the state space unreachable, which is represented by being greyed out.

2.3.2 Desiderata for shielding specifications

Next, we discuss four characteristics of languages like the ones presented in Section 2.2 and argue why they are useful characteristics for a specification language for reward-contingent shielding. Subsequently, we use these characteristics to motivate our choice of extending PLBP with rewards.

Model checking

When using temporal logic to specify requirements for shielding, an important characteristic is whether the model checking problem of this logic is decidable. Decidability means that there exists an algorithm that can decide for any specification in that logic, whether a given model – e.g. an MDP – satisfies that specification or not. If model checking is undecidable for the logic we use to specify constraints, we cannot generally check if an agent's behaviour conforms to any rule, rendering them practically useless for the purpose of shielding. However, even in this case, there may be fragments of the logic that are still expressive enough to be used to specify RL constraints, and that do have a model checking algorithm.

The complexity of the model checking problems of a selection of the probabilistic temporal logics from Sections 2.2.2 and 2.2.3, is given in Table 2.1.

Logic	Complexity in size of the model	Complexity in size of the formula
PCTL	polynomial time	linear time
pCTL*	polynomial time	exponential time
PCTL_{nd}	polynomial time	linear time
pCTL^*_{nd}	polynomial time	double exponential time
PLBP	polynomial space	polynomial space
PRCTL	exponential time	exponential time
rPCTL	unknown	unknown
MTL	unkown	unkown

▶ **Table 2.1:** Model checking complexity of several probabilistic temporal logics.

Satisfiability

Satisfiability checking is answering the question: is there *any* model that satisfies this specification? While not as crucial to RL applications as the model checking problem, it can still be very valuable to have an algorithm that can decide satisfiability. For example, if the environment is partially or completely unknown, we can check whether there is any possible version of the world in which we can abide by our constraints (i.e. are the constraints *satisfiable*). This can help us to use the information embedded in our constraints, even when we still have incomplete information about the environment.

Another motivation to want a logic with a decidable satisfiability problem, is that this has the added benefit that the logical consequence problem is also decidable. This means we can answer the question: does a given finite set of specifications imply that a certain formula is always true? This comes down to

checking whether there exists a model/world in which the set of specifications is true while the formula is false (i.e. whether this combination is *satisfiable*). This can help us to check for (partially) unknown environments whether certain constraints are a logical consequence of our knowledge of the world (and thus true for all environments still possible).

As is the case for model checking, a logic for which the satisfiability problem lies in a lower complexity class is preferable to one in a higher complexity class. However, since many probabilistic temporal logics actually have an undecidable (or at least, not yet a proven decidable) satisfiability problem, this complexity class is generally of less interest than it is in the context of model checking. The decidability and complexity of the model checking problems of a selection of the probabilistic temporal logics from Sections 2.2.2 and 2.2.3, is given in Table 2.2.

Logic	Decidability	Complexity
PCTL	unknown	-
pCTL*	unknown	-
PCTL_{nd}	unknown	-
pCTL^*_{nd}	unknown	-
PCTL_b	proven	exponential time on a non-deterministic machine
pCTL^*_b	unknown	-
PLBP	proven	double exponential space
PRCTL	unknown	-
rPCTL	unknown	-
MTL	unknown	-

▶ **Table 2.2:** Complexity of the satisfiability problem for several probabilistic temporal logics.

(In)Finite Time

Most traditional temporal logics, like Linear Time Logic (LTL) and Computation Tree Logic (CTL), as well as their probabilistic counterparts, are evaluated over infinite traces. This means that if we are model checking the expression "p is always true" for example, then this must be true for any future state: there is no point at which we stop checking whether p is true. In contrast, a logic that is evaluated over finite traces, has only expressions that have some finite bound on the time for which they should hold. For example, in Probabilistic Logic for Bounded Policies (PLBP), temporal operators can only occur within the probability operator, which has a bound on the number of steps in which the formulas are evaluated.

Since finite time logics are more restricted than their infinite time counterparts, they usually have less complex (i.e. space and time-consuming) model checking and satisfiability problems. Therefore, in applications where finite time is sufficiently expressive, it is preferable. In the shielding applications described so far, finite traces are actually sufficient (e.g. [Sadigh et al., 2014, Hasanbeig et al., 2020]): for practical purposes, any individual run in a reinforcement learning algorithm is cut off after some set amount of steps. Therefore, it usually suffices that constraints are only evaluated on finite traces, bounded by that same amount of steps.

Moreover, even when disregarding the complexity argument, performing shielding only for up to a finite horizon can still be preferable. The MDPs that are used as models in reinforcement learning are often learned by observation, and the probabilities are approximations of the *real* probabilities. As noted by [Jansen et al., 2020], when determining the probability of events occurring in infinite time, any errors in these approximations accumulate and lead to incorrect results. This problem is less prevalent when looking ahead only a bounded amount of steps.

Agency

Probabilistic temporal logics can roughly be divided into three categories: logics that reason about agent-less processes, single-agent, or multi-agent processes. Agent-less processes are usually modelled with state transition systems or Markov Chains, depending on whether the probabilities of the transitions are known (or relevant to the application). There is no 'agency' or decision modelled that influences the transitions. Since RL is all about learning how actions influence the environment, these Markov Chains

are not used to model the environments that RL algorithms operate on. An example of a logic that uses Markov chains as models is PCTL. The effect is that we cannot use PCTL to reason about the effect of actions, let alone a multi-step plan or policy.

In contrast, single-agent systems are usually described by a Markov Decision Process (MDP). These model the environment by enumerating for each state a set of actions that each have their own resulting probability distribution over transitions. In more practical terms: given we take a certain action, how likely are we to transition to each of the other states in the environment? MDPs are therefore very suitable for modelling the environments of RL algorithms that attempt to optimise the behaviour of a single agent. Whether actions can be reasoned about explicitly differs between logics. For example, the non-deterministic variant of PCTL does not acknowledge the existence of actions, but only non-determinism: for each state there is a set of possible probability distributions. Which probability distribution is selected may depend on an agent performing an action, but this is not modelled explicitly. In contrast, PLBP does allow for expressions with explicit actions, such as: in our current state, taking action A will take us to a safe state with a 0.99 likeliness. This explicit kind of agency enables reasoning about policies as well, which makes it very suitable for shielding applications.

In multi-agent processes, the probability distribution of the transitions depends on the actions of all the agents together – either synchronous (all agents choose at the same time) or asynchronous (turn-wise). Analogously to how MDPs are suitable for modelling environments for single-agent RL problems, multi-agent MDPs are suitable for multi-agent RL problems. While applying probabilistic temporal logics to these more complex RL tasks is a valuable pursuit, the research in this thesis is limited to single-agent tasks.

Conclusion

To summarise the discussion above, the desired logical language for reward-contingent shielding is a logic that

i) can be used to reason about how rewards develop over time (i.e. a temporal logic with rewards);

ii) has operators to explicitly reason about the probability of an occurrence within a model;

iii) lets us reason explicitly about the actions of an agent and their effect;

iv) has a model checking problem of reasonable complexity.

As we have seen in the literature review, logics have been formulated before that are close to fitting this description. Both PRCTL and rPCTL facilitate reasoning about probabilities and rewards, but do not allow us to model or reason about actions, nor have proven decidable satisfiability. Similarly, MTL by [Jamroga, 2008] introduces reasoning about rewards in a probabilistic and non-deterministic system, but does not have operators to explicitly reason about probabilities, nor has decidable satisfiability. In contrast, PLBP by [Motamed et al., 2023] permits reasoning about probabilities in systems with non-determinism and has decidable model checking *and* satisfiability, but does not include vocabulary to reason about rewards.

In this thesis, we select PLBP as a basis for our intended language, due to its favourable model checking and satisfiability, and its ability to reason about the existence of bounded policies that have a particular probability of an event occurring. To make this a language suitable for reward-contingent shielding, we alter the underlying models of PLBP to include rewards and introduce new operators to reason about them. The design choices for this extension, rPLBP, are discussed in the upcoming chapter.

Chapter 3

Defining rPLBP

In this chapter, we propose the logic rPLBP as a specification language for reward-contingent shielding. First, we discuss the underlying model and define concepts such as paths and policies within it. Second, we introduce the new operators of the extension, along with their intended purpose. Third, we present the complete syntax and semantics of rPLBP.

3.1 Model

As is the case for PLBP, we define the underlying model for rPLBP relative to an infinite set of propositional variables Prop and an *action set* $\mathcal{A}$, according to Definition 3. For the underlying model, we take an MDP with rewards as defined in Definition 5. Within an MDP, we can refer to specific bounded policies or paths. We define these concepts below, largely following the definitions as given in [Motamed et al., 2023].

▶ **Definition 9 (k-step policy).** Given an MDP $M = \langle S, P, V, R\rangle$ defined over the actions $\mathcal{A}$, a $k \geq 1$, and a state $s \in S$, a *k-step policy* from s in M is a function $\pi : S_s^{\leq k} \to \mathcal{A}$ where $P(s_k, \pi(s_1 \dots s_k))\downarrow$ for all $s_1 \dots s_k \in S_s^{\leq k}$. To make the initial state s explicit, we usually write π^s instead of π. We denote the set of all k-step policies in M as $\mathsf{Pol}^{M,k}$.

▶ **Definition 10 (Path).** Given an MDP $M = \langle S, P, V, R\rangle$ defined over the actions $\mathcal{A}$, a *path* λ in M is an alternating sequence of states $s \in S$ and actions $a \in \mathcal{A}$ of the form $s_1 a_1 \dots s_k a_k s_{k+1}$ such that $P(s_i, a_i)\downarrow$ for all $1 \leq i \leq k$. We denote the set of all paths in M as Paths^M. The set of all paths from a k-step policy π^s is defined as

$$\mathsf{Paths}(\pi^s) = \{s_1 a_1 \dots s_k a_k s_{k+1} \in \mathsf{Paths}^M \mid s_1 = s \text{ and } \pi^s(s_1 s_2 \dots s_i) = a_i \text{ for all } 1 \leq i \leq k\}. \quad (3.1)$$

▶ **Definition 11 (Path distribution).** The *path distribution* of a k-step policy π^s in an MDP $M = \langle S, P, V, R\rangle$, is the probability distribution $\mu_\pi^{M,s}$ over $\mathsf{Paths}(\pi^s)$ defined as

$$\mu_\pi^{M,s}(s_1 a_1 \dots s_k a_k s_{k+1}) = \prod_{i=1}^{k} P(s_i, a_i)(s_{i+1}) \quad (3.2)$$

As standard, this extends to sets of paths X as

$$\mu_\pi^{M,s}(X) = \sum_{\lambda \in X} \mu_\pi^{M,s}(\lambda). \quad (3.3)$$

We drop the subscript π and superscript M, s from μ, whenever the variables are clear from context.

3.1.1 Reward measures

Given an MDP with rewards, which includes information about the rewards of individual transitions, we can also reason about the reward that is accumulated by the agent when following a particular path: the cumulative reward.

▶ **Definition 12 (Cumulative reward).** The *cumulative reward* of a path in an MDP $M = \langle S, V, P, R \rangle$ is given by a function $C^M : \mathsf{Paths}^M \to \mathbb{R}$. The cumulative reward function of a path $\lambda = s_1 a_1 \dots a_k s_{k+1}$ is

$$C^M(\lambda) = \sum_{i=1}^{k} R(s_i, a_i, s_{i+1}). \tag{3.4}$$

Moreover, since for a bounded policy, we are usually not certain which path will come from it, we can express what the expected accumulated reward is for this policy, by multiplying the reward of each possible path with its probability. We define both these concepts below and give an example.

▶ **Definition 13 (Expected reward).** Let $D^{M,k}$ be a set of tuples (π^s, l, u), where π^s is a k-step policy, and l and u are a lower and upper bound of at most k: $D^{M,k} = \{(\pi^s, l, u) \in \mathsf{Pol}^{M,k} \times \mathbb{Z} \times \mathbb{Z}) \mid 1 \leq l \leq u \leq k\}$. The function $E^{M,k} : D^{M,k} \to \mathbb{R}$, is the *expected reward* function of a policy π^s in MDP M between steps l and u. It is defined as

$$E^{M,k}(\pi^s, l, u) = \sum_{s_1 a_1 \dots s_k a_k s_{k+1} \in \mathsf{Paths}(\pi^s)} (\mu^s_\pi(s_1 a_1 \dots s_k a_k s_{k+1}) \cdot C^M(s_l a_l \dots s_u a_u s_{u+1})). \tag{3.5}$$

▶ **Example 10 (Robot on a grid, continued).** We turn to our example of a robot on a grid again, to show what our new definitions allow us to calculate about it. Using the MDP M from Example 5, we can formulate a 2-step policy π^{s_0} as

$$\begin{aligned} \pi^{s_0}(s_0) &= \mathsf{up}, \\ \pi^{s_0}(s_0 s_0) &= \mathsf{up}, \\ \pi^{s_0}(s_0 s_1) &= \mathsf{right}. \end{aligned} \tag{3.6}$$

This policy has the following elements λ in its set of paths $\mathsf{Paths}(\pi^s)$, with corresponding probabilities $\mu(\lambda)$:

$$\begin{aligned} &\lambda_1 = s_0\ \mathsf{up}\ s_1\ \mathsf{right}\ s_3, && \mu(\lambda_1) = \frac{9}{10} \cdot \frac{9}{10} = \frac{81}{100}, && C^M(\lambda_1) = 0 + 10 = 10; \\ &\lambda_2 = s_0\ \mathsf{up}\ s_1\ \mathsf{right}\ s_1, && \mu(\lambda_2) = \frac{9}{10} \cdot \frac{1}{10} = \frac{9}{100}, && C^M(\lambda_2) = 0; \\ &\lambda_3 = s_0\ \mathsf{up}\ s_0\ \mathsf{up}\ s_1, && \mu(\lambda_3) = \frac{1}{10} \cdot \frac{9}{10} = \frac{9}{100}, && C^M(\lambda_3) = 0; \\ &\lambda_4 = s_0\ \mathsf{up}\ s_0\ \mathsf{up}\ s_0, && \mu(\lambda_4) = \frac{1}{10} \cdot \frac{1}{10} = \frac{1}{100}, && C^M(\lambda_4) = 0. \end{aligned} \tag{3.7}$$

For each of these paths, we can also use the reward function R to determine the cumulative reward $C^M(\lambda)$, which, multiplied by the probabilities from Equation (3.7), gives us the expected reward $E^{M,2}(\pi^{s_0}, 1, 2)$:

$$\begin{aligned} E^{M,2}(\pi^{s_0}, 1, 2) &= \sum_{s_1 a_1 \dots s_k a_k s_{k+1} \in \mathsf{Paths}(\pi^s)} (\mu(s_1 a_1 \dots s_k a_k s_{k+1}) \cdot C^M(s_1 a_1 s_2 a_2 s_3)) \\ &= \sum_{\lambda \in \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}} (\mu(\lambda) \cdot C^M(\lambda)) \\ &= \mu(\lambda_1) \cdot C^M(\lambda_1) + \mu(\lambda_2) \cdot C^M(\lambda_2) + \mu(\lambda_3) \cdot C^M(\lambda_3) + \mu(\lambda_4) \cdot C^M(\lambda_4) \\ &= \frac{81}{100} \cdot 10 + \frac{9}{100} \cdot 0 + \frac{9}{100} \cdot 0 + \frac{1}{100} \cdot 0 = 8\frac{1}{10} \end{aligned} \tag{3.8}$$

3.2 Formulas

Now that we have defined the necessary concepts for the underlying model of rPLBP, we discuss how the syntax and semantics of PLBP formulas are extended to enable reasoning about policies and rewards. We conclude by presenting the complete syntax and semantics of rPBLP.

3.2.1 Policy formulas

The logic PLBP has only two types of formulas, state and path formulas, conventionally denoted by the variables φ and γ respectively. In the context of bounded policies, the only thing we can assert in

the language, is that there exists some bounded policy within a model, for which the probability of a path formula being satisfied is bounded by some value. This means PLBP has the limitation that only one probability restriction can be asserted per existential operator ($\Diamond$). It is therefore not possible, for example, to assert that there exists a k-step policy for which the probability for a path formula γ_1 is at most c_1 and the probability for a path formula γ_2 is at most c_2.

For rPLBP, we introduce an additional formula type: policy formulas, ξ. As their name suggests, policy formulas assert facts about a policy. More formally, $M, \pi^s \vDash \xi$ means that formula ξ holds for policy π^s in model M.

To be able to reason about the probability that a bounded policy produces a path that satisfies some path formula, we introduce a probability operator for policy formulas.

▶ **Formula 1 (Probability operator for policies).**

Syntax:	$\xi^k ::= \mathsf{P}_{\bowtie c}(\gamma^k)$.
Semantics:	$M, s \vDash \mathsf{P}_{\bowtie c}(\gamma)$ iff $\mu^s_\pi(\{\lambda \in \mathsf{Paths}(\pi^s) \mid M, \lambda \vDash \gamma\}) \bowtie c$(i.e. iff the probability of π^s producing a path for which γ holds is $\bowtie c$).
Example use:	$M, \pi^s \vDash \mathsf{P}_{\geq 0.4}(\mathsf{XX}p)$, meaning that for the policy π^s (which must be at least a 2-step policy), p holds in the third step with a probability of at least 0.4.

Besides this probability operator, a policy formula can be some reward measure formula (defined next in Section 3.2.2), a negation of a policy formula, or a conjunction of two policy formulas. We alter the existential operator $\Diamond^k_{\bowtie c}(\gamma)$ from PLBP to use these policy formulas explicitly.

▶ **Formula 2 (Existential bounded policy operator).**

Syntax:	$\varphi ::= \Diamond^k(\xi^k)$.
Semantics:	$M, s \vDash \Diamond^k(\xi)$ iff $\exists \pi^s$, s.t. $M, \pi^s \vDash \xi$ (i.e. iff there exists a k-step policy for which the policy formula ξ holds).
Example use:	$M, s \vDash \Diamond^2(\mathsf{P}_{\leq 0.4}(\mathsf{XX}p))$, meaning that there exists a 2-step policy π^s, for which p holds in the third step with a probability of at least 0.4.

3.2.2 Reward formulas

Now that we have a model with a reward function, we want to introduce a way to reason about those rewards. The simplest way to do so is by looking at the rewards gained along a specific path in the model, i.e. the *cumulative reward*. There are more properties to reason about than just the cumulative reward, however. We discuss three main reward measures as identified by [Forejt et al., 2011]: instantaneous, step-bounded cumulative, and target-bounded cumulative. The first looks at the reward at a specific future step, the second looks at the sum of all rewards received up to a specified step, and the third looks at the sum of all rewards received until to a specified target state is reached. In [Forejt et al., 2011] these measures are only introduced to reason about expected reward over multiple paths, but we adapt their definitions to refer to single paths instead.

Instantaneous reward

Introducing *instantaneous* reward measures allows us to isolate rewards earned at specific future steps. This can let us identify properties such as 'after 4 steps, the agent will receive a reward of at least 20'.

▶ **Formula 3 (Instantaneous reward of a path).**

Syntax:	$\gamma^k ::= \mathsf{I}^u_{\bowtie r}$, where $u \leq k$.
Semantics:	$M, s_1 a_1 \dots s_k a_k s_{k+1} \vDash \mathsf{I}^u_{\bowtie r}$ iff $R(s_u, a_u, s_{u+1}) \bowtie r$ (i.e. iff the reward received at the uth step of the path is $\bowtie r$).
Example use:	$M, \lambda \vDash \mathsf{XX}p \wedge \mathsf{I}^3_{\leq 4}$, meaning that for path λ (which must contain at least 3 actions), p holds in the third step and at least 4 "points" are received in step 3.

Extending PLBP with $\mathsf{I}^u_{\bowtie r}$ requires only a trivial adaptation of the PLBP model checking algorithm. Checking that $\mathsf{I}^u_{\bowtie r}$ holds for some path is similar in procedure and complexity to checking that $\mathrm{X}^u p$ holds for some path (where X^u represents repeating X u times). The only difference is checking whether $R(s_u, a_u, s_{u+1}) \bowtie r$ instead of whether $s_u \in \mathcal{V}(p)$.

Cumulative reward

Introducing a *cumulative reward* measure allows us to express that for a path λ, the combined rewards of all the steps up to and including step u are $\bowtie r$. We can do this by extending PLBP with the following operator.

▶ **Formula 4 (Step-bounded cumulative reward of a path).**

Syntax:	$\gamma^k ::= \mathsf{C}^u_{\bowtie r}$, where $u \leq k$.
Semantics:	$M, s_1 a_1 \dots s_k a_k s_{k+1} \vDash \mathsf{C}^u_{\bowtie r}$ iff $C(s_1 a_1 \dots a_u s_{u+1}) \bowtie r$ (i.e. iff the accumulated reward in the first u steps of the path is $\bowtie r$).
Example use:	$M, \lambda \vDash \mathsf{XX}p \wedge \mathsf{C}^3_{\leq 4}$, meaning that for path λ (which must contain at least 3 actions), p holds in the third step and at least 4 "points" are accumulated over 3 steps.

Checking that $\mathsf{C}^k_{\bowtie r}$ holds for a path requires more time and space than for the instantaneous reward operator $\mathsf{I}^k_{\bowtie r}$. Instead of only looking up once whether $R(s_u, a_u, s_{u+1}) \bowtie r$ for step u, we need to sum up $R(s_i, a_i, s_{i+1})$ for each step i up to and including step u. That said, it is not much more complex than model checking a bounded 'always' operator $\mathsf{G}^u p$, equivalent to $p \wedge \mathsf{X}p \wedge \wedge \mathsf{XX}p \wedge \dots \wedge \mathsf{X}^u p$.

Target-bounded cumulative reward

Now suppose we are interested in the cumulative reward earned up to a certain event, but we do not know exactly when this takes place. For example, in case the rewards modelled actually represent cost and we are specifically interested in the cost of reaching a state where p holds. If we now set a step-bound, like in Formula 4, but there is a way to complete this task in fewer steps, the cost of the irrelevant remainder of the time spent is also counted. To solve this, we can introduce the following target-bounded cumulative reward operator instead.

▶ **Formula 5 (Target-bounded cumulative reward of a path).**

Syntax:	$\gamma ::= \mathsf{F}^k_{\bowtie r}(\varphi)$, where φ is a state formula.
Semantics:	$M, s_1 a_1 \dots s_k a_k s_{k+1} \vDash \mathsf{F}^k_{\bowtie r}(\varphi)$ iff there exists a $j \leq k$ such that $C(s_1 a_1 \dots a_j s_{j+1}) \bowtie r$ and $M, s_{j+1} \vDash \varphi$ (i.e. iff the accumulated reward up to the state where φ holds, is $\bowtie r$).
Example use:	$M, s \vDash \Diamond^3_{>0.5}(\mathsf{F}^3_{\leq 4}(p))$, meaning there exists a 3-step policy such that with probability > 0.5, at least 4 "points" are accumulated until p holds (which is also within 3 steps).

It is possible to define both the instantaneous (I) and target-bounded cumulative (F) reward operators in terms of the step-bounded cumulative reward operator (C) and the existing PLBP operators. Consider the following equivalences:

$$\mathsf{I}^k_{\bowtie r} \quad \longleftrightarrow \quad \mathsf{X}^{k-1}\mathsf{C}^1_{\bowtie r}$$
$$\mathsf{F}^k_{\bowtie r}(\varphi) \quad \longleftrightarrow \quad ((\mathsf{X}\varphi \wedge \mathsf{C}^1_{\bowtie r}) \vee (\mathsf{XX}\varphi \wedge \mathsf{C}^2_{\bowtie r}) \vee \dots \vee (\mathsf{X}^k\varphi \wedge \mathsf{C}^k_{\bowtie r}))$$

We therefore only the step-bounded cumulative reward operator, Formula 4, in the syntax and semantics of rPLBP.

Expected reward

When considering the rewards that can be gained with a specific bounded policy, we have the problem that we are uncertain of which path will result from it. However, we do know precisely what the probability distribution over all the possible paths is. Therefore, we introduce an expected reward policy formula, which asserts a bound on the expected accumulated reward between some lower and upper bound.

▶ **Formula 6 (Expected step-bounded cumulative reward).**

Syntax:	$\varphi ::= \mathsf{E}^{l,u}_{\bowtie r}$, where $l \leq u \leq k$.
Semantics:	$M, \pi^s \vDash \mathsf{E}^{l,u}_{\bowtie r}$ iff $E^{M,k}(\pi^s, l, u) \bowtie r$.
Example use:	$M, \pi^s \vDash \mathsf{E}^{1,2}_{>3} \wedge \mathsf{P}_{<0.3}(\mathsf{XX}p)$, meaning that the 2-step policy pi^s has an expected reward in the first two steps of more than 3, *and* the probability of reaching a state where p holds is less than 0.3.

3.3 Syntax and semantics

We now present the syntax and semantics of rPLBP. Given an infinite set of Prop and an action set $\mathcal{A}$, the syntax of rPLBP is inductively defined by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond^k(\xi^k),$$

$$\xi^k ::= \mathsf{P}_{\bowtie c}(\gamma^k) \mid \neg\xi^k \mid \xi^k \wedge \xi^k \mid \mathsf{E}^{l,u}_{\bowtie r}$$

$$\gamma^0 ::= \varphi$$

$$\gamma^k ::= \varphi \mid \mathsf{do}_a \mid \gamma^k \wedge \gamma^k \mid \neg\gamma^k \mid \mathsf{X}\gamma^{k-1} \mid \mathsf{C}^u_{\bowtie r},$$

where $p \in \mathsf{Prop}$, $a \in \mathcal{A}$, $\bowtie = \{<, =, >\}$, $c \in [0,1] \cap \mathbb{Q}$, $r \in \mathbb{R}$, and $l, u, k \in \mathbb{Z}^+$ such that $1 \leq l \leq u \leq k$. We also define the shorthand $\Box^k(\xi) \equiv \neg\Diamond^k(\neg\xi)$, which has the informal meaning that "for all k-step policies, ξ holds."

The semantics of state formulas φ is defined with respect to an MDP $M = \langle S, P, V, R\rangle$ and a state s in M as

$M, s \vDash p$	iff	$s \in V(p)$;
$M, s \vDash \neg\varphi$	iff	$M, s \nvDash \varphi$;
$M, s \vDash \varphi_1 \wedge \varphi_2$	iff	$M, s \vDash \varphi_1$ and $M, s \vDash \varphi_2$;
$M, s \vDash \Diamond^k(\xi)$	iff	there exists a k-step policy π^s such that $M, \pi^s \vDash \xi$.

The semantics of the policy formulas ξ is defined with respect to an MDP $M = \langle S, P, V, R\rangle$ and a k-step policy π^s for M as

$M, \pi^s \vDash \mathsf{P}_{\bowtie c}(\gamma)$	iff	$\mu^s_\pi(\{\lambda \in \mathsf{Paths}(\pi^s) \mid M, \lambda \vDash \gamma\}) \bowtie c$;
$M, \pi^s \vDash \neg\xi$	iff	$M, \pi^s \nvDash \xi$;
$M, \pi^s \vDash \xi_1 \wedge \xi_2$	iff	$M, \pi^s \vDash \xi_1$ and $M, \pi^s \vDash \xi_2$;
$M, \pi^s \vDash \mathsf{E}^{l,u}_{\bowtie r}$	iff	$E^{M,k}(\pi^s, l, u) \bowtie r$.

The semantics of the path formulas γ is defined with respect to an MDP $M = \langle S, P, V, R\rangle$ and a finite path $\lambda = s_1 a_1 \ldots a_k s_{k+1}$ in M as

$M, \lambda \vDash \varphi$	iff	$M, s_1 \vDash \varphi$;
$M, \lambda \vDash \mathsf{do}_a$	iff	$a_1 = a$.
$M, \lambda \vDash \neg\gamma$	iff	$M, \lambda \nvDash \gamma$;
$M, \lambda \vDash \gamma_1 \wedge \gamma_2$	iff	$M, \lambda \vDash \gamma_1$ and $M, \lambda \vDash \gamma_2$;
$M, \lambda \vDash \mathsf{X}\gamma$	iff	$M, s_2 a_2 \ldots s_{k+1} \vDash \gamma$;
$M, \lambda \vDash \mathsf{C}^u_{\bowtie r}$	iff	$C^M(s_1 a_1 \ldots a_u s_{u+1}) \bowtie r$.

Now that we have defined the logic rPLBP formally, we turn to the problem of verifying the truth of a formula for a model: *model checking*.

Chapter 4

Model checking

In this section we investigate whether it is always possible to verify that a formula in rPLBP is true for a certain model, i.e.: model checking. For rPLBP, we define the model-checking problem as follows:

▶ **Definition 14 (Model checking rPLBP).** The *model checking problem for rPLBP* is the following decision problem.

Instance:	An MDP M over $\mathcal{A}$, with associated pre_a and Post_a for all $a \in \mathcal{A}$, a state s in M, and a state formula φ in rPLBP. Let all numbers in the model and formula be written in unary.
Question:	Does it hold that $M, s \vDash \varphi$?

▶ **Remark 1.** (Unary) We choose for the numbers to be written in unary to have a more intuitive relation between the numbers and their size. For an integer k written in unary, its size equals the number itself: $|k| = k$, e.g. $|111| = 111$. This means that if we require k space for an algorithm, the space complexity is linear with respect to the input k. In contrast, if we were to write the number k in binary, we would have that $|k| = \log_2(k)$, and an algorithm running in k space would then run in space exponential with respect to the input k. Not all numbers in the input (M and φ) are integers that can be translated directly into unary: the probabilities and rewards are rational numbers. All rationals are first written to share the same divisor, e.g. the set of rational numbers $\left\{\frac{c_1}{d_1}, \frac{c_2}{d_2}, \ldots \frac{c_n}{d_n}\right\}$, is written as

$$\left\{\frac{c_1 \cdot d_2 \cdot \ldots \cdot d_n}{d_1 \cdot d_2 \cdot \ldots \cdot d_n}, \frac{d_1 \cdot c_2 \cdot \ldots \cdot d_n}{d_1 \cdot d_2 \cdot \ldots \cdot d_n}, \ldots, \frac{d_1 \cdot d_2 \cdot \ldots \cdot d_{n-1} \cdot c_n}{d_1 \cdot d_2 \cdot \ldots \cdot d_{n-1} \cdot d_n}\right\}, \tag{4.1}$$

after which they are each written as the division of two unary numbers, e.g. the fraction $\frac{2}{3}$ can be written as $\frac{11}{111}$.

Similar to [Motamed et al., 2023], we provide two decision procedures for the model-checking problem. First, we show that this problem is decidable, by giving an algorithm that decides it in EXPSPACE. A second algorithm is then offered, which decides it in PSPACE. The first approach is included despite its higher complexity since it has the ability to provide proof of the existence of certain policies by providing a 'witness policy' that satisfies the requirements. This way, we can use this logic for policy synthesis: by asking the question "Does there exist a policy that reaches the intended goal with a good expected reward?", our algorithm can then provide such a policy if it exists.

4.1 Exponential-space algorithm

To decide the model checking question for a given model, state, and rPLBP formula, we determine for all the state formulas in φ – in order of increasing complexity – which states from M satisfy them. As an example, consider the formula $\varphi = \Diamond^2(\mathsf{P}_{>0}(\mathsf{XX}(p \wedge q)))$. Our set of state sub-formulas, presented in the order we would evaluate them in, is then $Subf(\varphi) = \{p, q, p \wedge q, \Diamond^2(\mathsf{P}_{>0}(\mathsf{XX}(p \wedge q)))\}$. The knowledge of which states satisfy p and q helps us decide which states satisfy $p \wedge q$, which in turn helps us determine

the next sub-formula. Since in the final step we have determined the set of states in which the main formula φ holds, we can check if s is in this set to deduce whether $M, s \vDash \varphi$. The corresponding algorithm is given in Algorithm 1, which uses the notation $[\varphi]_M = \{s \in S \mid M, s \vDash \varphi\}$.

Algorithm 1 Model checking state formula φ

```
1:  function MODEL-CHECK(M, s, φ)
2:      for φ' ∈ Subf(φ) do
3:          case φ' = p
4:              [φ']_M ← {t ∈ S | p ∈ V(t)}
5:          case φ' = ¬φ
6:              [φ']_M ← S \ [φ]_M
7:          case φ' = φ ∧ ψ
8:              [φ']_M ← [φ]_M ∩ [ψ]_M
9:          case φ' = ◇^k(ξ)
10:             [φ']_M ← {t ∈ S | ∃π^t ∈ Pol^{M,k} s.t. MODEL-CHECK-POLICY(M, π^t, ξ)}
11:     return s ∈ [φ]_M
```

For all policy formulas, the process for determining the set of states satisfying it follows directly from the semantics. In the case of a propositional variable, $[p]_M$ should contain the states of which the valuation includes p (lines 3-4). In the case of a negation, $[\neg\psi]_M$ should contain all states except those satisfying ψ (5-6). In case of a conjunction, $[\psi_1 \wedge \psi_2]_M$ should contain the states satisfying both ψ_1 and ψ_2 (lines 7-8). In the case of an existential operator, $[\Diamond^k(\xi)]_M$ should contain all states s in M for which a k-step policy π^s exists that satisfies ξ. Thus, we iterate over all states t and all k-step policies π^t, and add the state to $[\Diamond^k(\xi)]_M$ if the subroutine MODEL-CHECK-POLICY(M, π^t) returns **true** (lines 9-10). Note that if we would like to have an algorithm that returns witness policies – a policy that proves (sub-)formulas of the form $\Diamond^k(\xi)$ – we can add a variable that stores π^t in line 10.

4.1.1 Model checking policy formulas

Next, we zoom in on the process of model checking a policy formula: $M, \pi^s \vDash \xi$. These rPLBP policy formulas can contain a probabilistic operator over a path formula, an expected reward operator, or a Boolean combination of any policy formulas. Again, the process of model checking each of these follows intuitively from the semantics. We show a program for this in Algorithm 2.

Algorithm 2 Model checking policy formula ξ

```
1:  function MODEL-CHECK-POLICY(M, π^s, ξ)
2:      case ξ = ξ' ∧ ξ''
3:          return MODEL-CHECK-POLICY(M, π^s, ξ') and MODEL-CHECK-POLICY(M, π^s, ξ'')
4:      case ξ = ¬γ'
5:          return not MODEL-CHECK-POLICY(M, π^s, ξ')
6:      case ξ = P_{⋈c}(γ)
7:          c' ← 0
8:          for λ ∈ Paths(π^s) do
9:              if MODEL-CHECK-PATH(M, λ, γ) then
10:                 c' ← c' + μ_π^{M,s}(λ)
11:         return c' ⋈ c
12:     case ξ = E^{l,u}_{⋈r}
13:         r' ← 0
14:         for s_1 a_1 ... a_k s_{k+1} ∈ Paths(π^s) do
15:             r' ← r' + μ_π^{M,s}(s_1 a_1 ... a_k s_{k+1}) × C^M(s_l a_l ... a_u s_{u+1})
16:         return r' ⋈ r
```

In the case that ξ is a conjunction of two policy formulas, $\xi = \xi_1 \wedge \xi_2$, we check both $M, \pi^s \vDash \xi_1$ and $M, \pi^s \vDash \xi_2$ individually, and only when both hold, we know that $M, \pi^s \vDash \xi$ (lines 2-3). In the case that ξ is a negation of a policy formula, $\xi = \neg\xi'$, we check whether $M, \pi^s \vDash \xi'$ and only when this does not

hold, we know that $M, \pi^s \vDash \xi$ (lines 4-5). The case of a probabilistic operator, $\xi = \mathsf{P}_{\bowtie c}(\gamma)$ requires us to generate the set $\mathsf{Paths}(\pi^s)$ and sum the probability of each of those paths that satisfies γ (lines 6-10). If that sum $\bowtie c$, then $M, \pi^s \vDash \xi$ (line 11). The case of an expected reward operator, $\xi = \mathsf{E}^{l,u}_{\bowtie r}$, is similar. Again, we generate the set $\mathsf{Paths}(\pi^s)$, but now we sum for each path the multiplication of the probability and the cumulative reward in steps l up to and including u. If that sum $\bowtie r$, then $M, \pi^s \vDash \xi$ (lines 12-16).

4.1.2 Model checking path formulas

One last element to consider is that the model checking of the probability operator in a policy formula requires the model checking of path formulas: $M, \lambda \vDash \gamma$ or equivalently MODEL-CHECK-PATH(M, λ, γ) called in line 9 of Algorithm 2. An implementation of this subroutine is given in Algorithm 3.

Algorithm 3 Model checking path formulas

1: **function** MODEL-CHECK-PATH$(M, s_1 a_1 \dots a_{k-1} s_k, \gamma)$
2: **case** $\gamma = \varphi'$
3: **return** $s_1 \in [\varphi']_M$
4: **case** $\gamma = \mathsf{do}_a$
5: **return** $a_1 = a$
6: **case** $\gamma = \mathsf{C}^u_{\bowtie r}$
7: **return** $C^M(s_1 a_1 \dots a_{u-1} s_u) \bowtie r$
8: **case** $\gamma = \gamma' \wedge \gamma''$
9: **return** MODEL-CHECK-PATH$(M, s_1 \dots s_k, \gamma')$ **and** MODEL-CHECK-PATH$(M, s_1 \dots s_k, \gamma'')$
10: **case** $\gamma = \neg\gamma'$
11: **return not** (MODEL-CHECK-PATH$(M, s_1 \dots s_k, \gamma')$)
12: **case** $\gamma = \mathsf{X}\gamma'$
13: **return** MODEL-CHECK-PATH$(M, s_2 \dots s_k, \gamma')$

In case the path formula is a state formula φ', this sub-formula is already assigned a set of states in which they hold (since we evaluate state sub-formulas in order of increasing complexity). Thus, we can simply check whether the first state of our path is in this set to determine $M, \lambda \vDash \varphi'$ (lines 2-3). In case the path formula is do_a, we simply look up whether the first action of λ matches s (lines 4-5). In the case of the cumulative reward operator, $\gamma = \mathsf{C}^u_{\bowtie r}$, we check whether the sum of rewards in step 1 up to and including u, is $\bowtie r$ (lines 6-7). The satisfaction of Boolean connectives is determined by applying these connectives to recursive calls to Algorithm 3 (lines 8-11). In the final case where the path formula is $\mathsf{X}\gamma'$, we recursively call Algorithm 3 with as a path λ without the first state and action (lines 12-13).

4.1.3 Complexity

The decision procedure described above (combining Algorithms 1 to 3) shows that model checking rPLBP formulas is decidable. We now show that this decision procedure runs in EXPSPACE.

▶ **Theorem 1 (Inclusion in EXPSPACE).** *The model checking problem for rPLBP is decidable in EXPSPACE.*

Proof. To decide $M, s \vDash \varphi$, the non-recursive function MODEL-CHECK in Algorithm 1 is called once per model checking problem and iterates over the set of all subformulas of φ. For each of these iterations, the space used for determining the set of states satisfying the sub-formula can be reused. Since the first three cases are simple set operations (which require only linear space), the worst-case scenario is that a sub-formula is of the form $\Diamond^k(\xi)$.

The model checking of the formula $\Diamond^k(\xi)$ is done by iterating over all $t \in S$, and all k-step policies from that state t each time. Each k-step policy consists of a mapping from sequences of at most k states in S, $S^{\leq k}$[1], to actions from $\mathcal{A}$. The size of each policy grows linearly with respect to the size of $S^{\leq k}$ (if there are more states, more sequences can be made from them) and can thus be characterised as

[1] Technically, the mapping is only from the subset of $S^{\leq k}$ where for each s_i and s_{i+1} the probability of transitioning is not zero. However, since we are looking for an upper bound and the worst-case scenario is that the graph is fully connected, this detail is irrelevant here.

$\mathcal{O}(|S^{\leq k}|) = \mathcal{O}(|S|^k)$. We only need to consider one policy at a time which means that the space required for model checking the existential operator $\Diamond^k(\xi)$ is upper bounded by $\mathcal{O}(|S|^k)$, plus the space required for model checking one instance of $M, \pi^t \vDash \xi$.

Checking whether $M, \pi^s \vDash \xi$ is done by the recursive function MODEL-CHECK-POLICY in Algorithm 2. This recursive function has two general cases, $\xi' \wedge \xi''$ and $\neg\xi'$, and two base cases, $\mathsf{P}_{\bowtie c}(\gamma)$ and $\mathsf{E}^{l,u}_{\bowtie r}$. The space required for a recursive algorithm depends on the space required to keep track of the recursive 'trace' (the information necessary to return to and complete the previous recursive calls), plus the worst-case space required for the base cases.[2] The amount of recursive function calls in one branch of the recursion tree is bounded from above by $|\xi|$, since the size of the policy formula becomes strictly smaller in all recursive calls until a base case is inevitably reached. Since the arguments M and π^s remain unchanged in each recursive call, we only need to store the sub-formula ξ' each time. Thus, the space required to keep track of the recursive trace is at most polynomial.

Now let us consider the space that is required for handling the base cases $\mathsf{P}_{\bowtie c}(\gamma)$ and $\mathsf{E}^{l,u}_{\bowtie r}$. In both cases, the algorithm iterates over all paths from π^s, while keeping track of a sum. To decide $\mathsf{P}_{\bowtie c}(\gamma)$, the sum we keep track of eventually equals $\mu(\{\lambda \mid M, \lambda \vDash \gamma\})$. The function P in the input contains all probabilities for individual transitions, which, by Remark 1, are each written as a division of two unary integers and all share the same denominator. To get $\mu(\lambda)$ for a single path λ of length k, k such probabilities are multiplied. This means

$$|\mu(\lambda)| \leq |\frac{c^k}{d^k}| = \mathcal{O}(|c^k| + |d^k|) = \mathcal{O}(c^k + d^k) = \mathcal{O}(|P|^k), \tag{4.2}$$

where c is the largest numerator of all the probabilities, and d is the shared denominator. At most, $\mu(\{\lambda \mid M, \lambda \vDash \gamma\})$ is the result of summing $|\mathsf{Paths}(\pi^s)| = \mathcal{O}(|S|^k)$ such terms. Since $\mu(\lambda)$ shares the same denominator (d^k) for each λ, the size of the sum is bounded by $\mathcal{O}(|S|^k \times |P|^k)$. Thus, the space required to keep track of this sum is exponential in the size of the input.

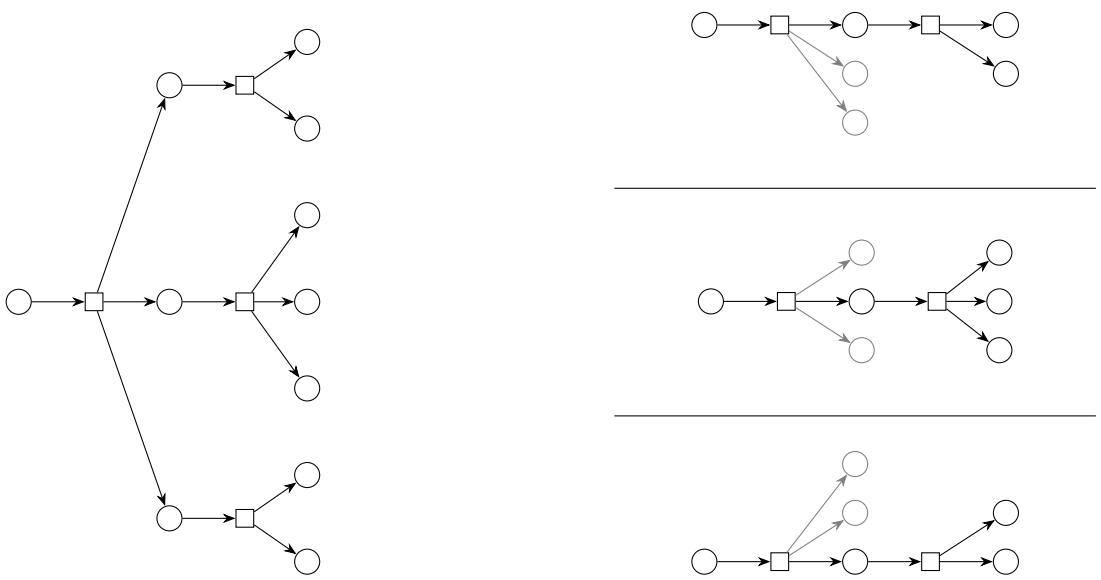
For each path measured for $\mathsf{P}_{\bowtie c}(\gamma)$, the function MODEL-CHECK-PATH from Algorithm 3 is called as well. This recursive function has three general cases, φ', do_a, and $\mathsf{C}^u_{\bowtie r}$, and three base cases, $\gamma' \wedge \gamma''$, $\neg\gamma'$, and $\mathsf{X}\gamma'$. The first two base cases, φ', do_a are simple look-ups, and require only constant space. The third base case, $\mathsf{C}^u_{\bowtie r}$, requires the summation of the rewards of at most k transitions. Since by Remark 1, each reward shares the same denominator d, the result of this summation has a size of $\mathcal{O}(k \times |R|)$. The general recursive cases add only a constant amount of overhead each time. Therefore, the worst-case space required for a single call to MODEL-CHECK-PATH is polynomial in the size of the input. The base case $\mathsf{P}_{\bowtie c}(\gamma)$, therefore, runs in space exponential in the size of the input.

For the base case $\mathsf{E}^{l,u}_{\bowtie r}$, the algorithm iterates over all paths from π^s, again keeping track of a sum written in unary: the expected reward this time. For each $\lambda = s_1 a_1 \dots s_k a_k s_{k+1} \in \mathsf{Paths}(\pi^s)$, the term $\mu(\lambda)$ is multiplied with $C^M(s_l a_l \dots s_u a_u s_{u+1})$. The former has the denominator d^k, while the latter has the denominator d. Thus, the multiplication always results in a term with denominator d^{k+1} and a nominator of size $\mathcal{O}(|P|^k \times k \times |R|)$, where r is the largest nominator of the rewards. Then, at most $|\mathsf{Paths}(\pi^s)| = \mathcal{O}(|S|^k)$ of such terms with a shared denominator are summed, which requires space exponential in the size of the input.

To summarise, the recursive function MODEL-CHECK-POLICY in Algorithm 2 requires $\mathcal{O}(|\varphi|^2)$ space to keep track of the recursive trace, and $\mathcal{O}(|M|^k)$ space for the base cases. Since the space required for the MODEL-CHECK function in Algorithm 1 is upper bounded by the space required for calling MODEL-CHECK-POLICY once, we have that the model checking problem of rPLBP can be decided in EXPSPACE. □

As noted earlier, it is possible to adapt this algorithm to one which returns policies that are proof of the truth of (sub-)formulas of the form $\Diamond^k(\xi)$. If we want to have such witness policies, we can not optimise the space complexity of the algorithm any further: if we must produce policies of size $\mathcal{O}(|S|^k)$ as a part of our result, then the space required is always exponential in the size of the formula. The same observation was made by [Motamed et al., 2023] for model checking PLBP.

[2] For a more detailed account of the space complexity of recursive functions, see [Goodrich et al., 2014].



(a) A full 2-step policy.

(b) The same example 2-step policy, if only one path is evaluated at a time.

▶ **Figure 4.1:** Illustration of how the required space can be reduced if the full policy does not need to be stored. The square nodes represent actions and the circle nodes represent states, the outcomes of actions.

4.2 Polynomial-space algorithm

If we do not need to know which policies were found to satisfy the policy formulas, we can further optimise the algorithm to use less space. In [Motamed et al., 2023], this is done by introducing a sub-routine PLBP-MEASURE (see Algorithm 4, which is called once for each $s \in S$ whenever $\Diamond^k_{\bowtie c}(\gamma)$ occurs in a PLBP sub-formula. This subroutine non-deterministically generates a policy by guessing actions and measures the probability of γ for that policy. By the nature of a non-deterministic algorithm, if in any possible run of this algorithm (e.g. for any policy), the outcome is a P_γ that is $\bowtie c$, then our main algorithm will decide that $\mathsf{P}_{\bowtie c}(\gamma)$ holds.

The subroutine is structured in such a way that we never have to remember the full policy, but only ever need to represent one branch of the random policy at a time (see Figure 4.1 for an illustration of how this saves space). Using this non-deterministic sub-routine, the model checking problem of PLBP can be decided in NPSPACE. By Savitch's theorem [Savitch, 1970], any problem that can be decided on a non-deterministic Turing machine in $\mathcal{O}(f(n))$ space, can also be decided on a deterministic Turing machine in $\mathcal{O}(f(n)^2)$ space. Since $f(n)$ is a polynomial function in our case, $f(n)^2$ is also a polynomial function. Therefore, the model-checking problem for PLBP can be decided in PSPACE [Motamed et al., 2023].

Algorithm 4 Measuring a single value on a non-deterministically generated policy

```
1: function PLBP-MEASURE(λ = s_1 a_1 ... a_{n-1} s_n, γ, k, P_γ)
2:     if n > k then
3:         if λ ⊨ γ then P_γ ← P_γ + μ(λ)
           return P_γ
4:     else
5:         guess a ∈ actions(s_n)
6:         for o ∈ outcomes(s_n, a) do
7:             P_γ ← MEASURE(s_1 a_1 ... a_{n-1} s_n a o, γ, k, P_γ)
           return P_γ
```

We cannot mimic this approach of performing measurements in a non-deterministic sub-routine directly, since we have introduced the notion of policy formulas to rPLBP. This means that for the formula

$\Diamond^k(\xi)$, the policy formula ξ may contain multiple formulas that we need to check for the same policy. For example, consider the formula $\Diamond^3(\mathsf{P}_{>0}(\mathsf{X}\varphi) \wedge \mathsf{E}^{1,2}_{>2})$. The first part can be verified with the measure function in Algorithm 4: generate a random policy and compute the probability that a path from this policy satisfies $\mathsf{X}\varphi$. However, once we move to the second part of the formula and we want to verify that the expected reward for this policy is more than 2, we can not reconstruct the same policy as before, since we have not stored it. Therefore, we are only able to say that there exists a policy such that $\mathsf{P}_{>0}(\mathsf{X}\varphi)$ and a policy such that $\mathsf{E}^{1,2}_{>2}$, but we can not verify with this method that there exists one policy that satisfies both.

To solve this issue, we devise a way to measure multiple requirements from a policy formula in parallel. According to the syntax for policy formulas,

$$\xi^k ::= \mathsf{P}_{\bowtie c}(\gamma^{k+1}) \mid \neg\xi^k \mid \xi^k \wedge \xi^k \mid \mathsf{E}^{l,u}_{\bowtie r},$$

they can contain a probability operator, an expected reward operator, a conjunction of two policy formulas, or a negation of a policy formula. By a single pass of ξ, we collect the set of sub-formulas of the form $\mathsf{P}_{\bowtie c}(\gamma)$ and $\mathsf{E}^{l,u}_{\bowtie r}$, i.e.: all properties we wish to measure in parallel. Note that the path formulas γ may contain state formulas that contain additional policy operators. We do not collect those, since all state formulas that are subformulas of ξ are already resolved at this point.

A new version of MODEL-CHECK is given in Algorithm 5. This new procedure still iterates over all states $t \in S$, but – instead of iterating over all policies from those states – calls a non-deterministic sub-routine MEASURE once per state (line 11). This sub-routine has roughly the same function as PLBP-MEASURE, but can measure multiple values in parallel. Once these measurements are made, another subroutine MODEL-CHECK-MEASUREMENTS is called, which returns true if the measurements are those of a policy satisfying ξ, and false if they are of a policy not satisfying ξ. In case the subroutine returns true, the state t is added to the set $[\Diamond^k(\xi)]_M$.

Algorithm 5 Non-deterministic algorithm for model checking state formula φ

```
1:  function MODEL-CHECK(M, s, φ)
2:      for φ' ∈ Subf(φ) do
3:          [φ']_M ← ∅
4:          ...
5:          case φ' = ◇^k(ξ)
6:              Ψ ← ∅
7:              for ξ' ∈ Subf(ξ) do
8:                  if ξ' = P_⋈c(γ) or ξ' = E^{l,u}_⋈r then
9:                      add (ξ', 0) to Ψ
10:             for t ∈ S do
11:                 Ψ' ← MEASURE(M, k, t, Ψ)
12:                 if MODEL-CHECK-MEASUREMENTS(M, Ψ', ξ) then
13:                     add t to [φ']_M
14:     return s ∈ [φ]_M
```

The pseudo-code for the sub-routine MEASURE is given in Algorithm 6. Instead of taking as input a variable P_γ and a path formula γ, it takes as input a set of pairs (ξ', v). The paths of a policy are randomly generated (lines 9-13), and at each endpoint of a path (when the length of the current path has reached k), the v values are updated according to the semantics of ξ'. For $\xi' = \mathsf{P}_{\bowtie c}(\gamma)$, the probability of the current path is added to the sum v if the path satisfies γ (lines 4-5). For $\xi' = \mathsf{E}^{l,u}_{\bowtie r}$, the probability of the current path is multiplied with the cumulative reward of the path from step l to u and then added to the sum v (lines 6-7).

The subroutine MODEL-CHECK-MEASUREMENTS is given in Algorithm 7. It is an adaptation of MODEL-CHECK-POLICY from Algorithm 2, which takes as input a set of measurements Ψ over a particular policy, instead of the policy itself. When encountering a $\mathsf{P}_{\bowtie c}(\gamma)$ or $\mathsf{E}^{l,u}_{\bowtie r}$ term, instead of performing the measurements on the spot, this subroutine checks the inequalities against the values stored in Ψ (lines 7 and 10). Thus, only when the measurements Ψ are of a policy satisfying ξ, does this subroutine return true.

Algorithm 6 Non-deterministic sub-routine for computing multiple measures in parallel

function MEASURE($M, k, \lambda = s_1 a_1 \dots a_{n-1} s_n, \Psi\}$)
  if $n > k$ **then**
    for $(\xi, v) \in \Psi$ **do**
      if $\xi = \mathsf{P}_{\bowtie c}(\gamma)$ **and** MODEL-CHECK-PATH(M, λ, γ) **then**
        $v \leftarrow v + \mu(\lambda)$
      else if $\xi = \mathsf{E}^{l,u}_{\bowtie r}$ **then**
        $v \leftarrow v + \mu(\lambda) * C^M(s_l a_l \dots a_u s_{u+1})$
    return Ψ
  else
    guess $a \in actions(s_n)$
    for $o \in outcomes(s_n, a)$ **do**
      $\Psi \leftarrow$ MEASURE($M, k, s_1 a_1 \dots a_{n-1} s_n\ a\ o, \Psi$)
    return Ψ

Algorithm 7 Model checking policy formula ξ for measurements Ψ

function MODEL-CHECK-MEASUREMENTS(M, Ψ, ξ)
  case $\xi = \xi' \wedge \xi''$
    return MODEL-CHECK-MEASUREMENTS(M, Ψ, ξ') **and** MODEL-CHECK-MEASUREMENTS(M, Ψ, ξ'')
  case $\xi = \neg\gamma'$
    return not MODEL-CHECK-MEASUREMENTS(M, Ψ, ξ')
  case $\xi = \mathsf{P}_{\bowtie c}(\gamma)$
    look up $(\mathsf{P}_{\bowtie c}(\gamma), v)$ in Ψ
    return $v \bowtie c$
  case $\xi = \mathsf{E}^{l,u}_{\bowtie r}$
    look up $(\mathsf{E}^{l,u}_{\bowtie r}, v)$ in Ψ
    return $v \bowtie r$

4.2.1 Complexity

▶ **Lemma 2 (Inclusion in PSPACE).** *The model-checking problem for rPLBP is decidable in PSPACE.*

Proof. To decide $M, s \vDash \varphi$, the non-recursive function MODEL-CHECK in Algorithm 5 is called once per model checking problem and iterates over the set of all subformulas of φ. For each of these iterations, the space used for determining the set of states satisfying the sub-formula can be reused. The worst-case scenario in terms of required space is when a sub-formula is of the form $\Diamond^k(\xi)$.

The model checking of the formula $\Diamond^k(\xi)$ is done by (1) collecting the terms that require measurement, and (2) iterating over all $t \in S$ while calling the subroutines MEASURE and MODEL-CHECK-MEASUREMENTS each time. Note that for this second part, the space can be reused after every iteration.

The subroutine MEASURE is a non-deterministic recursive function. In the base case, the values stored in Ψ are updated. As we have seen in the proof of Theorem 1, when stored in unary, those values each grow exponential in the size of the input. However, since we would like to show that our model-checking algorithm runs in PSPACE, we consider a more space-efficient alternative: writing the sum in binary. The size of a number k written in binary has a size of $\log_2(k)$. Rationals can also be written as the division of two binary integers, e.g. the fraction $\frac{2}{3}$ is written as $\frac{10}{11}$. The size of a binary rational $q = \frac{c}{d}$ then equals $\log_2(c) + \log_2(d)$.

As shown in the proof of Theorem 1, $\mu(\{\lambda \,|\, M, \lambda \vDash \gamma\})$ has a size bounded by $\mathcal{O}(|S|^k \times |P|^k)$ when written in unary. In contrast, we have that in binary, the sum is bounded by

$$\mathcal{O}(\log_2(|S|^k \times |P|^k)) = \mathcal{O}(k \times \log_2(|S| \times |P|)) = \mathcal{O}(k \times \log_2(|M|)). \tag{4.3}$$

The same holds for the variables in Ψ that store expected rewards. In unary, the size of such values is bounded by $O(|S|^k \times |P|^k \times k \times |R|)$. In binary, the size is bounded by

$$O(\log_2(|S|^k \times |P|^k \times k \times |R|)) = O(k \times \log_2(|M|)). \tag{4.4}$$

Thus, all values in Ψ use space polynomial in the size of the input. Moreover, the function call MODEL-CHECK-PATH that is made in the base case, still requires only polynomial space.

In the recursive case, a new action is guessed and for all possible outcomes the function calls itself with the original path appended with the guessed action and outcome. Since at each recursive call the length of the path λ grows by one until its length exceeds k, the depth of the recursion is bounded by k. At each recursive call, the outcomes that have not been considered yet need to be stored. The size of this is bounded by $|S|$, so the space required to keep track of the recursive trace at any point in time is $O(k \times |S|)$. This means that the recursive function MEASURE runs in polynomial space on a non-deterministic machine.

Now as for the subroutine MODEL-CHECK-MEASUREMENTS, the size of ξ decreases in each recursive call until either a $\mathsf{P}_{\bowtie c}(\gamma)$ or $\mathsf{E}^{l,u}_{\bowtie r}$ term is reached. The base cases are simple lookup and comparison actions, and do not require more space than Ψ. The recursive cases again give negligible overhead. Thus, we can conclude that the space required to run the MODEL-CHECK-MEASUREMENTS subroutine is dominated by the size of Ψ, which is polynomial.

To summarise, the space required to decide $M, s \vDash \varphi$ is dominated by the space required to model check subformulas of the form $\Diamond^k(\xi)$. This can be done in polynomial space on a non-deterministic machine, which means the model-checking problem for rPLBP can be decided in NPSPACE. By Savitch's theorem [Savitch, 1970], any problem that can be decided on a non-deterministic Turing machine in $\mathcal{O}(f(n))$ space, can also be decided on a deterministic Turing machine in $\mathcal{O}(f(n)^2)$ space. Thus, the model-checking problem for rPLBP is also in PSPACE. □

▶ **Theorem 3 (PSPACE-completeness).** *The model-checking problem for rPLBP is PSPACE-complete.*

Proof. To show our model-checking problem is PSPACE-hard, we take the model-checking problem for PLBP – proven in [Motamed et al., 2023] to be PSPACE-complete – and reduce it to the model-checking problem for rPLBP. Note that while the models used in rPLBP are MDPs *with* reward functions: $M = \langle S, P, V, R\rangle$, PLBP uses MDPs without reward functions: $M' = \langle S, P, V\rangle$.

We take an MDP $M\langle S, P, V\rangle$ defined over action set $\mathcal{A}$, a state $s \in S$, and a PLBP state formula φ_{PLBP}. We define a reward function R such that $R(s, a, s') = 0$ for all $s, s' \in S$ and $a \in \mathcal{A}$. Any other reward assignment would suffice as well, since none of the semantics in PLBP refer to this reward structure. Next, we define a recursive translation function Φ which maps PLBP formulas to rPLBP formulas as

$$\begin{aligned}
\Phi(p) &= p;\\
\Phi(\psi \wedge \psi') &= \Phi(\psi) \wedge \Phi(\psi');\\
\Phi(\neg\psi) &= \neg\Phi(\psi);\\
\Phi(\Diamond^k_{\bowtie c}(\gamma)) &= \Diamond^k(\mathsf{P}_{\bowtie c}(\Phi(\gamma)));\\
\Phi(\mathsf{X}\psi) &= \mathsf{X}\Phi(\psi);\\
\Phi(\mathsf{do}_a) &= \mathsf{do}_a;
\end{aligned} \tag{4.5}$$

All terms remain unchanged (and so do their semantics), except for the $\Diamond^k_{\bowtie c}(\gamma)$ terms. This operator does not exist in rPLBP, but by nesting the rPLBP probability operator $\mathsf{P}_{\bowtie c}(\gamma)$ in the existential operator $\Diamond^k$, we have that the semantics are exactly the same as for the original PLBP formula: "there exists a k-step policy π^s such that $\mu^s_\pi(\{\lambda \in \mathsf{Paths}(\pi^s) \mid M, \lambda \vDash \gamma\}) \bowtie c$". Thus, we have that $\langle S, P, V\rangle, s \vDash \varphi_{\mathsf{PLBP}}$ in holds in PLBP, iff $\langle S, P, V, R\rangle, s \vDash \Phi(\varphi_{\mathsf{PLBP}})$ holds in rPLBP.

By succesfully reducing a PSPACE-complete problem to the model-checking problem for rPLBP, we have shown that it is PSPACE-hard. Since by Lemma 2, we also have that the model-checking problem for rPLBP is in PSPACE, we can conclude that the model-checking problem for rPLBP is PSPACE-complete. □

Chapter 5

Satisfiability

In this section, we show that it is possible to determine whether there exists an MDP with a state that satisfies a given rPLBP formula. We define the satisfiability problem as follows.

▶ **Definition 15 (Satisfiability).** The *satisfiability problem for rPLBP* is the following decision problem.

Instance:	A state formula φ in rPLBP.
Question:	Is there an MDP $M = \langle S, P, V, R \rangle$ with a state $s \in S$ such that $M, s \vDash \varphi$?

We largely follow the proof of decidability of the satisfiability problem for PLBP by [Motamed et al., 2023], with adaptations to accommodate for the new features of our extension (rewards and policy formulas).

The outline of the proof we present is as follows. We first show that if there exists any model that satisfies a given rPLBP formula, then there is also one with no more states than some upper bound. This gives us the certainty that we never have to consider any models that are larger than this bound and can exhaustively iterate over all sets of states of this size and all possible valuations of these states. We can then for each of these options reduce the question of whether there exists a probability distribution function P and a reward structure R such that the formula is satisfied, to a decidable formula in the theory of real closed fields.

This allows us to decide whether there exists any model that satisfies the given formula, giving a decision procedure for satisfiability. Lastly, we show that this decision procedure for the satisfiability problem for rPLBP runs in 2EXPSPACE, the same complexity class as the satisfiability problem for PLBP is in.

5.1 Finite model property

First, we show that if there exists any model that satisfies the formula, then there is also one with a bounded size. To aid in this proof, we define the process of 'unravelling' an MDP.

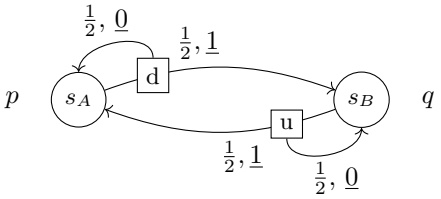
5.1.1 Unravelled MDP

▶ **Definition 16 (Unravelled MDP).** Given an MDP $M = \langle S, P, V, R \rangle$ that is defined over some set of actions $\mathcal{A}$ and propositions Prop and state $s \in S$, we define the MDP $M_s^{\mathsf{unrav}} = \langle S', P', V', R' \rangle$ as follows. The set of states contains all possible paths from s in M: $S' = \{s_1 a_2 s_2 \dots a_{k-1} s_k \mid s_1 = s, k \geq 1, s_i \in S$ for all $i \leq k\}$. We denote the last state of each path as $\mathsf{last}(s_1 a_2 s_2 \dots a_{k-1} s_k) = s_k$. We let $P'(\mathbf{h}, a)\downarrow$ iff $P(\mathsf{last}(\mathbf{h}), a)\downarrow$ for all $\mathbf{h} \in S'$ and $a \in \mathcal{A}$. Then, whenever $P'(\mathbf{h}, a)\downarrow$, we define the function P' as follows.

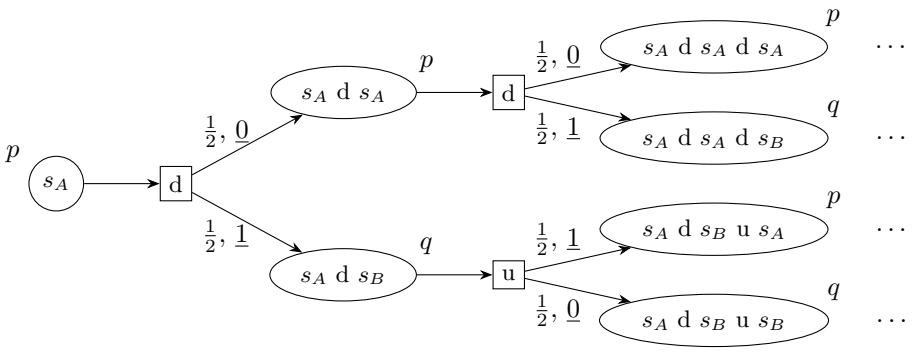
$$P'(\mathbf{h}, a)(\mathbf{h}') = \begin{cases} P(\mathsf{last}(\mathbf{h}), a)(s') & \text{for} \quad \mathbf{h}' = \mathbf{h}as' \\ 0 & \text{for} \quad \mathbf{h}' \neq \mathbf{h}as' \end{cases} \tag{5.1}$$

Moreover, we define the reward function R' in a similar fashion.

$$R'(\mathbf{h}, a, \mathbf{h}') = \begin{cases} R(\mathsf{last}(\mathbf{h}), a, s') & \text{for} \quad \mathbf{h}' = \mathbf{h}as' \\ 0 & \text{for} \quad \mathbf{h}' \neq \mathbf{h}as' \end{cases} \tag{5.2}$$



▶ **Figure 5.1:** Visualisation of MDP M. Circular nodes are states, square nodes are actions, underlined numbers are rewards, and valuations of the states are written in italics next to each state.



▶ **Figure 5.2:** The unravelled MDP $M^{\mathsf{unrav}}_{s_A}$. Circular nodes are states, square nodes are actions, underlined numbers are rewards, and valuations of the states are written in italics next to each state. The dots on the right-hand side denote that there are more states and transitions than can be explicitly represented.

Lastly, we define the new valuation function as $V'(\mathbf{h}) = V(\mathsf{last}(\mathbf{h}))$, which concludes our definition of M^{unrav}_s.

▶ **Example 11.** Suppose we have a set of propositional variables $\mathsf{Prop} = \{p, q\}$ and a set of actions $\mathcal{A} = \{\mathsf{d}, \mathsf{u}\}$, where $\mathsf{pre}_{\mathsf{d}} = p \wedge \neg q$, $\mathsf{pre}_{\mathsf{u}} = \neg p \wedge q$, $\mathsf{Post}_{\mathsf{u}} = \mathsf{Post}_{\mathsf{d}} = \{p \wedge \neg q, \neg p \wedge q\}$.

Suppose we have an MDP $M = \langle S, P, V, R\rangle$ over $\mathcal{A}$, where

- $S = \{s_A, s_B\}$,
- $P(s_A, \mathsf{d})(s') = \frac{1}{2}$ and $P(s_B, \mathsf{u})(s') = \frac{1}{2}$ for all $s' \in S$
- $V(s_A) = p$ and $V(s_B) = q$, and
- $R(s_A, \mathsf{d}, s_A) = 0$, $R(s_A, \mathsf{d}, s_B) = 1$, $R(s_B, \mathsf{u}, s_B) = 0$, and $R(s_B, \mathsf{u}, s_A) = 1$.

The MDP M is drawn in Figure 5.1. If we follow Definition 16 and unravel M from s_A, $M^{\mathsf{unrav}}_{s_A} = \{\langle S', P', V', R'\rangle\}$, with the following sets and functions.

The states in $M^{\mathsf{unrav}}_{s_A}$, are by definition all paths in M starting from s_A: s_A, $s_A\,\mathsf{d}\,s_A$, $s_A\,\mathsf{d}\,s_B$, $s_A\,\mathsf{d}\,s_A\,\mathsf{d}\,s_A$, and so on. Since $P'(\mathbf{h}, a)\downarrow$ iff $P(\mathsf{last}(\mathbf{h}), a)\downarrow$, the effect of action d is only defined in states ending in s_A, and u is only defined in states ending in s_B.

If we then apply the definitions for P', V', and R' to our example, we end up with the unravelled MDP $M^{\mathsf{unrav}}_{s_A}$ drawn in Figure 5.2.

Before demonstrating that this unravelled MDP satisfies the same formulas as the original MDP, we confirm that our unravelling approach maintains the well-definedness of the MDP, as per our requirements in Definition 5.

▶ **Lemma 4 (Well-definedness of M^{unrav}_s).** *If some MDP $M = \langle S, P, V, R\rangle$ that is defined over some set of actions $\mathcal{A}$ and propositions Prop is well-defined, then the unravelled MDP $M^{\mathsf{unrav}}_s = \langle S', P', V', R'\rangle$ is also well-defined for any $s \in S$.*

Proof. Suppose some MDP $M = \langle S, P, V, R \rangle$ is well-defined. M satisfies requirement (i), so for all s' in S there exists some a in $\mathcal{A}$ such that $M, s' \vDash \mathsf{pre}_a$. Since for any $\mathbf{h} \in S'$ it holds that $\mathsf{last}(\mathbf{h}) \in S$, this implies that $M, \mathsf{last}(\mathbf{h}) \vDash \mathsf{pre}_a$ for some $a \in \mathcal{A}$. Since pre_a is a conjunction of literals and $V'(\mathbf{h}) = V(\mathsf{last}(\mathbf{h}))$, $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \mathsf{pre}_a$ holds exactly when $M, \mathsf{last}(\mathbf{h}) \vDash \mathsf{pre}_a$. For any $\mathbf{h} \in S'$, there is some a in $\mathcal{A}$ such that $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \mathsf{pre}_a$ (namely, the a such that $M, \mathsf{last}(\mathbf{h}) \vDash \mathsf{pre}_a$). Thus, M_s^{unrav} satisfies requirement (i).

Additionally, M satisfies requirement (ii). Thus, $P(s,a)\downarrow$ iff $M, s \vDash \mathsf{pre}_a$ holds for all $s \in S$ and $a \in \mathcal{A}$. By construction, $P'(\mathbf{h}, a)\downarrow$ exactly when $P(\mathsf{last}(\mathbf{h}), a)\downarrow$. Moreover, we have determined in the previous step that since preconditions are conjunctions of literals, $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \mathsf{pre}_a$ holds exactly when $M, \mathsf{last}(\mathbf{h}) \vDash \mathsf{pre}_a$. Combining these two observations gives us directly that $P'(\mathbf{h}, a)\downarrow$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \mathsf{pre}_a$, thereby showing that M_s^{unrav} satisfies requirement (ii).

Lastly, M satisfies requirement (iii). This requirement has two parts.

By requirement (iii.i), given an $s \in S$ and $a \in \mathcal{A}$ such that $P(s,a)\downarrow$ it holds that for all $s' \in S$ where $P(s,a)(s') > 0$, there is exactly one $\mathsf{post}_{a,i}$ in Post_a such that $M, s' \vDash \mathsf{post}_{a,i}$. To prove requirement (iii.i) also holds for M_s^{unrav}, we choose an arbitrary $\mathbf{h} \in S'$ and $a \in \mathcal{A}$ such that $P'(\mathbf{h}, a)\downarrow$ and an $s' \in S$ such that $P'(\mathbf{h}, a)(\mathbf{h}as') > 0$. By Definition 16, we know that $P(\mathsf{last}(\mathbf{h}), a)\downarrow$ and $P(\mathsf{last}(\mathbf{h}), a)(s') > 0$. By (iii.i), we then also know there is exactly one $\mathsf{post}_{a,i}$ such that $M, s' \vDash \mathsf{post}_{a,i}$. Since individual postconditions are conjunctions of literals, and $V'(\mathbf{h}) = V(\mathsf{last}(\mathbf{h}))$, we can say $M_s^{\mathsf{unrav}}, \mathbf{h}as' \vDash \mathsf{post}_{a,i}$ is implied by the truth of $M, s' \vDash \mathsf{post}_{a,i}$. Thus, there is exactly one $\mathsf{post}_{a,i}$ such that $M_s^{\mathsf{unrav}}, \mathbf{h}as' \vDash \mathsf{post}_{a,i}$, proving M_s^{unrav} satisfies requirement (iii.i).

By requirement (iii.ii), given an $s \in S$ and $a \in \mathcal{A}$ such that $P(s,a)\downarrow$ it holds that for all $\mathsf{post}_{a,i}$ in Post_a there exists exactly one s' such that $P(s,a)(s') > 0$ and $M, s' \vDash \mathsf{post}_{a,i}$. To prove that requirement (iii.ii) also holds for M_s^{unrav}, we choose an arbitrary $\mathbf{h} \in S'$ and $a \in \mathcal{A}$ such that $P'(\mathbf{h}, a)\downarrow$ and an arbitrary $\mathsf{post}_{a,i} \in \mathsf{Post}_a$. By Definition 16, we know that $P(\mathsf{last}(\mathbf{h}), a)\downarrow$. Therefore, by (iii.ii), there is exactly one s' such that $P(s,a)(s') > 0$ and $M, s' \vDash \mathsf{post}_{a,i}$. By definition of P' and the the established knowledge that $M_s^{\mathsf{unrav}}, \mathbf{h}as' \vDash \mathsf{post}_{a,i}$ is implied by the truth of $M, s' \vDash \mathsf{post}_{a,i}$, we can conclude that there exists exactly one $\mathbf{h}'$ (namely $\mathbf{h}as'$) such that $P'(\mathbf{h}, a)(\mathbf{h}')$ and $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \mathsf{post}_{a,i}$. This proves M_s^{unrav} also satisfies requirement (iii.ii).

Since M and s were arbitrarily chosen, and M_s^{unrav} is shown to satisfy requirements (i), (ii), and (iii), we can conclude that the well-definedness of M_s^{unrav} is implied by the well-definedness of M for any MDP and starting state. □

5.1.2 Unravelled policies

Now that we have verified that our unravelling method results in a well-defined MDP, we continue towards our goal of showing that this unravelled MDP satisfies a formula φ, exactly when the original MDP satisfies it. As a part of this effort, we show that a k-step policy that satisfies policy formula ξ exists in MDP M, exactly when a k-step policy that satisfies ξ exists in MDP M_s^{unrav}. To this end, we define unravelled analogues of bounded policies in the initial MDP and show well-definedness of the unravelled policies. Moreover, we show that when we look at the respective sets of paths that the original policy and the unravelled one generate, each path has exactly one counterpart in the other set which shares its probability, cumulative reward, and path formula satisfaction.

▶ **Definition 17 (Unravelled policy).** Take an MDP M, a state s in M, a state $\mathbf{h}$ in M_s^{unrav} and a k-step policy $\pi^{\mathsf{last}(\mathbf{h})}$ in M. The k-step policy $\pi^{\mathsf{unrav},\mathbf{h}}$ in M_s^{unrav}, is defined as

$$\pi^{\mathsf{unrav},\mathbf{h}}(\mathbf{h}_1\mathbf{h}_2\dots\mathbf{h}_n) = \pi^{\mathsf{last}(\mathbf{h})}(\mathsf{last}(\mathbf{h}_1)\mathsf{last}(\mathbf{h}_2)\dots\mathsf{last}(\mathbf{h}_n)), \text{ for all } \mathbf{h}_i \in S' \text{ and } n \leq k. \tag{5.3}$$

▶ **Lemma 5 (Well-definedness of $\pi^{\mathsf{unrav},\mathbf{h}}$).** *Take an MDP M, a state s in M, a state $\mathbf{h}$ in M_s^{unrav}. A k-step policy $\pi^{\mathsf{last}(\mathbf{h})}$ in M is well-defined iff $\pi^{\mathsf{unrav},\mathbf{h}}$ is a well-defined k-step policy in M_s^{unrav}.*

Proof. Let M be an MDP $\langle S, P, V, R \rangle$, s a state in M, and $\mathbf{h}$ a state in M_s^{unrav}. We prove the bi-implication of this lemma in two parts.

For the left-to-right direction, suppose $\pi^{\mathsf{last}(\mathbf{h})}$ is a well-defined k-step policy in M. This means by Definition 6 that for all $s_1 \dots s_n \in S^{\leq k}_{\mathsf{last}(\mathbf{h})}$, it holds that $P(s_n, \pi(s_1 \dots s_n))\downarrow$. Since $\mathsf{last}(\mathbf{h}') \in S$ for all $\mathbf{h}' \in S'$, it follows that $\mathsf{last}(\mathbf{h}_1)\dots\mathsf{last}(\mathbf{h}_n) \in S^{\leq k}_{\mathsf{last}(\mathbf{h})}$ for all $\mathbf{h}_1 \dots \mathbf{h}_n \in (S')^{\leq k}_{\mathbf{h}}$. Thus, we know that for

all $\mathbf{h}_1 \dots \mathbf{h}_n \in (S')^{\leq k}_{\mathbf{h}}$, $P(\mathsf{last}(\mathbf{h}_n), \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n)))\downarrow$. By Definition 16, we know that

$$P(\mathsf{last}(\mathbf{h}_n), \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n))) = P'(\mathbf{h}_n, \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n))). \tag{5.4}$$

By Equation (5.3), we know that

$$P'(\mathbf{h}_n, \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n))) = P'(\mathbf{h}_n, \pi^{\mathsf{unrav}}(\mathbf{h}_1 \dots \mathbf{h}_n)). \tag{5.5}$$

Thus, we know that $P'(\mathbf{h}_n, \pi^{\mathsf{unrav}}(\mathbf{h}_1 \dots \mathbf{h}_n))\downarrow$ for all $\mathbf{h}_1 \dots \mathbf{h}_n \in (S')^{\leq k}_{\mathbf{h}}$. By Definition 6, this means that $\pi^{\mathsf{unrav},\mathbf{h}}$ is a well-defined k-step policy in M^{unrav}_s. This concludes the proof for the left-to-right direction of the implication.

For the right-to-left direction, suppose $\pi^{\mathsf{unrav},\mathbf{h}}$ is a well-defined k-step policy in M^{unrav}_s. This means by Definition 6 that $P'(\mathbf{h}_n, \pi^{\mathsf{unrav}}(\mathbf{h}_1 \dots \mathbf{h}_n))\downarrow$ for all $\mathbf{h}_1 \dots \mathbf{h}_n \in (S')^{\leq k}_{\mathbf{h}}$. By Equation (5.3), we know that

$$P'(\mathbf{h}_n, \pi^{\mathsf{unrav}}(\mathbf{h}_1 \dots \mathbf{h}_n)) = P'(\mathbf{h}_n, \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n))). \tag{5.6}$$

By Definition 16, we know that

$$P'(\mathbf{h}_n, \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n))) = P(\mathsf{last}(\mathbf{h}_n), \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n))). \tag{5.7}$$

Thus, we know that for all $\mathbf{h}_1 \dots \mathbf{h}_n \in (S')^{\leq k}_{\mathbf{h}}$, $P(\mathsf{last}(\mathbf{h}_n), \pi(\mathsf{last}(\mathbf{h}_1) \dots \mathsf{last}(\mathbf{h}_n)))\downarrow$.

Note that $\{\mathsf{last}(\mathbf{h}_i) | \mathbf{h}_i \in (S')^{\leq k}_{\mathbf{h}}\} = S^{\leq k}_{\mathsf{last}(\mathbf{h})}$ by the definition of S'. It follows that for all $s_1 \dots s_n \in S^{\leq k}_{\mathsf{last}(\mathbf{h})}$, $P(s_n, \pi(s_1 \dots s_n))\downarrow$. By Definition 6, this means that $\pi^{\mathsf{last}(\mathbf{h})}$ is a well-defined k-step policy in M. □

▶ **Lemma 6 (Probability and reward-preserving bijection).** *Take a path formula γ, an MDP $M = \langle S, P, V, R\rangle$, a state s in M, a state $\mathbf{h}$ in M^{unrav}_s, and a k-step policy $\pi^{\mathsf{last}(\mathbf{h})}$ in M. Let it be for all state sub-formulas φ in γ and states $\mathbf{h}'$ in M^{unrav}_s, that $M, \mathsf{last}(\mathbf{h}') \vDash \varphi$ iff $M^{\mathsf{unrav}}_s, \mathbf{h}' \vDash \varphi$. Then, we have that the function $\theta : \{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\} \to \{\lambda \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M^{\mathsf{unrav}}_s, \lambda \vDash \gamma\}$ defined as*

$$\theta(s_1 a_1 s_2 \dots s_k a_k s_{k+1}) = \mathbf{h} a_1 (\mathbf{h} a_1 s_2) \dots (\mathbf{h} a_1 s_2 a_2 \dots s_k) a_k (\mathbf{h} a_1 s_2 a_2 \dots s_k a_k s_{k+1}). \tag{5.8}$$

is a probability and reward-preserving bijection, such that

$$\mu(\lambda) = \mu(\theta(\lambda)) \text{ and} \tag{5.9}$$

$$R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) = R'(\theta(\lambda)_{s_i}, \theta(\lambda)_{a_i}, \theta(\lambda)_{s_{i+1}}) \text{ for all } 0 < i \leq k. \tag{5.10}$$

Proof. Take a path formula γ, an MDP $M = \langle S, P, V, R\rangle$, a state s in M, a state $\mathbf{h}$ in M^{unrav}_s, and a k-step policy $\pi^{\mathsf{last}(\mathbf{h})}$ in M. Let it be for all state sub-formulas φ in γ and states $\mathbf{h}'$ in M^{unrav}_s, that $M, \mathsf{last}(\mathbf{h}) \vDash \varphi$ iff $M^{\mathsf{unrav}}_s, \mathbf{h} \vDash \varphi$. Let set $B^\gamma = \{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\}$ and $C^\gamma = \{\lambda \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M^{\mathsf{unrav}}_s, \lambda \vDash \gamma\}$, i.e the paths from both respective policies that satisfy γ.

From the definition of P' and R', it follows immediately that:

$$\mu(\lambda) = \mu(\theta(\lambda)) \text{ and} \tag{5.11}$$

$$R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) = R'(\theta(\lambda)_{s_i}, \theta(\lambda)_{a_i}, \theta(\lambda)_{s_{i+1}}) \text{ for all } 0 < i \leq k. \tag{5.12}$$

We must also show that this function is a bijection between B^γ and C^γ. We do this by showing that in general for any $\lambda \in B^\gamma$, $\theta(\lambda) \in C^\gamma$ and that for any $\lambda' \in C^\gamma$, there is some $\lambda \in B^\gamma$ such that $\lambda' = \theta(\lambda)$. We start with the former.

Choose an arbitrary path $\lambda \in B^\gamma$. Since $\lambda \in B^\gamma$, it satisfies γ, which we assume is of the normal form

$$\gamma = \bigwedge_{v=1}^{e} \bigvee_{w=1}^{f} (\mathsf{X}^{t_{v,w}} \varphi_{v,w}), \text{ where } \varphi_{v,w} = \begin{cases} \text{a state formula } \psi, \\ \mathsf{do}_a \text{ or } \neg \mathsf{do}_a, \text{ or} \\ \mathsf{C}^u_{\bowtie r}. \end{cases} \tag{5.13}$$

Given a v, we know there exists a w such that $M, \lambda \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$. We have three scenarios.

- If $\varphi_{v,w}$ is a state formula ψ, we know that $M, \lambda_{t_{v,w}+1} \vDash \varphi_{v,w}$. Since for all states $\mathbf{h}'$ in M^{unrav}_s, $M, \mathsf{last}(\mathbf{h}') \vDash \varphi_{v,w}$ iff $M^{\mathsf{unrav}}_s, \mathbf{h}' \vDash \varphi_{v,w}$, we get that $M^{\mathsf{unrav}}_s, \theta(\lambda)_{t_{v,w}+1} \vDash \varphi_{v,w}$, which we can work back to $M^{\mathsf{unrav}}_s, \theta(\lambda) \vDash \mathsf{X}^{t_{v,w}} \psi$.

- If $\varphi_{v,w}$ is do_a, then we know that a is the $t_{v,w}+1$-th action of λ. By definition, a is the $t_{v,w}+1$-th action in $\theta(\lambda)$ if and only if it is the $t_{v,w}+1$-th action in λ. Thus, $M_s^{\mathsf{unrav}}, \theta(\lambda) \vDash \mathsf{X}^{t_{v,w}}\mathsf{do}_a$ follows immediately. The same holds if $\varphi_{v,w}$ is $\neg\mathsf{do}_a$.
- If $\varphi_{v,w}$ is $\mathsf{C}^u_{\bowtie r}$, we know that $\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) \bowtie r$. By Equations (5.11) and (5.12), we know that the following equality holds.

$$\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) = \sum_{i=t_{v,w}+1}^{t_{v,w}+u} R'(\theta(\lambda)_{s_i}, \theta(\lambda)_{a_i}, \theta(\lambda)_{s_{i+1}}) \bowtie r \tag{5.14}$$

  From this we can conclude that $M_s^{\mathsf{unrav}}, \theta(\lambda) \vDash \mathsf{X}^{t_{v,w}}\varphi_{v,w}$ if $\varphi_{v,w} = \mathsf{C}^u_{\bowtie r}$.

These three cases show that for all $1 \leq v \leq e$ there exists some $1 \leq w \leq f$ such that $M_s^{\mathsf{unrav}}, \theta(\lambda) \vDash \mathsf{X}^{t_{v,w}}\varphi_{v,w}$. Thus, $\lambda \in B^\gamma$ implies $\theta(\lambda) \in C^\gamma$.

Now, take some $\lambda' \in C^\gamma$. By Definition 16, this is a path in which all the states are a copy of the previous state with the addition of a new action and outcome. More formally,

$$\lambda' = \mathbf{h}a_1(\mathbf{h}a_1s_2)\dots(\mathbf{h}a_1s_2a_2\dots s_n)a_n(\mathbf{h}a_1s_2a_2\dots s_na_ns_{n+1}). \tag{5.15}$$

If we let

$$\lambda = \mathsf{last}(\mathbf{h})a_1s_2\dots a_ns_{n+1}, \tag{5.16}$$

we have the desired $\lambda' = \theta(\lambda)$. We now show that $\lambda \in B^\gamma$ (with a proof nearly identical to the other direction), to complete the proof.

Since $\lambda' \in C^\gamma$, it satisfies γ, which we assume is of the normal form given in Equation (5.13). Given a v, we know there exists a w such that $M_s^{\mathsf{unrav}}, \lambda' \vDash \mathsf{X}^{t_{v,w}}\varphi_{v,w}$. We have three scenarios.

- If $\varphi_{v,w}$ is a state formula ψ, we know that $M_s^{\mathsf{unrav}}, \lambda'_{t_{v,w}+1} \vDash \varphi_{v,w}$. Since for all states $\mathbf{h}'$ in M_s^{unrav}, $M, \mathsf{last}(\mathbf{h}') \vDash \varphi_{v,w}$ iff $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \varphi_{v,w}$, we get that $M, \lambda_{t_{v,w}+1} \vDash \varphi_{v,w}$, which we can work back to $M, \lambda \vDash \mathsf{X}^{t_{v,w}}\psi$.
- If $\varphi_{v,w}$ is do_a, then we know that a is the $t_{v,w}+1$-th action of λ'. By definition, a is the $t_{v,w}+1$-th action in λ' if and only if it is the $t_{v,w}+1$-th action in λ. Thus, $M, \lambda \vDash \mathsf{X}^{t_{v,w}}\mathsf{do}_a$ follows immediately. The same holds if $\varphi_{v,w}$ is $\neg\mathsf{do}_a$.
- If $\varphi_{v,w}$ is $\mathsf{C}^u_{\bowtie r}$, we know that $\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda'_{s_i}, \lambda'_{a_i}, \lambda'_{s_{i+1}}) \bowtie r$. By Equations (5.11) and (5.12), we know that the following equality holds.

$$\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda'_{s_i}, \lambda'_{a_i}, \lambda'_{s_{i+1}}) = \sum_{i=t_{v,w}+1}^{t_{v,w}+u} R'(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) \bowtie r \tag{5.17}$$

  From this we can conclude that $M, \lambda \vDash \mathsf{X}^{t_{v,w}}\varphi_{v,w}$ if $\varphi_{v,w} = \mathsf{C}^u_{\bowtie r}$.

These three cases show that for all $1 \leq v \leq e$ there exists some $1 \leq w \leq f$ such that $M, \lambda \vDash \mathsf{X}^{t_{v,w}}\varphi_{v,w}$. Thus, for our arbitrary $\lambda' \in C^\gamma$, there is some $\lambda \in B^\gamma$ such that $\lambda' = \theta(\lambda)$. □

So far, we have defined unravelled counterparts $\pi^{\mathsf{unrav},\mathbf{h}}$ to policies $\pi^{\mathsf{last}(\mathbf{h})}$ in M, verified that these are always well-defined policies in M_s^{unrav}, and proven that a probability and reward-preserving bijection exists between the paths of both policies satisfying a path formula γ. We continue to the next step: showing that our unravelled MDP satisfies precisely the same formulas as our original MDP.

▶ **Lemma 7 (Equivalence of M_s^{unrav}).** *Take an MDP $M = \langle S, P, V, R\rangle$ and any state $s \in S$. For any $\mathbf{h} \in S'$ and state formula φ, the following equivalence holds: $M, \mathsf{last}(\mathbf{h}) \vDash \varphi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \varphi$.*

Proof. We prove by induction that the described equivalence holds for all possible state formulas φ. Note that the state formula φ can have one of these four forms: p, for some $p \in \mathsf{Prop}$; $\neg\psi$, for some state formula ψ; $\psi \wedge \psi'$, for some state formulas ψ and ψ'; or $\Diamond^k(\xi)$, for some policy formula ξ.

Our base case for the induction is $\varphi = p$. Since by Definition 16, $V(\mathsf{last}(\mathbf{h})) = V'(\mathbf{h})$, it is also true that $p \in V(\mathsf{last}(\mathbf{h}))$ iff $p \in V'(\mathbf{h})$, which directly lets us conclude that $M, \mathsf{last}(\mathbf{h}) \vDash p$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash p$. Our inductive steps for the two Boolean operators follow directly from Definition 16 and the properties (e.g. distributivity) of these operators.

For our final case, $\varphi = \Diamond^k(\xi)$, we assume that our policy formula is in normal form:

$$\xi = \bigwedge_{m=1}^{c} \bigvee_{q=1}^{d} \xi_{m,q}, \text{ where } \xi_{m,q} = \begin{cases} \mathsf{P}_{\bowtie c}(\gamma), \text{ or} \\ \mathsf{E}^{l,u}_{\bowtie r}. \end{cases} \tag{5.18}$$

where $\xi_{m,q}$, can either be an expected reward operator $\mathsf{E}^{l,u}_{\bowtie r}$, or a probability operator $\mathsf{P}_{\bowtie c}(\gamma)$. Moreover, we assume each path formula γ is written in the following normal form:

$$\gamma = \bigwedge_{v=1}^{e} \bigvee_{w=1}^{f} (\mathsf{X}^{t_{v,w}} \varphi_{v,w}), \text{ where } \varphi_{v,w} = \begin{cases} \text{a state formula } \psi, \\ \mathsf{do}_a \text{ or } \neg\mathsf{do}_a, \text{ or} \\ \mathsf{C}^{u}_{\bowtie r}. \end{cases} \tag{5.19}$$

Suppose that for all state sub-formulas ψ that occur in ξ and for all $\mathbf{h}' \in S'$ it holds that $M, \mathsf{last}(\mathbf{h}') \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \psi$. We show that it follows from this induction hypothesis that $M, \mathsf{last}(\mathbf{h}) \vDash \Diamond^k(\xi)$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$, in two parts.

1. First, left to right: $M, \mathsf{last}(\mathbf{h}) \vDash \Diamond^k(\xi)$ implies $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$.

   Suppose $M, \mathsf{last}(\mathbf{h}) \vDash \Diamond^k(\xi)$, which means by definition that there exists some k-step policy $\pi^{\mathsf{last}(\mathbf{h})}$ in M, such that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi$. By Lemma 5, there exists an unravelled counterpart of this policy, the k-step policy $\pi^{\mathsf{unrav},\mathbf{h}}$ in M_s^{unrav}. Note that if we can show for any ξ that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi$ implies $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi$, we have proven the left-to-right direction of the inductive step.

   Since ξ is written in normal form, we know that given an m, there exists a q such that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi_{m,q}$. We consider two scenarios.

   - Suppose $\xi_{m,q} = \mathsf{E}^{l,u}_{\bowtie r}$. From our assumption that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, it follows that the expected reward of the policy from step l up to and including u, is $\bowtie r$:

   $$\sum_{\lambda \in \mathsf{Paths}\left(\pi^{\mathsf{last}(\mathbf{h})}\right)} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R(s_i, a_i, s_{i+1}) \right) \right) \bowtie r. \tag{5.20}$$

   This set below the first summation, $\mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})})$, can be the domain B^γ of our bijection θ, if $\gamma = \top$. This means we can map each path from policy $\pi^{\mathsf{last}(\mathbf{h})}$ to a path from $\pi^{\mathsf{unrav},\mathbf{h}}$ with the same probability and cumulative reward: $\theta(\lambda) = \lambda'$. Thus,

   $$\begin{aligned} &\sum_{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R(s_i, a_i, s_{i+1}) \right) \right) \\ &= \sum_{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}})} \left(\mu(\lambda') \cdot \sum_{i=l}^{u} \left(R'\left(\lambda'_{s_i}, \lambda'_{a_i}, \lambda'_{s_{i+1}} \right) \right) \right) \bowtie r, \end{aligned} \tag{5.21}$$

   from which it immediately follows that $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \mathsf{E}^{l,u}_{\bowtie r}$.

   - Suppose $\xi_{m,q} = \mathsf{P}_{\bowtie c}(\gamma)$. By our assumption that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, we have that

   $$\mu(\{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\}) \bowtie c. \tag{5.22}$$

   Since the bijection θ between $\{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\}$ and $\{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda' \vDash \gamma\}$ preserves probabilities, we infer that

   $$\mu(\{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\}) = \mu(\{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda' \vDash \gamma\}) \bowtie c. \tag{5.23}$$

   Thus, it immediately follows that $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \mathsf{P}_{\bowtie c}(\gamma)$.

   Now, we have shown for all ξ that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi$ implies $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi$, if we assume our induction hypothesis. This concludes the left-to-right part of our inductive step.

2. Second, right to left: $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$ implies $M, \mathsf{last}(\mathbf{h}) \vDash \Diamond^k(\xi)$.

   Suppose $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$, which means by definition that there exists some k-step policy $\pi^{\mathsf{unrav},\mathbf{h}}$ in M_s^{unrav}, such that $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi$. Where in the previous step we defined an unravelled counterpart of the original policy, we now translate a policy in the unravelled model into one in the original model: let $\pi^{\mathsf{last}(\mathbf{h})}(\mathsf{last}(\mathbf{h}_1)\mathsf{last}(\mathbf{h}_2)\dots\mathsf{last}(\mathbf{h}_n)) = \pi^{\mathsf{unrav},\mathbf{h}}(\mathbf{h}_1\mathbf{h}_2\dots\mathbf{h}_n)$. The proof that $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi$ implies $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi$ then follows in the same (though opposite) way from the induction hypothesis and the fact that there exists a bijection θ that preserves both probabilities and rewards.

   Since ξ is written in normal form, we know that given an m, there exists a q such that the statement $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi_{m,q}$ holds. We consider two scenarios.

   - Suppose $\xi_{m,q} = \mathsf{E}^{l,u}_{\bowtie r}$. From our assumption that $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, it follows that the expected reward of the policy from step l up to and including u, is $\bowtie r$:

   $$\sum_{\lambda \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R(s_i, a_i, s_{i+1}) \right) \right) \bowtie r. \tag{5.24}$$

   This set below the first summation, $\mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}})$, can be the range C^γ of the bijection θ by Lemma 6, if $\gamma = \top$. This means we can map each path from policy $\pi^{\mathsf{unrav},\mathbf{h}}$ to a path from $\pi^{\mathsf{last}(\mathbf{h})}$ with the same probability and cumulative reward: the path λ such that $\theta(\lambda) = \lambda'$. Thus,

   $$\begin{aligned} &\sum_{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}})} \left(\mu_\pi^{M_s^{\mathsf{unrav}},\mathbf{h}}(\lambda') \cdot \sum_{i=l}^{u} \left(R'\left(\lambda'_{s_i}, \lambda'_{a_i}, \lambda'_{s_{i+1}} \right) \right) \right) \\ = &\sum_{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})})} \left(\mu_\pi^{M,\mathsf{last}(\mathbf{h})}(\lambda) \cdot \sum_{i=l}^{u} \left(R(s_i, a_i, s_{i+1}) \right) \right) \bowtie r, \end{aligned} \tag{5.25}$$

   from which it immediately follows that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \mathsf{E}^{l,u}_{\bowtie r}$.

   - Suppose $\xi_{m,q} = \mathsf{P}_{\bowtie c}(\gamma)$. By our assumption that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, we have that

   $$\mu(\{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda' \vDash \gamma\}) \bowtie c. \tag{5.26}$$

   Since the bijection θ between $\{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\}$ and $\{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda' \vDash \gamma\}$ preserves probabilities by Lemma 6, we infer that

   $$\mu(\{\lambda' \in \mathsf{Paths}(\pi^{\mathsf{unrav},\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda' \vDash \gamma\}) = \mu(\{\lambda \in \mathsf{Paths}(\pi^{\mathsf{last}(\mathbf{h})}) \mid M, \lambda \vDash \gamma\}) \bowtie c. \tag{5.27}$$

   Thus, it immediately follows that $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \mathsf{P}_{\bowtie c}(\gamma)$.

   Now, we have shown for all ξ that $M_s^{\mathsf{unrav}}, \pi^{\mathsf{unrav},\mathbf{h}} \vDash \xi$ implies $M, \pi^{\mathsf{last}(\mathbf{h})} \vDash \xi$, if we assume our induction hypothesis. This concludes the left-to-right part of our inductive step.

We briefly summarise what we know so far. For an arbitrary MDP $M = \langle S, P, V, R \rangle$ and arbitrary state $s \in S$, from which we construct $M_s^{\mathsf{unrav}} = \langle S', P', V', R' \rangle$, it holds for any $\mathbf{h} \in S'$ that:

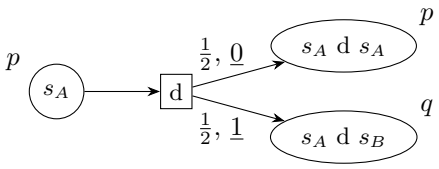
(i) *Base case*: $M, \mathsf{last}(\mathbf{h}) \vDash p$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash p$ for any $p \in \mathsf{Prop}$.

(ii) *Inductive step 1*: Given that state formulas ψ and ψ' hold for $\mathsf{last}(\mathbf{h})$ in M iff they hold for $\mathbf{h}$ in M_s^{unrav}, it follows that the formula $\psi \wedge \psi'$ holds for $\mathsf{last}(\mathbf{h})$ in M iff it holds for $\mathbf{h}$ in M_s^{unrav}. This step was not proven explicitly but follows directly from the definition.

(iii) *Inductive step 2*: Given that the state formula ψ holds for $\mathsf{last}(\mathbf{h})$ in M iff it holds for $\mathbf{h}$ in M_s^{unrav}, it follows that the formula $\neg\psi$ holds for $\mathsf{last}(\mathbf{h})$ in M iff it holds for $\mathbf{h}$ in M_s^{unrav}. This step was also not proven explicitly but again follows directly from the definition.

(iv) *Inductive step 3*: Given that all state formulas ψ within ξ hold for any $\mathbf{h}'$ in M_s^{unrav} iff they hold for $\mathsf{last}(\mathbf{h}')$, it follows that the formula $\Diamond^k(\xi)$ holds for $\mathsf{last}(\mathbf{h})$ in M iff it holds for $\mathbf{h}$ in M_s^{unrav}.

Thus, we have proven through induction that for any state formula φ and any $\mathbf{h} \in S'$, it holds that $M, \mathsf{last}(\mathbf{h}) \vDash \varphi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \varphi$. □



▶ **Figure 5.3:** A bounded version of the unravelled MDP M_s^{unrav} from Example 11.

5.1.3 Bounding the MDP

So far we have found a 'recipe' for creating an unravelled version of an MDP that satisfies the same state formulas as the original MDP. However, this does not yet bring us to our eventual goal of finding an MDP of a particular bounded size. After all, the set of states in M_s^{unrav} is of infinite size. Intuitively, we do not require all the information embedded in this boundless model, since our formulas can only assert facts about some bounded amount of steps. Therefore, we look into ways to cut off M_s^{unrav} to only the size we truly need. We reuse the concept of *policy depth* from PLBP and therefore cite the definition verbatim from [Motamed et al., 2023]:

▶ **Definition 18 (Policy depth).** "The policy depth of a state formula is inductively defined by putting $\mathsf{pd}(\varphi) = 0$ if φ is atomic, $\mathsf{pd}(\varphi \wedge \psi) = \max\{\mathsf{pd}(\varphi), \mathsf{pd}(\psi)\}$, $\mathsf{pd}(\neg\varphi) = \mathsf{pd}(\varphi)$, and $\mathsf{pd}(\Diamond^k(\xi)) = k + \max\{\mathsf{pd}(\varphi) \mid \text{state formula } \varphi \text{ appears in } \xi\}$."

This concept of policy depth gives us a precise formula for how far from our starting state we must be able to look to determine the truth of our formula. This means that the states that are *not* reachable from s within k steps can be eliminated without consequence for the truth of the formula. For example, if we have the formula $\varphi = \Diamond^1(\mathsf{P}_{>0}(\mathsf{X}p))$, which has $\mathsf{pd}(\varphi) = 1$, we can cut off the states in the unravelled MDP from Example 11 that are 2 or more steps away from the starting node s_A.

However, we can easily see that such an MDP is not well-defined according to our requirements: every state must have at least one executable action, to prevent deadlocks. The solution given in [Motamed et al., 2023] is to "let the probabilistic transition function behave arbitrarily" after this point, but this is a bit too simplistic.[1] After all, it does not suffice to allow just *any* action there, since precisely those actions must be executable for which the preconditions are satisfied ($\mathsf{P}(s,a)\downarrow$ iff $s \vDash \mathsf{pre}_a$).

To patch up our ill-defined result, we can take three approaches: (1) we change the definition of being well-defined to include our result, (2) we introduce a standard do_nothing action, or (3) we adapt our result to meet the requirements of being well-defined.

For the first – and most blunt – option, we could remove the requirement that there must not be deadlocks in the MDP. The states just before the cut-off may still satisfy the precondition of at least one action, which means our MDP violates another requirement as well: $\mathsf{P}(s,a)\downarrow$ iff $s \vDash \mathsf{pre}_a$ for all states and actions. If we remove both those requirements, there are some consequences we must consider. For example, if we have that not all paths have an infinite length due to running into dead ends, we have to redefine our semantics to account for this. What does it mean for our $\mathsf{P}_{\bowtie}(\gamma^k)$ if some paths that result from a policy are shorter than k? The effects of removing both requirements from the MDPs are so pervasive, that it results in a different logic altogether. We conclude that it is best to avoid this solution if there are other options that give us a well-defined MDP.

For the second approach, we require that there is always a do_nothing action in $\mathcal{A}$ that has $\mathsf{pre}_{\mathsf{do\_nothing}} = \top$ and $\mathsf{Post}_{\mathsf{do\_nothing}} = \{\top\}$. The intuition is that this option can then be used in any state where a deadlock occurs after cutting off the unravelled MDP and that it allows for a deterministic transition back to the same state. The problem here is that in order to have a well-defined MDP, we must still allow any other actions for which the preconditions are satisfied to be executable, so it does not actually solve our original problem. Moreover, the precondition of do_nothing is always satisfied, and must therefore be executable in any state, not just those affected by the cut. This is undesirable in applications where the agent does not have the option to abstain from acting (e.g. in a game where two players make alternating moves). All in all, the second approach has undesirable side effects and does not result in a well-defined MDP,

[1] The discovery that the process of cutting off the unravelled MDP was more complex than it was described in [Motamed et al., 2023] was made together with N. Motamed during meetings together.

unless it is combined with the first approach of relaxing the requirements of well-definedness.

The third approach is a more delicate cutting technique that requires no changes to our set of actions or our requirements for the MDP. Rather, we add new states and transitions that patch up our result to create a well-defined MDP. Note that what happens exactly in terms of probabilities or rewards in this new part is not of concern to us for the satisfaction of φ.

For each state that is an end-point of our cut-off MDP, we have that there are some actions that must be executable due to their preconditions. Therefore, for each of the postconditions of these actions, we introduce a new state that satisfies it and add a transition to it with an arbitrary, but non-zero probability. These new states then also have some actions that must be executable due to their preconditions and require states which satisfy their postconditions to transition to.

To solve this reoccurring problem in one go, we instead add a whole group of new states for each action in $\mathcal{A}$, such that within the group for some action a, there is exactly one state s for each of the postconditions in Post_a such that $M, s \vDash \mathsf{post}_{a,i}$. For each of these new states, whatever the actions are that must be executable, we can then transition to the corresponding group of states that contains all required postconditions.

▶ **Definition 19 (Bounded MDP).** Given an MDP M, a state s in M, the unravelled MDP $M_s^{\mathsf{unrav}} = \langle S, P, V, R\rangle$ and a state formula φ, the *bounded MDP* $M_s^{\varphi} = \langle S', P', V', R'\rangle$ is defined as follows. We let S' be

$$S' = S^{\mathsf{reg}} \sqcup S^{\mathsf{add}} = \{s_1 a_1 \dots s_k a_k s_{k+1} \in S \mid k \leq \mathsf{pd}(\varphi)\} \sqcup \{s_{a,i} \mid \text{for } a \in \mathcal{A}, i \in [1, |\mathsf{Post}_a|]\}. \tag{5.28}$$

We let $P'(s', a)\downarrow$ iff $P(s', a)\downarrow$ for all $s' \in S^{\mathsf{reg}}$ and $a \in \mathcal{A}$. We let $P'(s', a)\downarrow$ iff $s' \vDash \mathsf{pre}_a$ for all $s' \in S^{\mathsf{add}}$. Then, whenever $P'(s', a)\downarrow$, we define the function P' as follows.

$$P'(s', a)(s'') = \begin{cases} P(s', a)(s'') & \text{for} \quad s', s'' \in S^{\mathsf{reg}} \\ \dfrac{1}{|\mathsf{Post}_a|} & \text{for} \quad s' = s_1 a_1 \dots s_{\mathsf{pd}(\varphi)+1}, s'' = s_{a,i} \text{ for some } i \in [1, |\mathsf{Post}_a|] \\ \dfrac{1}{|\mathsf{Post}_a|} & \text{for} \quad s' \in S^{\mathsf{add}}, s'' = s_{a,i} \text{ for some } i \in [1, |\mathsf{Post}_a|] \\ 0 & \text{otherwise} \end{cases} \tag{5.29}$$

We define $V'(s') = V(s')$ for all $s' \in S^{\mathsf{reg}}$ and we define $V'(s_{a,i})$ for all $s_{a,i} \in S^{\mathsf{add}}$ in such a way that $s_{a,i} \vDash \mathsf{post}_{a,i}$ and that $s_{a,i} \vDash \mathsf{pre}_{a'}$ for some arbitrary $a' \in \mathcal{A}$.[2] We define the reward function R' as

$$R'(s', a, s'') = \begin{cases} R(s', a)(s'') & \text{for} \quad s'' \in S^{\mathsf{reg}} \\ 0 & \text{for} \quad s'' \in S^{\mathsf{add}} \end{cases} \tag{5.30}$$

Note that we let the set of states of the bounded MDP be a disjoint union of two sets. The first one, S^{reg} contains all the states from M_s^{unrav} which represent a path of at most k steps, while the second one, S^{add}, contains one additional state for each of the postconditions of all actions in $\mathcal{A}$.

As an example, for the MDP M_s^{unrav} in Example 11 and $\mathsf{pd}(\varphi) = 1$, this definition gives the bounded MDP M_s^{φ} shown in Figure 5.4.

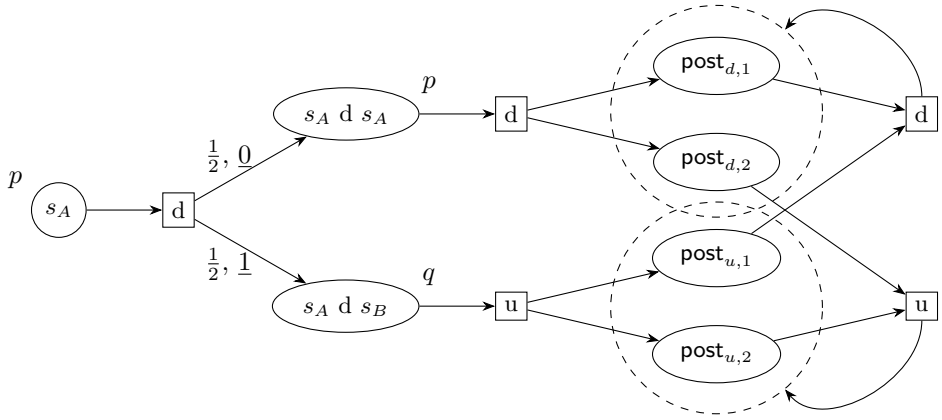
We now show that such bounded MDPs are well-defined by Definition 5.

▶ **Lemma 8 (Well-definedness of M_s^{φ}).** *Given an MDP M, a state s in M, the unravelled MDP $M_s^{\mathsf{unrav}} = \langle S, P, V, R\rangle$ and a state formula φ, we have that $M_s^{\varphi} = \langle S', P', V', R'\rangle$ is a well-defined MDP.*

Proof. Let M be an MDP, s an arbitrary state in M, and φ a state formula. By Lemma 4, we have that $M_s^{\mathsf{unrav}} = \langle S, P, V, R\rangle$ is a well-defined MDP and thus satisfies requirements (i), (ii), and (iii) from Definition 5.

MDP M_s^{unrav} satisfying (i) means that for all $s' \in S$ there exists some $a \in \mathcal{A}$ such that $M_s^{\mathsf{unrav}}, s' \vDash \mathsf{pre}_a$. By definition, the set of states S' is $S^{\mathsf{reg}} \cup S^{\mathsf{reg}}$. For all $s^{\mathsf{reg}} \in S^{\mathsf{reg}}$, we have that $V'(s^{\mathsf{reg}}) = V(s^{\mathsf{reg}})$. It

[2] Note that this is always possible, by the following restriction put on the postconditions in Definition 3: all postconditions must be consistent with the precondition of at least one action.



▶ **Figure 5.4:** A bounded version of the unravelled MDP M_s^{unrav} from Example 11, with additional states to satisfy the requirements of an MDP.

follows that for all $s^{\mathsf{reg}} \in S^{\mathsf{reg}}$, $M_s^\varphi, s^{\mathsf{reg}} \vDash \mathsf{pre}_a$ for some $a \in \mathcal{A}$: the a such that $M_s^{\mathsf{unrav}}, s^{\mathsf{reg}} \vDash \mathsf{pre}_a$. For all $s^{\mathsf{add}} \in S^{\mathsf{add}}$ we have that $V'(s^{\mathsf{add}})$ is constructed precisely so that $M_s^\varphi, s^{\mathsf{add}} \vDash \mathsf{pre}_a$ for some $a \in \mathcal{A}$. Thus, we have that for all $s' \in S'$, $M_s^\varphi, s' \vDash \mathsf{pre}_a$ for some $a \in \mathcal{A}$, which means that M_s^φ satisfies requirement (i).

MDP M_s^{unrav} satisfying (ii) means that for all $s' \in S$ and $a \in \mathcal{A}$, $P(s', a)\downarrow$ iff $M_s^{\mathsf{unrav}}, s' \vDash \mathsf{pre}_a$. For $s^{\mathsf{reg}} \in S^{\mathsf{reg}}$, we have that $P'(s^{\mathsf{reg}}, a)\downarrow$ iff $P(s^{\mathsf{reg}}, a)\downarrow$. Combining this with the fact that $V'(s^{\mathsf{reg}}) = V(s^{\mathsf{reg}})$, we have that $P'(s^{\mathsf{reg}}, a)\downarrow$ iff $M_s^\varphi, s^{\mathsf{reg}} \vDash \mathsf{pre}_a$. For all $s^{\mathsf{add}} \in S^{\mathsf{add}}$ we have that by definition $P'(s^{\mathsf{add}}, a)\downarrow$ iff $M_s^\varphi, s^{\mathsf{add}} \vDash \mathsf{pre}_a$. Thus, for all $s' \in S'$ and $a \in \mathcal{A}$, $P'(s', a)\downarrow$ iff $M_s^\varphi, s' \vDash \mathsf{pre}_a$, which means that M_s^φ satisfies requirement (ii).

Lastly, M_s^{unrav} satisfies requirement (iii) which has two parts. We first choose an $s' \in S'$ and $a \in \mathcal{A}$ such that $P'(s', a)\downarrow$, and an $s'' \in S'$ such that $P'(s', a)(s'') > 0$. Since $P'(s', a)(s'') > 0$, it must be, by definition of P', that either $s', s'' \in S^{\mathsf{reg}}$, or $s'' = s_{a,i}$ for some $i \in [1, |\mathsf{Post}_a|]$. In the case that $s', s'' \in S^{\mathsf{reg}}$, we have that $P(s', a)\downarrow$ and $P(s', a)(s'') > 0$. Since M_s^{unrav} satisfies requirement (iii.i), it follows that there is exactly one $\mathsf{post}_{a,i}$ in Post_a such that $M_s^{\mathsf{unrav}}, s'' \vDash \mathsf{post}_{a,i}$. Since $V'(s'') = V(s'')$, we have that for only this $\mathsf{post}_{a,i}$, $M_s^\varphi, s'' \vDash \mathsf{post}_{a,i}$. In the case that $s'' = s_{a,i}$ for some $i \in [1, |\mathsf{Post}_a|]$, we have that $V'(s_{a,i})$ is constructed precisely so that $M_s^\varphi, s_{a,i} \vDash \mathsf{post}_{a,i}$. Since the set of postconditions for any action must be mutually inconsistent, it follows that this is the only $\mathsf{post}_{a,i}$ that $s_{a,i}$ satisfies. In conclusion, given an $s' \in S'$ and $a \in \mathcal{A}$ such that $P'(s', a)\downarrow$ it holds that for all $s'' \in S'$ where $P'(s', a)(s'') > 0$, there is exactly one $\mathsf{post}_{a,i}$ in Post_a such that $M_s^\varphi, s'' \vDash \mathsf{post}_{a,i}$. Thus M_s^φ satisfies requirement (iii.i).

For the second part of (iii), we choose an $s' \in S'$ and $a \in \mathcal{A}$ such that $P'(s', a)\downarrow$ and an arbitrary $\mathsf{post}_{a,i} \in \mathsf{Post}_a$. In case $s' \in S^{\mathsf{reg}}$, we have that $P(s', a)\downarrow$. Since M_s^{unrav} satisfies requirement (iii.ii), it follows that there is exactly one s'' such that $P(s', a)(s'') > 0$ and $M_s^{\mathsf{unrav}}, s'' \vDash \mathsf{post}_{a,i}$. If this $s'' \in S'$, we have that $P'(s', a)(s'') = P(s', a)(s'') > 0$. Else, we have that $P'(s', a)(s_{a,i}) > 0$. In case $s' \in S^{\mathsf{add}}$, we also have that $P'(s', a)(s_{a,i}) > 0$. From the definition of V it follows that $M_s^\varphi, s_{a,i} \vDash \mathsf{post}_{a,i}$. In conclusion, given an $s' \in S'$ and $a \in \mathcal{A}$ such that $P'(s', a)\downarrow$ it holds that for all $\mathsf{post}_{a,i}$ in Post_a there exists exactly one s'' such that $P(s', a)(s'') > 0$ and $M, s'' \vDash \mathsf{post}_{a,i}$. Thus M_s^φ satisfies requirement (iii.ii).

Since we have that M_s^φ satisfies requirements (i), (ii), and (iii) from Definition 5, we conclude M_s^φ is a well-defined MDP. □

Next, we prove that the bounded MDP M_s^φ satisfies the state formula φ in the starting state s iff the unravelled MDP M_s^{unrav} (and by extension, the original MDP M) satisfies φ in state s. To prove this, we first define the concept of a bounded policy, show that such a policy is well-defined, and show that there exists a probability and reward-preserving bijection between the set of paths of the two policies.

▶ **Definition 20.** Take an MDP M, a state s in M, and a state formula φ, writing $M_s^{\mathsf{unrav}} = \langle S, V, P, R \rangle$

and $M_s^\varphi = \langle S', V', P', R' \rangle$. For a state $\mathbf{h} \in S \cap S'$ and a k-step policy $\pi^{\mathbf{h}}$ in M_s^{unrav}, the k-step policy $\pi^{\varphi,\mathbf{h}}$ in M_s^φ is defined as

$$\pi^{\varphi,\mathbf{h}}(\mathbf{h}_1\mathbf{h}_2\dots\mathbf{h}_n) = \begin{cases} \pi^{\mathbf{h}}(\mathbf{h}_1\mathbf{h}_2\dots\mathbf{h}_n) & \text{for} \quad \{\mathbf{h}_1,\mathbf{h}_2,\dots,\mathbf{h}_n\} \subseteq S \\ \text{arbitrary } a \in \mathcal{A} \text{ s.t. } M_s^\varphi, \mathbf{h}_n \vDash \mathsf{pre}_a & \text{for} \quad \{\mathbf{h}_1,\mathbf{h}_2,\dots,\mathbf{h}_n\} \nsubseteq S \end{cases} \tag{5.31}$$

For a state $\mathbf{h} \in S \cap S'$ and a k-step policy $\pi^{\varphi,\mathbf{h}}$ in M_s^φ, the k-step policy $\pi^{\mathbf{h}}$ in M_s^{unrav} is defined as

$$\pi^{\mathbf{h}}(\mathbf{h}_1\mathbf{h}_2\dots\mathbf{h}_n) = \begin{cases} \pi^{\varphi,\mathbf{h}}(\mathbf{h}_1\mathbf{h}_2\dots\mathbf{h}_n) & \text{for} \quad \{\mathbf{h}_1,\mathbf{h}_2,\dots,\mathbf{h}_n\} \subseteq S' \\ \text{arbitrary } a \in \mathcal{A} \text{ s.t. } M_s^{\mathsf{unrav}}, \mathbf{h}_n \vDash \mathsf{pre}_a & \text{for} \quad \{\mathbf{h}_1,\mathbf{h}_2,\dots,\mathbf{h}_n\} \nsubseteq S' \end{cases} \tag{5.32}$$

Note that it is always possible to select an arbitrary a as described in Equations (5.31) and (5.32), since by Definition 5 we have that for any state s in an MDP M there must exist an action a such that $M, s \vDash \mathsf{pre}_a$. Combining this with the fact that $P'(s,a)(s')\downarrow$ iff $P(s,a)(s')$ for $s, s' \in S \cap S'$ (by Definition 19), it follows that Definition 20 is well-defined.

To aid us in the next proof, we introduce the concept of *state depth*. Informally, this gives us the depth of the path that a state represents after unravelling: e.g., a state representing the path $s_A \cdot \mathsf{up} \cdot s_B$ has a state depth of 1. In the bounded MDP, not all states represent a path in the original MDP anymore: we also have states representing a specific postcondition. For such states, we choose to define the state depth as $\mathsf{pd}(\varphi) + 1$.

▶ **Definition 21.** Given an MDP M, a state s in M, a state formula φ, and a state $\mathbf{h}$ in M_s^{unrav} or M_s^φ, the *state depth* of $\mathbf{h}$ is

$$\mathsf{sd}(\mathbf{h}) = \begin{cases} n & \text{for} \quad \mathbf{h} = s_1 a_1 s_2 \dots a_n s_{n+1} \\ \mathsf{pd}(\varphi) + 1 & \text{for} \quad \mathbf{h} = s_{a,i} \text{ for some } a \in \mathcal{A} \text{ and } i \in [1, \mathsf{Post}_a] \end{cases} \tag{5.33}$$

▶ **Lemma 9.** *Take an MDP M, a state s in M, a state formula φ, integers $k, l, m \geq 0$, a path formula γ such that all state sub-formulas ψ in γ have $\mathsf{pd}(\psi) \leq m$, a state $\mathbf{h}$ in M_s^{unrav} with $\mathsf{sd}(\mathbf{h}) = l$ and a k-step policy $\pi^{\mathbf{h}}$ in M_s^{unrav}. Take set $B^\gamma = \{\mathbf{h}_1 a_1 \mathbf{h}_2 \dots a_k \mathbf{h}_{k+1} \in \mathit{Paths}(\pi^{\mathbf{h}}) \mid P(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\mathsf{unrav}}, \lambda \vDash \gamma\}$ and $C^\gamma = \{\mathbf{h}_1 a_1 \mathbf{h}_2 \dots a_k \mathbf{h}_{k+1} \in \mathit{Paths}(\pi^{\varphi,\mathbf{h}}) \mid P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^\varphi, \lambda \vDash \gamma\}$, i.e the paths from $\pi^{\mathbf{h}}$ and $\pi^{\varphi,\mathbf{h}}$ respectively that have a non-zero probability and satisfy γ. Suppose we have that*

(i) $k + l + m \leq \mathsf{pd}(\varphi)$, and

(ii) $M_s^\varphi, \mathbf{h}' \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \psi$ for all state sub-formulas ψ in γ and states $\mathbf{h}'$ in M_s^φ such that $\mathsf{sd}(\mathbf{h}') \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$.

Then, it follows that the function $\tau : B^\gamma \to C^\gamma$ defined as

$$\tau(\lambda) = \lambda, \tag{5.34}$$

is a probability and reward-preserving bijection, i.e.:

$$\mu(\lambda) = \mu(\tau(\lambda))^3 \text{ and} \tag{5.35}$$

$$R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) = R'(\tau(\lambda)_{s_i}, \tau(\lambda)_{a_i}, \tau(\lambda)_{s_{i+1}}) \text{ for all } 0 < i \leq k. \tag{5.36}$$

Proof. By Definition 16, we have that $P(\mathbf{h}', a)(\mathbf{h}'') = 0$ whenever there does not exist a state s' in M such that $\mathbf{h}'' = \mathbf{h}' a s'$. Thus, for all $\mathbf{h}_1 a_1 \mathbf{h}_2 \dots a_k \mathbf{h}_{k+1} \in B^\gamma$ and $1 \leq i \leq k$, we have that $\mathbf{h}_{i+1} = \mathbf{h}_i a_i s'$ for some s' in M. Since all paths B^γ start in $\mathbf{h}$, it follows that for all $\mathbf{h}_1 a_1 \mathbf{h}_2 \dots a_k \mathbf{h}_{k+1} \in B^\gamma$ and $1 \leq i \leq k$, we have

$$\begin{aligned} \mathsf{sd}(\mathbf{h}_{i+1}) &= \mathsf{sd}(\mathbf{h}_i) + 1 && (5.37) \\ &= \mathsf{sd}(\mathbf{h}) + i && (\text{since } \mathbf{h}_1 = \mathbf{h}) \\ &\leq l + k && (\text{since } \mathsf{sd}(\mathbf{h}) = l \text{ and } i \leq k) \\ &\leq \mathsf{pd}(\varphi) && (\text{since } k + l + m \leq \mathsf{pd}(\varphi)) \end{aligned}$$

[3] Remember from Definition 11 that we drop the superscripts and subscripts from $\mu_\pi^{M,s}$ when the variables are clear from context.

Since by Definition 19, $P'(\mathbf{h}', a)(\mathbf{h}'') = P(\mathbf{h}', a)(\mathbf{h}'')$ and $R'(\mathbf{h}', a, \mathbf{h}'') = R(\mathbf{h}', a, \mathbf{h}'')$ whenever $\mathsf{sd}(\mathbf{h}') \leq \mathsf{pd}(\varphi)$ and $\mathsf{sd}(\mathbf{h}'') \leq \mathsf{pd}(\varphi)$, it then follows immediately that for all $\lambda \in B^\gamma$:

$$\mu(\lambda) = \mu(\tau(\lambda)) \text{ and} \tag{5.38}$$

$$R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) = R'(\tau(\lambda)_{s_i}, \tau(\lambda)_{a_i}, \tau(\lambda)_{s_{i+1}}) \text{ for all } 0 < i \leq k. \tag{5.39}$$

Now that we have shown that τ is probability and reward preserving, all that is left is to show that this function is a bijection between B^γ and C^γ. We do this by showing that in general for any $\lambda \in B^\gamma$, $\tau(\lambda) \in C^\gamma$ and that for any $\lambda' \in C^\gamma$, there is some $\lambda \in B^\gamma$ such that $\lambda' = \tau(\lambda)$. We start with the former.

Choose an arbitrary path $\lambda \in B^\gamma$. Since $\lambda \in B^\gamma$, it satisfies γ, which we assume is of the normal form

$$\gamma = \bigwedge_{v=1}^{e} \bigvee_{w=1}^{f} (\mathsf{X}^{t_{v,w}} \varphi_{v,w}), \text{ where } \varphi_{v,w} = \begin{cases} \text{a state formula } \psi, \\ \mathsf{do}_a \text{ or } \neg\mathsf{do}_a, \text{ or} \\ \mathsf{C}^u_{\bowtie r}. \end{cases} \tag{5.40}$$

Given a v, we know there exists a w such that $M_s^{\mathsf{unrav}}, \lambda \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$. We have three scenarios.

- If $\varphi_{v,w}$ is a state formula ψ, we know that $M_s^{\mathsf{unrav}}, \lambda_{t_{v,w}+1} \vDash \varphi_{v,w}$. Since $t_{v,w} \leq k$, and $\mathsf{pd}(\psi) \leq m$, and $\mathsf{sd}(\lambda_{t_{v,w}+1}) = t_{v,w} + l$ (by Equation (5.37)), it follows that

  $$t_{v,w} + l + \mathsf{pd}(\psi) \leq k + l + m \leq \mathsf{pd}(\varphi). \tag{5.41}$$

  Thus, we have that

  $$t_{v,w} + l = \mathsf{sd}(\lambda_{t_{v,w}+1}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi). \tag{5.42}$$

  It then follows from $M_s^{\mathsf{unrav}}, \lambda_{t_{v,w}+1} \vDash \psi$ that $M_s^{\varphi}, \lambda_{t_{v,w}+1} \vDash \psi$, which we can work back to $M_s^{\varphi}, \tau(\lambda) \vDash \mathsf{X}^{t_{v,w}} \psi$.

- If $\varphi_{v,w}$ is do_a, then we know that a is the $t_{v,w}+1$-th action of λ. By definition, a is the $t_{v,w}+1$-th action in $\tau(\lambda)$ if and only if it is the $t_{v,w}+1$-th action in λ. Thus, $M_s^{\varphi}, \tau(\lambda) \vDash \mathsf{X}^{t_{v,w}} \mathsf{do}_a$ follows immediately. The same reasoning holds if $\varphi_{v,w}$ is $\neg\mathsf{do}_a$.

- If $\varphi_{v,w}$ is $\mathsf{C}^u_{\bowtie r}$, we know that $\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) \bowtie r$. By Equation (5.39), we know that the following equality holds.

  $$\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda_{s_i}, \lambda_{a_i}, \lambda_{s_{i+1}}) = \sum_{i=t_{v,w}+1}^{t_{v,w}+u} R'(\tau(\lambda)_{s_i}, \tau(\lambda)_{a_i}, \tau(\lambda)_{s_{i+1}}) \bowtie r \tag{5.43}$$

  From this we can conclude that $M_s^{\varphi}, \tau(\lambda) \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$ if $\varphi_{v,w} = \mathsf{C}^u_{\bowtie r}$.

These three cases show that for all $1 \leq v \leq e$ there exists some $1 \leq w \leq f$ such that $M_s^{\varphi}, \tau(\lambda) \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$. Thus, $\lambda \in B^\gamma$ implies $\tau(\lambda) \in C^\gamma$.

Now, choose some $\lambda' = \mathbf{h}_1 a_1 \mathbf{h}_2 \ldots a_k \mathbf{h}_{k+1} \in C^\gamma$. For any $1 \leq i \leq k$, it holds that $P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0$. Thus, by Definition 19, it must be that if $\mathsf{sd}(\mathbf{h}_{i+1}) \leq \mathsf{pd}(\varphi)$, then $\mathsf{sd}(\mathbf{h}_i) \leq \mathsf{pd}(\varphi)$ as well. From this it follows that $P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) = P(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1})$, which means $\mathsf{sd}(\mathbf{h}_{i+1}) = l + i$.

Note that it is not possible that $\mathsf{sd}(\mathbf{h}_{i+1}) = \mathsf{pd}(\varphi)+1$ for any i. We show this with a proof by contradiction. Suppose $\mathsf{sd}(\mathbf{h}_{i+1}) = \mathsf{pd}(\varphi)+1$ for some i. Since $P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0$, it must be that either $\mathsf{sd}(\mathbf{h}_i) = \mathsf{pd}(\varphi)$ or $\mathsf{sd}(\mathbf{h}_i) = \mathsf{pd}(\varphi) + 1$ as well. We consider both cases:

- If $\mathsf{sd}(\mathbf{h}_i) = \mathsf{pd}(\varphi)$, then it must be that $\mathsf{sd}(\mathbf{h}_{i-1}) \leq \mathsf{pd}(\varphi)$. From this it follows that $P'(\mathbf{h}_{i-1}, a_{i-1})(\mathbf{h}_i) = P(\mathbf{h}_{i-1}, a_{i-1})(\mathbf{h}_i)$, which means $\mathsf{sd}(\mathbf{h}_i) = l + (i-1) \neq \mathsf{pd}(\varphi)$. Thus, we have a contradiction.

- If $\mathsf{sd}(\mathbf{h}_i) = \mathsf{pd}(\varphi) + 1$, we have that $\mathsf{sd}(\mathbf{h}_j) = \mathsf{pd}(\varphi) + 1$ must hold for all $1 \leq j \leq i$ (since by our previous case it is not possible for any preceding state to have a state depth of $\mathsf{pd}(\varphi)$). However, we have that $\mathbf{h}_1 = \mathbf{h}$, for which we have $\mathsf{sd}(\mathbf{h}) = l < \mathsf{pd}(\varphi) + 1$. Thus, we again arrive at a contradiction.

Since both cases arrive at a contradiction, we can conclude that for any $1 \leq i \leq k$, we have that $\mathsf{sd}(\mathbf{h}_{i+1}) \leq \mathsf{pd}(\varphi)$, from which it follows that $\mathsf{sd}(\mathbf{h}_{i+1}) = l+i \leq \mathsf{pd}(\varphi)$. Thus, it follows from Definition 19 that $\mathbf{h}_{i+1} \in S$ and that $\tau(\lambda') = \lambda'$ is a path in M_s^{unrav}. We now show that $\lambda' \in B^\gamma$ (with a proof nearly identical to the other direction), to complete the proof.

Since $\lambda' \in C^\gamma$, it satisfies γ, which we assume is of the normal form given in Equation (5.40). Given a v, we know there exists a w such that $M_s^\varphi, \lambda' \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$. We have three scenarios.

- If $\varphi_{v,w}$ is a state formula ψ, we know that $M_s^\varphi, \lambda'_{t_{v,w}+1} \vDash \varphi_{v,w}$. Since $t_{v,w} \leq k$, and $\mathsf{pd}(\psi) \leq m$, and $\mathsf{sd}(\lambda'_{t_{v,w}+1}) = t_{v,w} + l$ (by Equation (5.37)), it follows that

$$t_{v,w} + l + \mathsf{pd}(\psi) \leq k + l + m \leq \mathsf{pd}(\varphi). \tag{5.44}$$

  Thus, we have that

$$t_{v,w} + l = \mathsf{sd}(\lambda'_{t_{v,w}+1}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi). \tag{5.45}$$

  It then follows from $M_s^\varphi, \lambda'_{t_{v,w}+1} \vDash \psi$ that $M_s^{\mathsf{unrav}}, \lambda'_{t_{v,w}+1} \vDash \psi$, which we can work back to $M_s^{\mathsf{unrav}}, \lambda' \vDash \mathsf{X}^{t_{v,w}} \psi$.

- If $\varphi_{v,w}$ is do_a, then we know that a is the $t_{v,w}+1$-th action of λ'. By definition, a is the $t_{v,w}+1$-th action in $\tau(\lambda')$ if and only if it is the $t_{v,w}+1$-th action in λ'. Thus, $M_s^{\mathsf{unrav}}, \lambda' \vDash \mathsf{X}^{t_{v,w}} \mathsf{do}_a$ follows immediately. The same reasoning holds if $\varphi_{v,w}$ is $\neg \mathsf{do}_a$.

- If $\varphi_{v,w}$ is $\mathsf{C}^u_{\bowtie r}$, we know that $\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R'(\lambda'_{s_i}, \lambda'_{a_i}, \lambda'_{s_{i+1}}) \bowtie r$. By Equation (5.39), we know that the following equality holds.

$$\sum_{i=t_{v,w}+1}^{t_{v,w}+u} R(\lambda'_{s_i}, \lambda'_{a_i}, \lambda'_{s_{i+1}}) = \sum_{i=t_{v,w}+1}^{t_{v,w}+u} R'(\tau(\lambda')_{s_i}, \tau(\lambda')_{a_i}, \tau(\lambda')_{s_{i+1}}) \bowtie r \tag{5.46}$$

  From this we can conclude that $M_s^{\mathsf{unrav}}, \lambda' \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$ if $\varphi_{v,w} = \mathsf{C}^u_{\bowtie r}$.

These three cases show that for all $1 \leq v \leq e$ there exists some $1 \leq w \leq f$ such that $M_s^{\mathsf{unrav}}, \lambda' \vDash \mathsf{X}^{t_{v,w}} \varphi_{v,w}$. Thus, for our arbitrary $\lambda' \in C^\gamma$, have that $\lambda' \in B^\gamma$. □

▶ **Theorem 10 (Equivalence of M_s^φ to M_s^{unrav}).** *Given an MDP M, a state s in M, and a state formula φ, we have that $M_s^{\mathsf{unrav}}, s \vDash \varphi$ iff $M_s^\varphi, s \vDash \varphi$.*

Proof. Let M be an MDP, s an arbitrary state in M, and φ a state formula.

We introduce a predicate $\mathcal{P}_{\mathsf{equiv}}(\psi)$ as shorthand for "for any state $\mathbf{h}$ in M_s^φ such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$, it holds that $M_s^\varphi, \mathbf{h} \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \psi$." We show by induction on state formula ψ that $\mathcal{P}_{\mathsf{equiv}}(\psi)$ holds, by proving the following four statements:

1. $\mathcal{P}_{\mathsf{equiv}}(p)$ for all $p \in \mathsf{Prop}$;
2. if $\mathcal{P}_{\mathsf{equiv}}(\psi)$, then $\mathcal{P}_{\mathsf{equiv}}(\neg\psi)$;
3. if $\mathcal{P}_{\mathsf{equiv}}(\psi)$ and $\mathcal{P}_{\mathsf{equiv}}(\psi')$, then $\mathcal{P}_{\mathsf{equiv}}(\psi \wedge \psi')$; and
4. if $\mathcal{P}_{\mathsf{equiv}}(\psi)$ for all state subformulas ψ in policy formula ξ, then $\mathcal{P}_{\mathsf{equiv}}(\Diamond^k(\xi))$.

First, we prove our base case, $\mathcal{P}_{\mathsf{equiv}}(p)$ for all $p \in \mathsf{Prop}$. We choose an arbitrary state $\mathbf{h}$ from M_s^φ such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi) = \mathsf{pd}(\varphi)$. By Definition 19, we then have that $V(\mathbf{h}) = V'(\mathbf{h})$. Thus, $p \in V(\mathbf{h})$ iff $p \in V'(\mathbf{h})$, which directly lets us conclude that $\mathcal{P}_{\mathsf{equiv}}(p)$ for all $p \in \mathsf{Prop}$.

Second, we show that if $\mathcal{P}_{\mathsf{equiv}}(\psi)$, then $\mathcal{P}_{\mathsf{equiv}}(\neg\psi)$. We choose an arbitrary state formula ψ and assume that $\mathcal{P}_{\mathsf{equiv}}(\psi)$. Let $\mathbf{h}$ be an arbitrary state from M_s^φ such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi) = \mathsf{pd}(\varphi) - \mathsf{pd}(\neg\psi)$. By $\mathcal{P}_{\mathsf{equiv}}(\psi)$, it then follows that $M_s^\varphi, \mathbf{h} \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \psi$. By the basic properties of the negation operator, it follows that $M_s^\varphi, \mathbf{h} \vDash \neg\psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \neg\psi$. Thus, we have that $\mathcal{P}_{\mathsf{equiv}}(\neg\psi)$ holds.

Third, we show that if $\mathcal{P}_{\mathsf{equiv}}(\psi)$ and $\mathcal{P}_{\mathsf{equiv}}(\psi')$, then $\mathcal{P}_{\mathsf{equiv}}(\psi \wedge \psi')$. We choose two arbitrary state formulas ψ and ψ' and assume that $\mathcal{P}_{\mathsf{equiv}}(\psi)$ and $\mathcal{P}_{\mathsf{equiv}}(\psi')$ hold. Let $\mathbf{h}$ be an arbitrary state from M_s^φ such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$ and $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi')$. From the fact that $M_s^\varphi, \mathbf{h} \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \psi$ and $M_s^\varphi, \mathbf{h} \vDash \psi'$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \psi'$, it follows that $M_s^\varphi, \mathbf{h} \vDash \psi \wedge \psi'$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \psi \wedge \psi'$. Since $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \max\{\mathsf{pd}(\psi), \mathsf{pd}(\psi')\} = \mathsf{pd}(\varphi) - \mathsf{pd}(\psi \wedge \psi')$, we can conclude that $\mathcal{P}_{\mathsf{equiv}}(\psi \wedge \psi')$.

Fourth, we show that if $\mathcal{P}_{\mathsf{equiv}}(\psi)$ for all state subformulas ψ in policy formula ξ, then $\mathcal{P}_{\mathsf{equiv}}(\Diamond^k(\xi))$. We choose an arbitrary integer $k \geq 0$ and policy formula ξ, such that $\Diamond^k(\xi)$ is a syntactically valid state formula. We assume that our policy formula ξ is in normal form:

$$\xi = \bigwedge_{m=1}^{c} \bigvee_{q=1}^{d} \xi_{m,q}, \text{ where } \xi_{m,q} = \begin{cases} \mathsf{P}_{\bowtie c}(\gamma), \text{ or} \\ \mathsf{E}^{l,u}_{\bowtie r}. \end{cases} \tag{5.47}$$

We assume that for all state sub-formulas ψ that occur in ξ, it holds that $\mathcal{P}_{\mathsf{equiv}}(\psi)$. We show that it follows from this induction hypothesis that $\mathcal{P}_{\mathsf{equiv}}(\Diamond^k(\xi))$ holds, in two parts.

1. First, left to right: "for any state $\mathbf{h}$ in M^{φ}_s such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\Diamond^k(\xi))$, it holds that if $M^{\varphi}_s, \mathbf{h} \vDash \Diamond^k(\xi)$, then $M^{\mathsf{unrav}}_s, \mathbf{h} \vDash \Diamond^k(\xi)$."

   To prove this, we let $\mathbf{h}$ be an arbitrary state such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\Diamond^k(\xi)) = \mathsf{pd}(\varphi) - (k + \max\{\mathsf{pd}(\psi) \mid \text{ state formula } \psi \text{ appears in } \xi\})$. Suppose $M^{\varphi}_s, \mathbf{h} \vDash \Diamond^k(\xi)$, which means that there exists some k-step policy $\pi^{\varphi,\mathbf{h}}$ in M^{φ}_s, such that $M^{\varphi}_s, \pi^{\varphi,\mathbf{h}} \vDash \xi$. By Definition 20, we have that $\pi^{\mathbf{h}}$ is a well-defined policy in M^{unrav}_s. Note that if we can show that $M^{\mathsf{unrav}}_s, \pi^{\mathbf{h}} \vDash \xi$, we have proven the left-to-right direction of the inductive step.

   Since ξ is written in normal form, we know that given an m, there exists a q such that $M^{\varphi}_s, \pi^{\varphi,\mathbf{h}} \vDash \xi_{m,q}$. We consider two scenarios.

   - Suppose $\xi_{m,q} = \mathsf{E}^{l,u}_{\bowtie r}$. From our assumption that $M^{\varphi}_s, \pi^{\varphi,\mathbf{h}} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, it follows that the expected reward of the policy from step l up to and including u, is $\bowtie r$:

   $$\sum_{\lambda \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R'(s_i, a_i, s_{i+1}) \right) \right) \bowtie r. \tag{5.48}$$

     Let $\gamma = \top$, and $B^{\gamma} = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid P(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M^{\mathsf{unrav}}_s, \lambda \vDash \gamma\}$ and $C^{\gamma} = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M^{\varphi}_s, \lambda \vDash \gamma\}$. We use the function $\tau : B^{\gamma} \to C^{\gamma}$ defined as $\tau(\lambda) = \lambda$ from Lemma 9. We have that the function τ is a reward and probability preserving bijection, since the following two conditions hold:

     (i') $k + \mathsf{sd}(\mathbf{h}) + \max\{\mathsf{pd}(\psi) \mid \text{ state formulas } \psi \text{ occurring in } \gamma\} = k + \mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi)$, and

     (ii') for all ψ occurring in γ and for any states $\mathbf{h}'$ in M^{φ}_s such that $\mathsf{sd}(\mathbf{h}') \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$, it holds that $M^{\varphi}_s, \mathbf{h}' \vDash \psi$ iff $M^{\mathsf{unrav}}_s, \mathbf{h}' \vDash \psi$ (vacuously true since there are no state formulas in $\top$).

     The following equivalence follows from the bijection:

     $$\begin{aligned} &\sum_{\lambda \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R'(s_i, a_i, s_{i+1}) \right) \right) \\ &= \sum_{\lambda \in C^{\gamma}} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R'(s_i, a_i, s_{i+1}) \right) \right) \\ &= \sum_{\lambda \in B^{\gamma}} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R(s_i, a_i, s_{i+1}) \right) \right) \\ &= \sum_{\lambda \in \mathsf{Paths}(\pi^{\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} \left(R(s_i, a_i, s_{i+1}) \right) \right) \bowtie r, \end{aligned} \tag{5.49}$$

     from which it immediately follows that $M^{\mathsf{unrav}}_s, \pi^{\mathbf{h}} \vDash \mathsf{E}^{l,u}_{\bowtie r}$.

   - Suppose $\xi_{m,q} = \mathsf{P}_{\bowtie c}(\gamma)$. By our assumption that $M^{\varphi}_s, \pi^{\varphi,\mathbf{h}} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, we have that

   $$\mu(\{\lambda \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid M^{\varphi}_s, \lambda \vDash \gamma\}) \bowtie c. \tag{5.50}$$

Let $B^\gamma = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid P(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\mathsf{unrav}}, \lambda \vDash \gamma\}$ and $C^\gamma = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\varphi}, \lambda \vDash \gamma\}$. We use the function $\tau : B^\gamma \to C^\gamma$ defined as $\tau(\lambda) = \lambda$ from Lemma 9. We have that the function $\tau(\lambda) = \lambda$ is a reward and probability preserving bijection, since the following two conditions hold:

(i') since $\mathbf{h}$ was chosen such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\Diamond^k(\xi)) = \mathsf{pd}(\varphi) - (k + \max\{\mathsf{pd}(\psi) \mid$ state formula ψ appears in $\xi\})$ it follows that $k + \mathsf{sd}(\mathbf{h}) + \max\{\mathsf{pd}(\psi) \mid$ state formulas ψ occurring in $\gamma\} \leq \mathsf{pd}(\varphi)$, and

(ii') from our induction hypothesis it follows that for all ψ occurring in γ and for any states $\mathbf{h}'$ in M_s^{φ} such that $\mathsf{sd}(\mathbf{h}') \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$, it holds that $M_s^{\varphi}, \mathbf{h}' \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \psi$.

The following equivalence follows:

$$\begin{aligned} \mu(\{\lambda \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid M_s^{\varphi}, \lambda \vDash \gamma\}) &= \mu(C^\gamma) = \mu(B^\gamma) \\ &= \mu(\{\lambda \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda \vDash \gamma\}) \bowtie c, \end{aligned} \tag{5.51}$$

from which it immediately follows that $M_s^{\mathsf{unrav}}, \pi^{\mathbf{h}} \vDash \mathsf{P}_{\bowtie c}(\gamma)$.

Now, we have shown for all ξ that $M_s^{\varphi}, \mathbf{h} \vDash \Diamond^k(\xi)$ implies $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$, if we assume our induction hypothesis. This concludes the left-to-right part of our inductive step.

2. Second, right to left: "for any state $\mathbf{h}$ in M_s^{φ} such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\Diamond^k(\xi))$, it holds that if $M_s^{\mathsf{unrav}}, \pi^{\mathbf{h}} \vDash \xi$, then $M_s^{\varphi}, \pi^{\varphi,\mathbf{h}} \vDash \xi$." This proof follows the same reasoning as the left-to-right direction.

   To prove this, we let $\mathbf{h}$ be an arbitrary state such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\Diamond^k(\xi)) = \mathsf{pd}(\varphi) - (k + \max\{\mathsf{pd}(\psi) \mid$ state formula ψ appears in $\xi\})$. Suppose $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$, which means that there exists some k-step policy $\pi^{\mathbf{h}}$ in M_s^{unrav}, such that $M_s^{\mathsf{unrav}}, \pi^{\mathbf{h}} \vDash \xi$. By Definition 20, we have that $\pi^{\varphi,\mathbf{h}}$ is a well-defined policy in M_s^{φ}. Note that if we can show that $M_s^{\varphi}, \pi^{\varphi,\mathbf{h}} \vDash \xi$, we have proven the left-to-right direction of the inductive step.

   Since ξ is written in normal form, we know that given an m, there exists a q such that $M_s^{\mathsf{unrav}}, \pi^{\mathbf{h}} \vDash \xi_{m,q}$. We consider two scenarios.

   - Suppose $\xi_{m,q} = \mathsf{E}^{l,u}_{\bowtie r}$. From our assumption that $M_s^{\mathsf{unrav}}, \pi^{\mathbf{h}} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, it follows that the expected reward of the policy from step l up to and including u, is $\bowtie r$:

$$\sum_{\lambda \in \mathsf{Paths}(\pi^{\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} (R(s_i, a_i, s_{i+1})) \right) \bowtie r. \tag{5.52}$$

     Let $\gamma = \top$, and $B^\gamma = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid P(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\mathsf{unrav}}, \lambda \vDash \gamma\}$ and $C^\gamma = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\varphi}, \lambda \vDash \gamma\}$. We use the function $\tau : B^\gamma \to C^\gamma$ defined as $\tau(\lambda) = \lambda$ from Lemma 9. We have that the function τ is a reward and probability preserving bijection, since the following two conditions hold:

     (i') $k + \mathsf{sd}(\mathbf{h}) + \max\{\mathsf{pd}(\psi) \mid$ state formulas ψ occurring in $\gamma\} = k + \mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi)$, and

     (ii') for all ψ occurring in γ and for any states $\mathbf{h}'$ in M_s^{φ} such that $\mathsf{sd}(\mathbf{h}') \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$, it holds that $M_s^{\varphi}, \mathbf{h}' \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \psi$ (vacuously true since there are no state formulas in $\top$).

The following equivalence follows from the bijection:

$$\begin{aligned}
&\sum_{\lambda \in \mathsf{Paths}(\pi^{\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} (R(s_i, a_i, s_{i+1})) \right) \\
&= \sum_{\lambda \in B^{\gamma}} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} (R(s_i, a_i, s_{i+1})) \right) \\
&= \sum_{\lambda \in C^{\gamma}} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} (R'(s_i, a_i, s_{i+1})) \right) \\
&= \sum_{\lambda \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}})} \left(\mu(\lambda) \cdot \sum_{i=l}^{u} (R'(s_i, a_i, s_{i+1})) \right) \bowtie r,
\end{aligned} \tag{5.53}$$

from which it immediately follows that $M_s^{\varphi}, \pi^{\varphi,\mathbf{h}} \vDash \mathsf{E}^{l,u}_{\bowtie r}$.

- Suppose $\xi_{m,q} = \mathsf{P}_{\bowtie c}(\gamma)$. By our assumption that $M_s^{\mathsf{unrav}}, \pi^{\mathbf{h}} \vDash \xi_{m,q}$ for our chosen m and q, and the semantics of rPLBP, we have that

$$\mu(\{\lambda \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid M_s^{\varphi}, \lambda \vDash \gamma\}) \bowtie c. \tag{5.54}$$

Let $B^{\gamma} = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid P(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\mathsf{unrav}}, \lambda \vDash \gamma\}$ and $C^{\gamma} = \{\mathbf{h}_1 a_1 \dots a_k \mathbf{h}_{k+1} \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid P'(\mathbf{h}_i, a_i)(\mathbf{h}_{i+1}) > 0 \text{ for all } 1 \leq i \leq k \text{ and } M_s^{\varphi}, \lambda \vDash \gamma\}$. We use the function $\tau : B^{\gamma} \to C^{\gamma}$ defined as $\tau(\lambda) = \lambda$ from Lemma 9. We have that the function $\tau(\lambda) = \lambda$ is a reward and probability preserving bijection, since the following two conditions hold:

(i') since $\mathbf{h}$ was chosen such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\Diamond^k(\xi)) = \mathsf{pd}(\varphi) - (k + \max\{\mathsf{pd}(\psi) \mid$ state formula ψ appears in $\xi\})$ it follows that $k + \mathsf{sd}(\mathbf{h}) + \max\{\mathsf{pd}(\psi) \mid$ state formulas ψ occurring in $\gamma\} \leq \mathsf{pd}(\varphi)$, and

(ii') from our induction hypothesis it follows that for all ψ occurring in γ and for any states $\mathbf{h}'$ in M_s^{φ} such that $\mathsf{sd}(\mathbf{h}') \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$, it holds that $M_s^{\varphi}, \mathbf{h}' \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h}' \vDash \psi$.

The following equivalence follows:

$$\begin{aligned}
\mu(\{\lambda \in \mathsf{Paths}(\pi^{\mathbf{h}}) \mid M_s^{\mathsf{unrav}}, \lambda \vDash \gamma\}) &= \mu(B^{\gamma}) = \mu(C^{\gamma}) \\
&= \mu(\{\lambda \in \mathsf{Paths}(\pi^{\varphi,\mathbf{h}}) \mid M_s^{\varphi}, \lambda \vDash \gamma\}) \bowtie c,
\end{aligned} \tag{5.55}$$

from which it immediately follows that $M_s^{\varphi}, \pi^{\varphi,\mathbf{h}} \vDash \mathsf{P}_{\bowtie c}(\gamma)$.

Now, we have shown for all ξ that $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \Diamond^k(\xi)$ implies $M_s^{\varphi}, \mathbf{h} \vDash \Diamond^k(\xi)$, if we assume our induction hypothesis. This concludes the left-to-right part of our inductive step.

By combining both directions, we conclude that if $\mathcal{P}(\psi)$ for all state subformulas ψ in policy formula ξ, then $\mathcal{P}(\Diamond^k(\xi))$. Thus, we have shown by induction that for any state formula ψ and for any state $\mathbf{h}$ in M_s^{φ} such that $\mathsf{sd}(\mathbf{h}) \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\psi)$, it holds that $M_s^{\varphi}, \mathbf{h} \vDash \psi$ iff $M_s^{\mathsf{unrav}}, \mathbf{h} \vDash \psi$. Now if we let $\psi = \varphi$, we have that $\mathsf{sd}(s) = 0 \leq \mathsf{pd}(\varphi) - \mathsf{pd}(\varphi)$, and it follows that $M_s^{\varphi}, s \vDash \varphi$ iff $M_s^{\mathsf{unrav}}, s \vDash \varphi$. □

We now have the knowledge that for any MDP satisfying a state formula, there exists a bounded, unravelled MDP that satisfies the same state formula. We use this to show that a state formula is satisfiable, if and only if it is also satisfied by an MDP of some calculable size. This is more formally referred to as the finite model property.

▶ **Theorem 11 (Finite model property).** *If a formula φ is satisfiable by any model, then it is also satisfiable by an MDP that has most $(\sum_{a \in \mathcal{A}} |\mathsf{Post}_a|)^{\mathsf{pd}(\varphi)} + \sum_{a \in \mathcal{A}} |\mathsf{Post}_a|$ states.*

Proof. Suppose there exists an MDP with a state s such that $M, s \vDash \varphi$. By Lemma 4 and Lemma 7, there exists an unravelled MDP M_s^{unrav}, such that $M_s^{\mathsf{unrav}}, s \vDash \varphi$ (where we treat s as a single-state path $\mathbf{h}$ such that $\mathsf{last}(\mathbf{h}) = s$). Cutting off and patching this unravelled MDP, we can create M_s^{φ} for which it holds that $M_s^{\varphi}, s \vDash \varphi$. Up to and including depth $\mathsf{pd}(\varphi)$, we have that this MDP has a tree structure. Worst case, the agent can execute any action in all states, which means that there are transitions to

$\sum_{a\in\mathcal{A}} |\mathsf{Post}_a|$ subsequent states. This means the branching factor of our 'tree' is at most $\sum_{a\in\mathcal{A}} |\mathsf{Post}_a|$. The depth is $\mathsf{pd}(\varphi)$, since we have cut it off there. Due to our patching, we have also added a new state for all postconditions. Thus our MDP M_s^φ, which satisfies φ, contains at most as many states as there are nodes in a tree with branching factor $\sum_{a\in\mathcal{A}} |\mathsf{Post}_a|$ and depth $\mathsf{pd}(\varphi)$, with the addition of $\sum_{a\in\mathcal{A}} |\mathsf{Post}_a|$ additional states. This is amount is bounded from above by $(\sum_{a\in\mathcal{A}} |\mathsf{Post}_a|)^{\mathsf{pd}(\varphi)} + \sum_{a\in\mathcal{A}} |\mathsf{Post}_a|$. □

5.2 Decidability

The knowledge that there exists an upper bound on the size of the smallest MDP that satisfies a formula φ, and even what that bound is, allows us to take somewhat of a brute-force approach to the satisfiability question: trying all possible configurations of an MDP within this bound and testing which one works. However, beyond this intuition lies the issue that while the MDP is bounded in the number of states, there are still infinite possibilities for how the probabilities and rewards are assigned to transitions.

Once we have a set of states and a valuation of those states, we can decide whether there exists some probability distribution function and a reward structure that completes our MDP in such a way that the formula φ is satisfied by the first state of this model, s_1. Like [Motamed et al., 2023] we translate this problem into a first-order logic (FOL) formula in the signature of real closed fields [Chang and Keisler, 2012] that is decidable in the theory of real closed fields. This process is described in detail in the proof below.

▶ **Theorem 12 (Decidability of satisfiability problem).** *The satisfiability problem for rPLBP can be decided in 2EXPSPACE.*

Proof. Suppose we have a state formula φ and a set of actions $\mathcal{A}$ with preconditions and postconditions. Let n be the combined size of this input. By Theorem 11, we know that we only need to consider MDPs where the number of states is at most $b = (\sum_{a\in\mathcal{A}} |\mathsf{Post}_a|)^{\mathsf{pd}(\varphi)} + \sum_{a\in\mathcal{A}} |\mathsf{Post}_a|$. We iterate over all possible $S = \{s_1, s_2, \ldots, s_k | 1 \leq k \leq b\}$ and valuations V of those S, using only the set of all propositional variables occurring in φ and the preconditions and postconditions of $\mathcal{A}$. In terms of the input n, there are $\mathcal{O}(n^n)$ different S, and for each of those, $\mathcal{O}(2^n)^{n^n}$ different valuations. Thus, we have at most $O(2^{2^{n^2}})$ unique configurations of S and V to consider.

For each set of states S and valuation function V such that $|S| \leq b$ and for each $s \in S$, $s \vDash \mathsf{pre}_a$ for some $a \in \mathcal{A}$, we must then decide whether there exist a P and R such that $\langle S, P, V, R\rangle, s_1 \vDash \varphi$. Note that it is sufficient to consider only the first state, s_1, even when $\langle S, P, V, R\rangle, s_i \vDash \varphi$ for some $i \neq 1$. This is because all possible valuation functions for S are considered, including V' such that $V'(s_i) = V(s_1)$, $V'(s_1) = V(s_i)$, and $V'(s_j) = V(s_j)$ for all $j \neq 1$ and $j \neq i$ (essentially swapping the valuations of s_1 and s_i). Therefore, $\langle S, P, V', R\rangle, s_1 \vDash \varphi$ iff $\langle S, P, V, R\rangle, s_i \vDash \varphi$, meaning we only have to consider the first state of S each time.

To determine whether a P and R exist such that $\langle S, P, V, R\rangle, s_1 \vDash \varphi$, we create two sequences of variables to represent P and R: $\mathbf{p}$ and $\mathbf{r}$ respectively. For each $s, t \in S$ and $a \in \mathcal{A}$, intuitively $p_{s,a,t}$ represents the value of $P(s, a)(t)$ and $r_{s,a,t}$ represents the value of $R(s, a, t)$. We then construct a first-order logic (FOL) formula α_φ where we quantify over these sequences of variables, such that α_{φ,s_1} holds in the theory of real closed fields, iff there exist a well-defined P and R such that $\langle S, P, V, R\rangle, s_1 \vDash \varphi$.

$$\alpha_\varphi = \exists\mathbf{p}\exists\mathbf{r}(\beta(\mathbf{p}) \wedge \delta_{\varphi,s_1}(\mathbf{p}, \mathbf{r})) \tag{5.56}$$

The first term, $\beta(\mathbf{p})$, should be true whenever the probability distribution function is well-defined. The sequence $\mathbf{p}$ must satisfy the following requirements:

1. All values must lie between 0 and 1.
2. The probability of all outcomes of an action in a particular state must add up to 1 if this state satisfies the preconditions.
3. The probability of all outcomes of an action in a particular state must add up to 0 if this state does not satisfy the preconditions.
4. All postconditions must correspond to reachable states.
5. All reachable states must correspond to postconditions.

Therefore, we define $\beta(\mathbf{p})$ as $\bigwedge_{i\in\{1,2,3,4,5\}} \beta_i$, where

$$\begin{aligned}
\beta_1 &= \bigwedge_{s,a,t} (0 \le p_{s,a,t} \wedge p_{s,a,t} \le 1),\\
\beta_2 &= \bigwedge_{s,a \text{ such that } s \vDash \mathsf{pre}_a} \sum_t p_{s,a,t} \approx 1,\\
\beta_3 &= \bigwedge_{s,a \text{ such that } s \nvDash \mathsf{pre}_a} \sum_t p_{s,a,t} \approx 0,\\
\beta_4 &= \bigwedge_{s,a,t} (p_{s,a,t} > 0 \to \rho_{t,a}),\\
\beta_5 &= \bigwedge_{s,a \text{ such that } s \vDash \mathsf{pre}_a} \bigwedge_{\mathsf{post}_{a,i} \in \mathsf{Post}_a} \bigvee_t ((p_{s,a,t} > 0 \wedge \tau_{t,a,i}) \wedge \bigwedge_{t' \neq t} \neg(p_{s,a,t'} > 0 \wedge \tau_{t',a,i})).
\end{aligned} \tag{5.57}$$

The $\approx$ symbol represents equality in the theory of real closed fields. The term $\rho_{t,a}$ holds iff there is a unique $\mathsf{post}_{a,i} \in \mathsf{Post}_a$ with $t \vDash \mathsf{post}_{a,i}$. The term $\tau_{t,a,i}$ holds iff $t \vDash \mathsf{post}_{a,i}$. Whether for a state s and action a it holds that $s \vDash \mathsf{pre}_a$ or $s \vDash \mathsf{post}_{a,i}$, is determined outside of RCF with propositional reasoning.

Note that the reward function has no requirements, and therefore there is no formula necessary to verify the well-definedness of R.

The second term of α_φ, the FOL formula $\delta_{\varphi,s_1}(\mathbf{p},\mathbf{r})$, should be true when the rPLBP state formula φ is satisfied, as long as the formula $\beta(\mathbf{p})$ holds. To this end, we define by simultaneous induction the formulas FOL δ_ψ, s, $\chi_{\xi,\pi,s}$, and $\kappa_{\gamma,\lambda}$, which should hold when $s \vDash \psi$, $\pi^s \vDash \xi$, and $\lambda \vDash \gamma$ hold in rPLBP respectively.

The state formulas are as follows:

$$\begin{aligned}
\delta_{p,s} &= \top \text{ if } p \in V(s), \text{ else } \bot;\\
\delta_{\psi\wedge\zeta,s} &= \delta_{\psi,s} \wedge \delta_{\zeta,s};\\
\delta_{\neg\psi,s} &= \neg\delta_{\psi,s};\\
\delta_{\Diamond^k(\xi),s} &= \bigvee_{k\text{-step } \pi,s} \chi_{\xi,\pi,s};
\end{aligned} \tag{5.58}$$

For the last formula, we use the term $\chi_{\xi,\pi,s}$, which should be true iff $\pi^s \vDash \xi$. We define these formulas inductively as:

$$\begin{aligned}
\chi_{\eta\wedge\omega,\pi,s} &= \chi_{\eta,\pi,s} \wedge \chi_{\omega,\pi,s};\\
\chi_{\neg\eta,\pi,s} &= \neg\chi_{\eta,\pi,s};\\
\chi_{\mathsf{E}^{l,u}_{\bowtie r},\pi,s} &= \sum_{s_1 a_1 \ldots s_k a_k s_{k+1} \in \mathsf{Paths}(\pi^s)} \left(\prod_{i=1}^{k} p_{s_i,a_i,s_{i+1}}\right) \times \sum_{i=l}^{u} r_{s_i,a_i,s_{i+1}} \bowtie r;\\
\chi_{\mathsf{P}_{\bowtie c}(\gamma),\pi,s} &= \bigvee_{X \subseteq \mathsf{Paths}(\pi^s)} \varepsilon_{X,\gamma,s,\pi} \to \eta_{X,\bowtie,c};
\end{aligned} \tag{5.59}$$

The last formula, $\chi_{\mathsf{P}_{\bowtie c}(\gamma),\pi,s}$, contains two new formulas concerning the set X. The first term, $\varepsilon_{X,\gamma,s,\pi}$, should be true when X is exactly the set $\{\lambda \in \mathsf{Paths}(\pi^s) \mid \lambda \vDash \gamma\}$. The second term, $\eta_{X,\bowtie,c}$, should be true when $\mu^s_\pi(X) \bowtie c$. We define them as:

$$\begin{aligned}
\varepsilon_{X,\gamma,s,\pi} &= \bigwedge_{\lambda\in X} \kappa_{\gamma,\lambda} \wedge \bigwedge_{\lambda \in \mathsf{Paths}(\pi^s)\setminus X} \neg\kappa_{\gamma,\lambda};\\
\eta_{X,\bowtie,c} &= \sum_{s_1 a_1 \ldots s_k a_k s_{k+1} \in X} \left(\prod_{i=1}^{k} p_{s_i,a_i,s_{i+1}}\right) \bowtie c;
\end{aligned} \tag{5.60}$$

This brings us to our last set of formulas, $\kappa_{\gamma,\lambda}$, which should be true iff $\lambda \vDash \gamma$. We define them as:

$$\begin{aligned}
\kappa_{\psi,\lambda} &= \delta_{\psi,s_1}; \\
\kappa_{\gamma\wedge\sigma,\lambda} &= \kappa_{\gamma,\lambda} \wedge \kappa_{\sigma,\lambda}; \\
\kappa_{\neg\gamma,\lambda} &= \neg\kappa_{\gamma,\lambda}; \\
\kappa_{\mathsf{do}_a,\lambda} &= \top \text{ if } a_1 = a, \text{ else } \bot; \\
\kappa_{\mathsf{X}\gamma,s_1a_1\ldots s_ka_ks_{k+1}} &= \kappa_{\gamma,s_2a_2\ldots s_{k+1}}; \\
\kappa_{\mathsf{C}^u_{\bowtie r},s_1a_1\ldots s_ka_ks_{k+1}} &= \sum_{i=1}^{u} r_{s_i,a_i,s_{i+1}} \bowtie r;
\end{aligned} \tag{5.61}$$

With these, our definition of α_{φ,s_1} is complete and this formula now holds in the theory of real closed fields iff there exist a P and a R such that $\langle S,P,V,R\rangle, s_1 \vDash \varphi$. This gives us a decision procedure to determine whether such P and a R exist. The formula φ is satsifiable iff one of the pairs of (S,V) that is iterated over has a (P,R) such that $\langle S,P,V,R\rangle, s_1 \vDash \varphi$. This gives us a decision procedure for the satisfiability of an rPLBP formula.

By [Canny, 1988], any existential first-order logic formula can be decided in PSPACE with respect to the size of the formula. The formula α_φ is such an existential first-order logic formula, and its size is dominated by $\delta_{\Diamond^k(\xi),s}$, where ξ, or a subformula of ξ is of the form $\mathsf{P}(\gamma)$. In this scenario, $\delta_{\Diamond^k(\xi),s}$ has a disjunct for each k-step policy from s. There are at most $|\mathcal{A}|^{|S|^{k+1}}$ such policies, where the size of S is at most $(\sum_{a\in\mathcal{A}}|\mathsf{Post}_a|)^{\mathsf{pd}(\varphi)} + \sum_{a\in\mathcal{A}}|\mathsf{Post}_a|$. In terms of the input n, the size of S is $\mathcal{O}(n^n)$, which means we have at most $\mathcal{O}(n^{(n^n)^n})$ k-step policies from s. For each policy, there is an additional disjunct for each subset X of the set $\mathsf{Paths}(\pi^s)$. The total number of subsets X is at most $2^{|S|^{k+1}}$. Therefore, in terms of the input n, the size of the formula $\delta_{\Diamond^k(\xi),s}$ is $\mathcal{O}(n^{(n^n)^n} \times 2^{(n^n)^n}) = \mathcal{O}(2^{2^{I^3}})$. Thus, we can decide whether or not there exist a P and a R such that $\langle S,P,V,R\rangle, s_1 \vDash \varphi$ in 2EXPSPACE. Repeating this process for each of the $O(2^{2^{n^2}})$ configurations of S and V, requires less than double exponential overhead in space. Thus, we can decide the satisfiability problem for rPLBP in 2EXPSPACE. □

Chapter 6

Applications in Reinforcement Learning

In this chapter we show how rPLBP can be used as a language for safety specifications in shielded RL. First, we present an example RL task that we model as an MDP with rewards. Second, we give examples of safety specifications with increasing complexity and show what the effect would be if they were used in a shield.

6.1 Frozen Lake

A commonly used example problem in reinforcement learning is the "frozen lake" task from the Open AI gym [Brockman et al., 2016]. The environment is a 4 by 4 grid as illustrated in Section 6.1 and the player's task is to move across the ice from a starting state to a goal state, without falling into any of the holes in the ice. The effect of the actions of the agent (moving up, down, left, or right) is not deterministic: since the ice is slippery, the player only moves in the intended direction with a probability of $\frac{1}{3}$. The player can also accidentally slide to each of the perpendicular directions (e.g. when moving up, those are left and right), both with a probability of $\frac{1}{3}$. What happens when one of these perpendicular actions is impossible (due to encountering the edge of the playing field), is unclear in the openAI gym documentation. For our model, we assume that the agent then remains in the same square with a probability of $\frac{1}{3}$. For example, moving right in square A, results in the agent being in square E, B or A all with a probability of $\frac{1}{3}$. The rewards structure is very simple: only when the goal state is reached, is 1 point awarded to the agent.

To model the environment, we must decide on the propositional variables, the actions, and the relations between them (in terms of preconditions and postconditions). An intuitive way to model this is to create a propositional variable for each of the squares in the environment, inA, inB, ..., inP, corresponding to the labels assigned in Section 6.1. We define the set of positional propositional variables as

$$\mathsf{Prop}^{\mathsf{pos}} = \{\mathsf{inA}, \mathsf{inB}, \mathsf{inC}, \mathsf{inD}, \mathsf{inE}, \mathsf{inF}, \mathsf{inG}, \mathsf{inH}, \mathsf{inI}, \mathsf{inJ}, \mathsf{inK}, \mathsf{inL}, \mathsf{inM}, \mathsf{inN}, \mathsf{inO}, \mathsf{inP}\}. \tag{6.1}$$

Note that the propositions indicating a place in the grid are all mutually exclusive in our environment: only one can be true at a time. This is not true for the set of propositions by definition, however. Thus, we often have to express that one of them holds, but no other. We define the shorthand $\overline{p}$ for any $p \in \mathsf{Prop}^{\mathsf{pos}}$ to denote that all propositions regarding positioning other than p are false:

$$\overline{p} = \bigwedge_{q \in \mathsf{Prop}^{\mathsf{pos}} \setminus \{p\}} \neg q \tag{6.2}$$

There are some additional characteristics we can assign to states in the model, that are more concerned with game play. For squares with holes in them, we can use the proposition: inHole, and the first and last square can be represented by $\mathsf{atStart}$ and atGift respectively. We define the set of informative propositional variables as

$$\mathsf{Prop}^{\mathsf{inf}} = \{\mathsf{atStart}, \mathsf{inHole}, \mathsf{atGift}\}. \tag{6.3}$$

We then define the complete set of propositional variables Prop as

$$\mathsf{Prop} = \mathsf{Prop}^{\mathsf{inf}} \cup \mathsf{Prop}^{\mathsf{pos}}. \tag{6.4}$$

▶ **Figure 6.1:** Example environment of a reinforcement learning task: the OpenAI gym "Frozen lake" task, where the player (the elf) should move across the ice from its stool to the gift, without falling into any of the holes in the ice.

In any square, the agent can choose from four directions to move towards, unless a movement is towards the edge of the grid. Thus, we can define our action set and preconditions as

$$\begin{aligned} \mathcal{A} &= \{\mathsf{left}, \mathsf{right}, \mathsf{up}, \mathsf{down}\}, \\ \mathsf{pre}_{\mathsf{left}} &= \neg\mathsf{inA} \wedge \neg\mathsf{inE} \wedge \neg\mathsf{inI} \wedge \neg\mathsf{inM}, \\ \mathsf{pre}_{\mathsf{right}} &= \neg\mathsf{inD} \wedge \neg\mathsf{inH} \wedge \neg\mathsf{inL} \wedge \neg\mathsf{inP}, \\ \mathsf{pre}_{\mathsf{up}} &= \neg\mathsf{inA} \wedge \neg\mathsf{inB} \wedge \neg\mathsf{inC} \wedge \neg\mathsf{inD}, \\ \mathsf{pre}_{\mathsf{down}} &= \neg\mathsf{inM} \wedge \neg\mathsf{inN} \wedge \neg\mathsf{inO} \wedge \neg\mathsf{inP}. \end{aligned} \tag{6.5}$$

Defining the postconditions of $\mathcal{A}$ proves difficult. The effect of an action depends on the position in which it is executed: e.g. when in square E, the action right can result in a move to square F, A, or I, while in square K, the action right can result in a move to square G, L, or O. Moreover, in any square which contains a hole, any action can result only in staying in the hole. Note that in this particular environment, each square can be the result of *any* action. One might conclude from this observation that the set of postconditions for each action must include one postcondition for each square in the grid:

$$\mathsf{Post}_a = \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inB} \wedge \overline{\mathsf{inB}}, \ldots, \mathsf{inP} \wedge \overline{\mathsf{inP}}\}, \text{ for all } a \in \mathcal{A}. \tag{6.6}$$

However, by Definition 4, any well-defined MDP $M = \langle S, P, V, R\rangle$ over $\mathcal{A}$ must have that for any state s and executable action a, all postconditions $\mathsf{post}_{a,i}$ must correspond to exactly one state s' such that $P(s,a)(s') > 0$ and $M, s' \vDash \mathsf{post}_{a,i}$ (requirement (iii.ii)). Thus, for example in any state where inE holds, the action right must result in a non-zero probability of transitioning to 16 states, each satisfying a different postcondition. This is not a correct model of our task environment: e.g. from a state where inE holds, the action right should never result in a state where inP holds, since we cannot move to any square other than the adjacent ones.

We will discuss two possible solutions to this modelling problem: choosing the action set differently, and changing the way preconditions and postconditions are modelled. For the second option, great care has to be taken to see what the effect is on the semantics of the logic, as well as on the results regarding the complexity of model checking and satisfiability.

6.1.1 Modelling state-specific actions

Let us start with the easiest change to implement: choosing the action set differently to represent each unique state-action pair individually. Rather than saying we have the general action right, we now distinguish between the action of moving right when in square A (and thus towards square B), moveAB, the action of moving right when in square B, moveBC, etc. This gives us an alternative set of actions in which for each square, we have actions representing the move to each adjoining square

$$\mathcal{A}' = \{\mathsf{moveAB}, \mathsf{moveAE}, \mathsf{moveBA}, \mathsf{moveBF}, \mathsf{moveBC}, \ldots, \mathsf{moveOP}\}. \tag{6.7}$$

Since each move action is now unique to one square in the environment, we can choose the set of postconditions to match exactly with the three reachable squares.

$$
\begin{array}{lll}
\mathsf{pre}_{\mathsf{moveAB}} & = \mathsf{inA} & \mathsf{Post}_{\mathsf{moveAB}} = \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inB} \wedge \overline{\mathsf{inB}}, \mathsf{inE} \wedge \overline{\mathsf{inE}}\} \\
\mathsf{pre}_{\mathsf{moveAE}} & = \mathsf{inA} & \mathsf{Post}_{\mathsf{moveAE}} = \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inB} \wedge \overline{\mathsf{inB}}, \mathsf{inE} \wedge \overline{\mathsf{inE}}\} \\
\mathsf{pre}_{\mathsf{moveBA}} & = \mathsf{inB} & \mathsf{Post}_{\mathsf{moveBA}} = \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inF} \wedge \overline{\mathsf{inF}}, \mathsf{inB} \wedge \overline{\mathsf{inB}}\} \\
\mathsf{pre}_{\mathsf{moveBF}} & = \mathsf{inB} & \mathsf{Post}_{\mathsf{moveBF}} = \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inF} \wedge \overline{\mathsf{inF}}, \mathsf{inC} \wedge \overline{\mathsf{inC}}\} \\
\mathsf{pre}_{\mathsf{moveBC}} & = \mathsf{inB} & \mathsf{Post}_{\mathsf{moveBC}} = \{\{\mathsf{inB} \wedge \overline{\mathsf{inB}}, \mathsf{inF} \wedge \overline{\mathsf{inF}}, \mathsf{inC} \wedge \overline{\mathsf{inC}}\} \\
\ldots & & \\
\mathsf{pre}_{\mathsf{moveOP}} & = \mathsf{inO} & \mathsf{Post}_{\mathsf{moveOP}} = \{\mathsf{inO} \wedge \overline{\mathsf{inO}}, \mathsf{inK} \wedge \overline{\mathsf{inK}}, \mathsf{inP} \wedge \overline{\mathsf{inP}}\}
\end{array}
$$

We then define MDP $M = \langle S, P, V, R \rangle$ over Prop and $\mathcal{A}'$, such that

- $S = \{s_A, s_B, s_C, \ldots, s_P\}$;
- P : see Figure 6.2;
- $$
\begin{array}{llll}
V(s_A) = \{\mathsf{inA}, \mathsf{atStart}\}, & V(s_B) = \{\mathsf{inB}\}, & V(s_C) = \{\mathsf{inC}\}, & V(s_D) = \{\mathsf{inD}\}, \\
V(s_E) = \{\mathsf{inE}\}, & V(s_F) = \{\mathsf{inB}, \mathsf{inHole}\}, & V(s_G) = \{\mathsf{inG}\}, & V(s_H) = \{\mathsf{inH}, \mathsf{inHole}\}, \\
V(s_I) = \{\mathsf{inI}\}, & V(s_J) = \{\mathsf{inJ}\}, & V(s_K) = \{\mathsf{inK}\}, & V(s_L) = \{\mathsf{inL}, \mathsf{inHole}\}, \\
V(s_M) = \{\mathsf{inM}, \mathsf{inHole}\}, & V(s_N) = \{\mathsf{inN}\}, & V(s_O) = \{\mathsf{inP}\}, & V(s_P) = \{\mathsf{inP}, \mathsf{atGift}\};
\end{array}
$$
- $R : R(s,a)(s') = \begin{cases} 1 & \text{for} \quad s \neq s_P \text{ and } s' = s_P \\ 0 & \text{elsewhere} \end{cases}$

The MDP M, is well-defined according to Definition 4: there are no deadlocks, the probability function is always defined for any executable action, and the postconditions for one action all correspond to a unique, reachable state and vice versa.

While this method of creating state-specific actions works, it is rather laborious. Moreover, we lose the relation between the state-specific actions and the original overarching actions such as left or right. This leads to some unnecessarily complicated expressions. For example, suppose we would like to know if there exists a 3-step policy with an expected reward of at least 5 in which we always choose to move right in the second step. If right would be in our action set, we would be able to express this as

$$\Diamond^3(\mathsf{E}^{1,3}_{\geq 5} \wedge \mathsf{P}_{=1}(\mathsf{X}\mathsf{do}_{\mathsf{right}})). \tag{6.8}$$

However, since right is not in $\mathcal{A}'$, we have to use the state-specific actions. Since we do not know in which state we are after the first step, we must use a disjunction of all state-specific 'right' actions instead.

$$\Diamond^3(\mathsf{E}^{1,3}_{\geq 5} \wedge \mathsf{P}_{=1}(\mathsf{X}(\mathsf{do}_{\mathsf{moveAB}} \vee \mathsf{do}_{\mathsf{moveBC}} \vee \cdots \vee \mathsf{do}_{\mathsf{moveOP}}))) \tag{6.9}$$

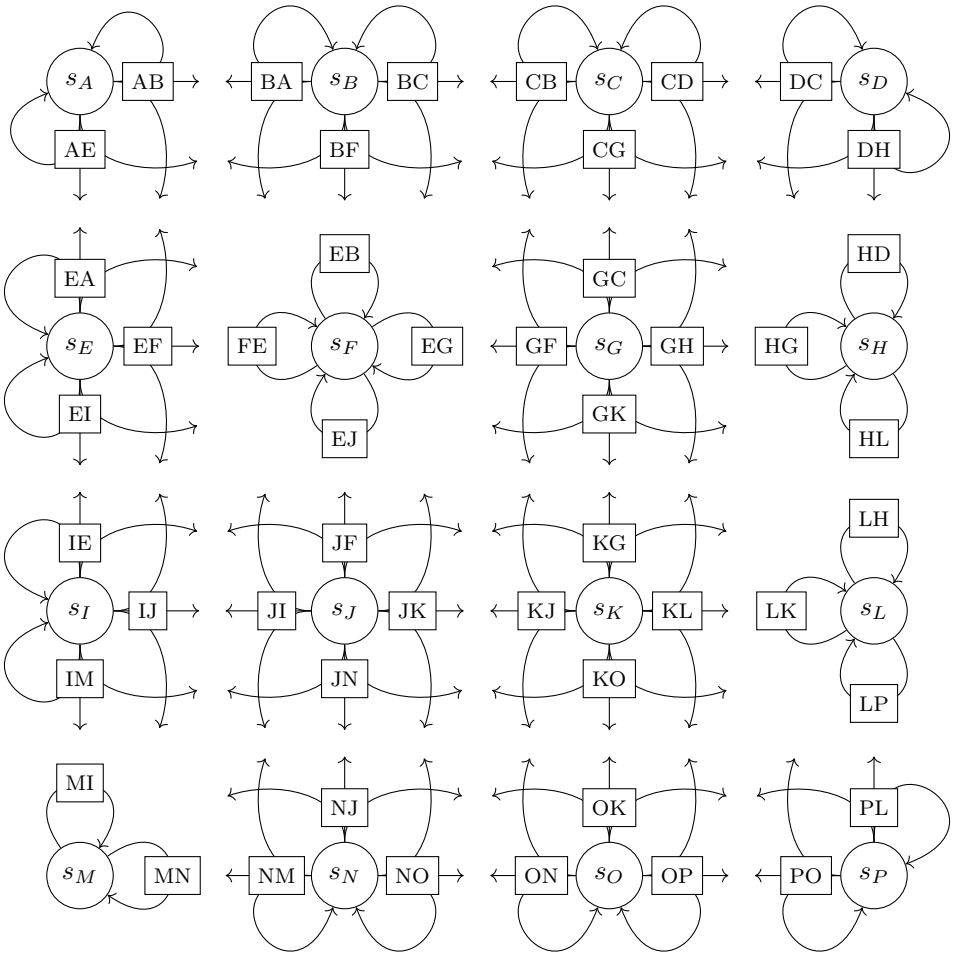
The characteristics of the frozen lake task – a small action space with state-specific effects – can be found in many RL tasks. Therefore, to make the application of the logic to RL more practical, two more structural solutions that do not require state-specific actions are considered next.

6.1.2 Postconditions as functions

Our environment has actions whose effects depend on the conditions before execution (the 'origin square', so to speak). Rather than distinguishing between each state-action combination like in the previous solution, it might be more intuitive to model the postconditions of each action as a function of a conjunction of literals. For example, denoting the effect of moving right in a state where inA holds as

$$\mathsf{Post}_{\mathsf{right}}(\mathsf{inA}) = \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inB} \wedge \overline{\mathsf{inB}}, \mathsf{inE} \wedge \overline{\mathsf{inE}}\}. \tag{6.10}$$

We also change the definition of a precondition to make it a set. This set of conjunctions of literals (that must all be mutually exclusive) can then serve as the domain of our postcondition function. Below we construct the preconditions and postconditions of the left action according to our new definitions.



▶ **Figure 6.2:** MDP model of the frozen lake problem. For any action the probabilities of all drawn transitions are equal (e.g. in case of three arrows: $\frac{1}{3}$ for each option).

$$\mathsf{Pre}_{\mathsf{left}} = \{\mathsf{inB} \wedge \overline{\mathsf{inB}}, \mathsf{inC} \wedge \overline{\mathsf{inC}}, \ldots, \mathsf{inO} \wedge \overline{\mathsf{inO}}\}$$

$$\mathsf{Post}_{\mathsf{left}}(\varphi) = \begin{cases} \{\mathsf{inA} \wedge \overline{\mathsf{inA}}, \mathsf{inF} \wedge \overline{\mathsf{inF}}, \mathsf{inB} \wedge \overline{\mathsf{inB}}\} & \text{for} \quad \varphi = \mathsf{inB} \wedge \overline{\mathsf{inB}} \\ \{\mathsf{inB} \wedge \overline{\mathsf{inB}}, \mathsf{inG} \wedge \overline{\mathsf{inG}}, \mathsf{inC} \wedge \overline{\mathsf{inC}}\} & \text{for} \quad \varphi = \mathsf{inC} \wedge \overline{\mathsf{inC}} \\ \ldots & \\ \{\mathsf{inL} \wedge \overline{\mathsf{inL}}, \mathsf{inO} \wedge \overline{\mathsf{inO}}, \mathsf{inP} \wedge \overline{\mathsf{inP}}\} & \text{for} \quad \varphi = \mathsf{inP} \wedge \overline{\mathsf{inP}} \end{cases}$$

The same can be done for the remaining actions from $\mathcal{A}$: right, up and down.

The effect on the semantics of the logic, and possibly on the complexity of the satisfiability problem, should be investigated further. In Section 7.2, we discuss what such research may look like, and a hypothesis is presented. For the remainder of this chapter, we continue with the MDP M defined in Section 6.1.1 over state-specific action set $\mathcal{A}'$.

6.2 Safety specifications

Now that we have a model of the task environment (the MDP M in Section 6.1.1, over $\mathcal{A}'$), we can see what kind of safety specifications we can define for a shield in this environment. We start simple for the sake of illustration and incrementally include more complex parts of the logic, working towards reward-constrained requirements.

As a first example, consider the following strict safety requirement of never choosing an action in which the probability of falling into a hole is not zero. Even at first glance, we can already see that such a policy is unreasonable: in the first step (square A), both available actions are allowed (down and right), but in squares B and E – one of which the robot *must* pass in order to reach the goal – there are no actions at all that do not have a transition to the hole state. Thus, taking some risks is required to solve this task. Alternatively, consider the requirement that no action may be taken for which the probability of ending in a hole is more than $\frac{1}{3}$. The act of taking an action a in a specific state s can be represented as a 1-step policy π^s, which has only the rule $\pi^s(s) = a$. The action can then only be chosen when the following holds:

$$M, \pi^s \vDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole}) \tag{6.11}$$

When we do offline shielding, we check in advance whether this requirement holds for each state-action combination in our model. See Table 6.1 for the different state-action pairs and whether they satisfy the safety requirement in Equation (6.11).

s	$\pi^s(s)$	$\mu^s_\pi(\{\lambda \in \mathsf{Paths}(\pi^s) \mid M, \lambda \vDash \mathsf{X}\, \mathsf{inHole}\})$	Safety requirement	Conclusion
$s = s_{\mathsf{A}}$	moveAB	$\mu^s_\pi(\emptyset) = 0$	$M, \pi^s \vDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Allowed
$s = s_{\mathsf{A}}$	moveAE	$\mu^s_\pi(\emptyset) = 0$	$M, \pi^s \vDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Allowed
$s = s_{\mathsf{B}}$	moveBA	$\mu^s_\pi(\emptyset) = 0$	$M, \pi^s \vDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Allowed
$s = s_{\mathsf{B}}$	moveBF	$\mu^s_\pi(\{s_B\ \mathsf{moveBF}\ s_F\}) = \frac{1}{3}$	$M, \pi^s \vDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Allowed
...	...	...	...	...
$s = s_{\mathsf{G}}$	moveGC	$\mu^s_\pi(\{s_G\ \mathsf{moveGC}\ s_F, s_G\ \mathsf{moveGC}\ s_H\}) = \frac{2}{3}$	$M, \pi^s \nvDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Blocked
$s = s_{\mathsf{G}}$	moveGK	$\mu^s_\pi(\{s_G\ \mathsf{moveGK}\ s_F, s_G\ \mathsf{moveGK}\ s_H\}) = \frac{2}{3}$	$M, \pi^s \nvDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Blocked
...	...	...	...	...
$s = s_{\mathsf{P}}$	movePO	$\mu^s_\pi(\emptyset) = 0$	$M, \pi^s \vDash P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$	Allowed

▶ **Table 6.1:** Results of offline shielding with the safety requirement $P_{\leq \frac{1}{3}}(\mathsf{X}\, \mathsf{inHole})$. Only a selection of state-action pairs is presented, but no blocked state-action pairs were omitted.

We can conclude from Table 2.1 that only the actions moveGK and moveGC in state s_G violate the safety requirement in Equation (6.11) and are therefore blocked if we apply a shielded RL algorithm. However, we may feel that states such as s_G are undesirable to enter at all: once we are there, our safest options are moving left or right which can also take us back up, away from the target. If we want to avoid entering states like s_G, or other undesirable states, we have to have a safety requirement that looks further ahead.

6.2.1 Looking further ahead

If we want to look further ahead before taking an action, we can create a requirement along the lines of "only take actions after which we have a k-step policy of which we are certain it does not lead us to an unsafe state". The higher the k, the further we look ahead to check for danger, so to speak. As an example, we can say we simply want to avoid the square G with a look-ahead of 1. Again, our requirement takes the form of model checking a 1-step policy which represents a state-action combination:

$$M, \pi^s \vDash P_{=1}(\mathsf{X}(\Diamond^1(P_{=0}(\mathsf{X}\,\mathsf{inG})))) \tag{6.12}$$

This requirement essentially states that the 1-step policy (i.e. the selected action) must lead with certainty to a state from which the agent can follow a 1-step policy which never leads to square G.

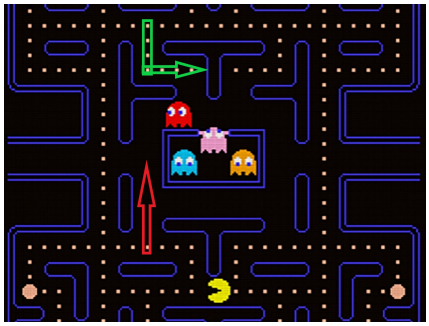
However, we only know from manual inspection of this small example that G is a problematic square, and we may wish to generalise the notion of an undesirable state more. For example, states that have some action (i.e. 1-step policy) that leads to a hole with a probability $> \frac{1}{3}$:

$$M, \pi^s \vDash P_{=1}(\mathsf{X}(\Diamond^1(P_{=0}(\mathsf{X}\,(\Diamond^1(P_{>\frac{1}{3}}(\mathsf{inHole}))))))) \tag{6.13}$$

If we were to use offline shielding and check this property for each state-action pair, we would end up doing more work than necessary: the shield actually prevents certain states from being visited at all (or at least significantly reduces the probability). This makes it more suitable for online shielding. Below we show the first three steps of an example run of the algorithm.

1. The agent is in s_A.
   (a) The RL algorithm selects the action moveAB.
   (b) The shield checks whether the safety requirement holds for policy $\pi^{s_A}(s_A) = \mathsf{moveAB}$. This is true since only state s_G satisfies $\Diamond^1(P_{>\frac{1}{3}}(\mathsf{inHole}))$, and s_G cannot be reached in two steps from s_A with any policy.
   (c) The agent performs action moveAB.
2. The agent is in s_B.
   (a) The RL algorithm selects the action moveBC.
   (b) The shield checks whether the safety requirement holds for policy $\pi^{s_B}(s_B) = \mathsf{moveBC}$. It does not, since moveBC can take the agent to s_C, from which any policy has a non-zero probability of leading to s_G, in which $\Diamond^1(P_{>\frac{1}{3}}(\mathsf{inHole}))$ holds.
   (c) The agent is forced to reconsider and selects action moveBA.
   (d) The shield checks whether the safety requirement holds for policy $\pi^{s_B}(s_B) = \mathsf{moveBA}$. The action moveBA can take the agent to states s_A, s_B, or s_{hole}, which are all at least two steps away from s_G, the only state in which $\Diamond^1(P_{>\frac{1}{3}}(\mathsf{inHole}))$ holds. Thus, the safety requirement holds.
   (e) The agent performs the action moveBA.
3. The agent is in s_A.
   (a) The RL algorithm selects the action moveAE.
   (b) The shield checks whether the safety requirement holds for policy $\pi^{s_A}(s_A) = \mathsf{moveAE}$. This is true since only state s_G satisfies $\Diamond^1(P_{>\frac{1}{3}}(\mathsf{inHole}))$, and s_G cannot be reached in two steps from s_A with any policy.
   (c) The agent takes action moveAE.

These first three steps of an example run should also help convince you that some states are never reached using this particular shield: whenever the agent is in s_B, it must 'turn back' since only moveBA is allowed. Thus, s_C, s_D, and s_G are never reached, and it would therefore be wasted work to check in advance which actions are allowed in those states. In this example, only a few states are rendered unreachable by the safety restrictions, but as explained in Section 2.3.1, for large and sparse environments (e.g. the example MDP in Figure 2.5), this can significantly reduce the computation time.



▶ **Figure 6.3:** Example of routes in the PAC-MAN game that are allowed (green) and not allowed (red) when using a shield that only allows the agent to approach the ghosts if there are rewards on the path.

6.2.2 Incorporating reward

So far we have motivated the use of a probabilistic logic that can reason about specific policies as well as the existence of certain policies, but we have yet to see an example application of the reward operators in rPLBP. Next, we discuss two types of shields that include rewards: one where the rewards themselves are safety-critical, and one where the rewards influence the safety risks that can be taken.

First we look at a scenario where the rewards the agent can receive are integral to the safety. Consider, for example, a task in which a robot moves through a warehouse while occasionally passing charging stations. If the rewards in our model represent battery life, and both positive (charging) and negative (energy-using) actions exist in the environment, we might wish to model some safety requirements in terms of rewards. To decrease the chance of the robot running out of battery in the middle of a busy warehouse, we can demand that we only take routes where some strategy exists such that the accumulated reward in the foreseeable future (e.g. 10 steps) is always at least 0. Formally, this would mean using a shield with the following formula:

$$M, \pi^s \vDash P_{=1}(\mathsf{X}(\Diamond^{10}(\mathsf{P}_{=1}(\mathsf{C}^{10}_{\geq 0})))) \tag{6.14}$$

This sentence reads as: "The next action always takes us to a state where there exists a 10-step policy that gives us with certainty a reward of at least 0."

In such examples, the rewards are truly tied to physical (or virtual) safety, but this is not the only scenario where reasoning about rewards can be useful. In environments where the goals are only reachable by taking safety risks, having strict safety requirements prevents us from solving the task. However, simply easing up on the requirements may be undesirable: the agent is now also allowed to take unsafe actions in all other parts of the environment. Rather, we want to create safety requirements that are conditional on rewards. This way, certain safety risks are allowed but are contingent on their expected success. This is where we can use our reward operators.

An excellent example of tasks that can benefit from reward-contingent shielding, is the task of devising a strategy for the PAC-MAN game. As explained earlier in Section 2.3.1, the player (our learning agent) must collect dots (rewards), without being captured by any of the ghosts patrolling the maze. A very risk-averse shielding definition that keeps the agent far away from the ghosts can hinder the agent from collecting all the dots and lead to a sub-optimal strategy. However, simply allowing the agent to approach the ghosts further, means the agent can also do that when there are no dots in that area left to collect: a useless risk. Without going into too much detail about modelling the PAC-MAN environment, we can consider the following reward-contingent shielding requirement on the 1-step policy $\pi^s(s) = a$ for any state-action pair (s, a):

$$M, \pi^s \vDash \mathsf{P}_{=1}(\mathsf{X}(\Diamond^5(\mathsf{E}^{1,5}_{=0} \rightarrow \mathsf{P}_{\leq \frac{1}{5}}(\mathsf{X}^5\mathsf{isCaught})))), \tag{6.15}$$

where X^n is shorthand for repeating X n times. Given that the model has a proposition $\mathsf{isCaught}$ which is true in states where the agent is in the same square as a ghost, this reads as: "For any outcome of performing action a in state s, we have that there is a 5-step policy such that if the total expected reward is 0, then the probability of being caught in the next 5 steps is at most $\frac{1}{5}$. For a visual example of what types of routes are allowed and not allowed in the PAC-MAN game when using such a shield, see Figure 6.3.

Chapter 7

Conclusion

In this thesis, we have introduced rPLBP, which extends the logic PLBP with policy formulas and reward operators. This logic allows us to make statements about both safety and reward, such as "the agent can act in such a way in the next three steps, that either the expected reward is at least 2, or the probability of catching fire is less than 0.1." We have shown that rPLBP's model-checking problem is PSPACE-complete and its satisfiability problem lies in 2EXPSPACE. Moreover, we have illustrated through theoretical examples that rPLBP can be of practical use for shielded reinforcement learning.

7.1 Discussion

The results of this thesis, which are summarised above, can be criticised on two main points: the lack of implementation and experimental examples of shielding with rPLBP, and the difficulties encountered when modelling a 2D grid environment.

First of all, we discuss the lack of experimental examples for using rPLBP as the specification language for shielded learning. Regarding rPLBP, we draw two conclusions: (1) computationally speaking, using rPLBP for shielding is feasible, and (2) rPLBP can be of practical use for shielded reinforcement learning. Concerning the first conclusion, the absence of an implemented rPLBP model checking or shielding is not necessarily problematic. We have demonstrated that the model-checking problem for rPLBP belongs to the same complexity class (PSPACE) as LTL, which is the widely used language for shielding specifications. Consequently, we can infer that rPLBP shielding is computationally as feasible as LTL shielding.

Nevertheless, for the second conclusion, the inclusion of experimental results would undoubtedly bolster the finding. The reward-constrained specifications mentioned in Section 6.2.2 offer hypothetical suggestions on how a language with reward operators and policy formulas could enhance the ability to balance safety and risk. Conducting tests using such shielding specifications on a benchmark set of RL tasks would provide valuable insights into whether this approach improves performance and/or safety compared to existing (probabilistic) shielding methods. Thus, we emphasise this area as one of the most crucial avenues for future research.

Second of all, we discuss the difficulties that we have encountered in Section 6.1 when modelling the frozen lake problem, a 2D grid environment common in many RL tasks. The strict conditions on the relation between the actions of the logic and the MDP describing the environment were found to hinder a smooth modelling process. To make rPLBP a more suitable language for its intended purpose in RL, we propose that it is investigated whether changes to the underlying model are possible with a limited impact on the complexity results. As a first example, as already alluded to in Section 6.1.2, it may be more suitable to formulate postconditions of actions as functions of preconditions. While the action set is not technically a part of the model of rPLBP, the definition of MDPs in the context of rPLBP would of course have to be altered to accommodate this change. A more drastic change may also be considered: swapping out the underlying models, MDPs, for alternative models that are more accommodating to grid-like environments. For both of these changes, great care would have to be taken to verify that the complexity results of the model-checking and satisfiability problem remain valid.

7.2 Future work

The discussion of the limitations in the previous section has already brought forward two suggested topics for future research. We introduce four additional research questions in this segment. While the earlier suggestions aimed to bolster the conclusion of this thesis, the topics presented here provide avenues to complement and extend this thesis further.

Comparison operators

Extending rPLBP with operators that assert a certain comparative property of two paths, policies, and/or states could prove useful for RL applications. For instance, introducing an operator capable of expressing "the probability that policy A produces a path satisfying γ, is higher than the probability that policy B produces such a path", may be beneficial if we have some benchmark policy for the agent that any other policy must outperform in terms of safety or reward.

Moreover, operators to compare a single path, policy, or state to the rest of the model could be introduced to the logic. For instance, we could express statements like "policy A has the highest expected reward among all policies from the same starting state" or "path X from policy A has a cumulative reward higher than the expected reward of policy A." These comparative assessments could also extend to probabilities of paths instead of rewards, in order to assess how the probability of an undesirable outcome for one policy compares to (the average of) all other available policies.

Shielding on policies

Another interesting research question would be whether it is possible and/or desirable to apply shielding on policies instead of single actions. In shielding implementations so far, the safety specifications are verified for any action selected by the agent. Within these specifications – if they in rPLBP – we can make assertions about policies, but we do not reason about the actual policy of the agent in the forseeable future. In many reinforcement learning algorithms, the policy is updated incrementally based on experience. Therefore, it is worthwhile to, theoretically or experimentally, conceptualise shielding on policies instead of isolated actions.

We expect this approach to raise interesting implementation questions, such as: if a policy is blocked by the shield, how can we efficiently repair our policy to one that meets the safety requirements. After all, if it is a 5-step policy, and each action roughly has 3 possible outcomes, how do we know which of the 121 action choices to change? However, we feel that such questions are exactly what makes it a worthwhile effort to investigate it, since, to our knowledge, no research has been done before on the topic of policy shielding.

Multi-agent extension

Another direction in which rPLBP may be explored is in the development of a multi-agent extension. In multi-agent processes, the probability distribution of the transitions depends on the actions of all the agents together – either synchronous (all agents choose at the same time) or asynchronous (turn-wise). A multi-agent extension of rPLBP might be able to express statements such as "together, agent A and B have a strategy with which they can win the game in three steps with a probability of 0.3." A strong link will likely exist between such an extension and PATL*, by [Chen and Lu, 2007], which is also a probabilistic multi-agent strategy logic. However, notable differences are that PATL* has unbounded-time formulas and that PATL* does not allow explicit assertions about actions in the formulas.

Relation to resource-bounded logics

Throughout this thesis, we have discussed a few times that our definition of rewards may also allow for negative rewards that represent costs. If this is the case, we can reason with rPLBP about whether there exists a policy for which the expected costs are at most some bound b, and which satisfies some other requirements (e.g. reaching a target). This seems to be closely related to a different kind of temporal logic: resource-bounded logic. Resource-bounded logic provides a framework to express the abilities and limitations of agents under resource constraints. For instance, [Alechina et al., 2010], is a multi-agent resource-bounded logic with which one can express statements such as "agent A and agent B can strategise to win the game together, without spending more than 10 coins (supposing these are a resource within the game)". While this is a multi-agent, infinite-time, and deterministic version of the

type of expressions we can make with rPLBP, it is unmistakably similar in nature. A formal analysis of how rPLBP can be used (or be extended in order to be used) as a resource=bounded logic, therefore seems promising.

Bibliography

[Alechina et al., 2010] Alechina, N., Logan, B., Nga, N. H., and Rakib, A. (2010). Resource-bounded alternating-time temporal logic. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 481–488. Citeseer.

[Alshiekh et al., 2018] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. (2018). Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

[Andova et al., 2004] Andova, S., Hermanns, H., and Katoen, J.-P. (2004). Discrete-time rewards model-checked. *Lecture notes in computer science*, 2791:88–104.

[Aziz et al., 1995] Aziz, A., Singhal, V., Balarin, F., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. (1995). It usually works: The temporal logic of stochastic systems. In *Computer Aided Verification: 7th International Conference, CAV'95 Liège, Belgium, July 3–5, 1995 Proceedings 7*, pages 155–165. Springer.

[Bacci et al., 2021] Bacci, G., Delahaye, B., Larsen, K. G., and Mariegaard, A. (2021). Quantitative analysis of interval markov chains. *Model Checking, Synthesis, and Learning: Essays Dedicated to Bengt Jonsson on The Occasion of His 60th Birthday*, pages 57–77.

[Bianco and Alfaro, 1995] Bianco, A. and Alfaro, L. d. (1995). Model checking of probabilistic and nondeterministic systems. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[Canny, 1988] Canny, J. (1988). Some algebraic and geometric computations in pspace. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 460–467.

[Chang and Keisler, 2012] Chang, C. C. and Keisler, H. J. (2012). *Model Theory*. Courier Corporation.

[Chen and Lu, 2007] Chen, T. and Lu, J. (2007). Probabilistic alternating-time temporal logic and model checking algorithm. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, volume 2, pages 35–39. IEEE.

[Chodil et al., 2022] Chodil, M., Kučera, A., and Křetínskỳ, J. (2022). Satisfiability of quantitative probabilistic ctl: Rise to the challenge. In *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, pages 364–387. Springer.

[Clarke and Emerson, 1981] Clarke, E. M. and Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on logic of programs*, pages 52–71. Springer.

[De Giacomo and Vardi, 2013] De Giacomo, G. and Vardi, M. Y. (2013). Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery.

[Emerson and Halpern, 1986] Emerson, E. A. and Halpern, J. Y. (1986). "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM (JACM)*, 33(1):151–178.

[Forejt et al., 2011] Forejt, V., Kwiatkowska, M., Norman, G., and Parker, D. (2011). Automated verification techniques for probabilistic systems. *Formal Methods for Eternal Networked Software Systems:*

11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures 11, pages 53–113.

[Garcıa and Fernández, 2015] Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

[Goodrich et al., 2014] Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2014). *Data structures and algorithms in Java*, chapter 5. John wiley & sons.

[Hansson and Jonsson, 1994] Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal aspects of computing*, 6:512–535.

[Hasanbeig et al., 2020] Hasanbeig, M., Abate, A., and Kroening, D. (2020). Cautious reinforcement learning with logical constraints. *arXiv preprint arXiv:2002.12156*.

[Jamroga, 2008] Jamroga, W. (2008). A temporal logic for markov chains. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 697–704. Citeseer.

[Jansen et al., 2020] Jansen, N., Könighofer, B., Junges, S., Serban, A., and Bloem, R. (2020). Safe reinforcement learning using probabilistic shields. In *31st International Conference on Concurrency Theory*, pages 31–316. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

[Könighofer et al., 2021] Könighofer, B., Rudolf, J., Palmisano, A., Tappler, M., and Bloem, R. (2021). Online shielding for stochastic systems. In *NASA Formal Methods Symposium*, pages 231–248. Springer.

[Motamed et al., 2023] Motamed, N., Alechina, N., Dastani, M., Doder, D., and Logan, B. (2023). Probabilistic temporal logic for reasoning about bounded policies. In *IJCAI 2023: The 32nd International Joint Conferences on Artificial Intelligence*.

[Odriozola-Olalde et al., 2023] Odriozola-Olalde, H., Zamalloa, M., and Arana-Arexolaleiba, N. (2023). Shielded reinforcement learning: A review of reactive methods for safe learning. In *2023 IEEE/SICE International Symposium on System Integration (SII)*, pages 1–8. IEEE.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee.

[Sadigh et al., 2014] Sadigh, D., Kim, E. S., Coogan, S., Sastry, S. S., and Seshia, S. A. (2014). A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 1091–1096. IEEE.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192.