A Specification Logic for Programs in the Probabilistic Guarded Command Language
(Extended Version)

Raúl Pardo[1], Einar Broch Johnsen[2], Ina Schaefer[3], and Andrzej Wąsowski[1]

[1] IT University of Copenhagen, Copenhagen, Denmark,
[2] University of Oslo, Oslo, Norway
[3] Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract. The semantics of probabilistic languages has been extensively studied, but specification languages for their properties have received little attention. This paper introduces the probabilistic dynamic logic pDL, a specification logic for programs in the probabilistic guarded command language (pGCL) of McIver and Morgan. The proposed logic pDL can express both first-order state properties and probabilistic reachability properties, addressing both the non-deterministic and probabilistic choice operators of pGCL. In order to precisely explain the meaning of specifications, we formally define the satisfaction relation for pDL. Since pDL embeds pGCL programs in its box-modality operator, we first provide a formal MDP semantics for pGCL programs. The satisfaction relation is modeled after PCTL, but extended from propositional to first-order setting of dynamic logic, so also embedding program fragments. We study basic properties of this specification language, such as weakening and distribution, that can support reasoning systems. Finally, we demonstrate the use of pDL to reason about program behavior.

1 Introduction

This paper introduces a specification language for probabilistic programs. Probabilistic programming techniques and systems are becoming increasingly important not only for machine-learning applications but also for, e.g., random algorithms, symmetry breaking in distributed algorithms and in the modelling of fault tolerance. The semantics of probabilistic languages has been extensively studied in recent research [1–5], but specification languages for their properties have received little attention (but see, e.g., [6]).

The specification language we define in this paper is the probabilistic dynamic logic pDL, a specification logic for programs in the probabilistic guarded command language pGCL of McIver and Morgan [7]. This programming language combines the guarded command language of Dijkstra [8], in which the non-deterministic scheduling of threads is guarded by Boolean assertions, with state-dependent probabilistic choice. Whereas guarded commands can be seen as a core language for concurrent execution, pGCL can be seen as a core language for probabilistic, non-deterministic execution.

The proposed logic pDL can express both first-order state properties and reachability properties, addressing the non-deterministic as well as the probabilistic choice operators of pGCL. Technically, pDL is a probabilistic extension of (first-order) dynamic logic [9], a modal logic in which programs can occur within the modalities of logical formulas. The semantics of dynamic logic is defined as a Kripke-structure over the set of valuations of program variables. Dynamic logic allows reachability properties to be expressed for given (non-probabilistic) programs by means of modalities. The probabilistic extension pDL allows probabilistic reachability properties to be similarly expressed.

In order to precisely explain the meaning of specifications expressed in pDL, we formally define the semantics of this logic. Since pDL embeds pGCL programs in its modalities, this formalization builds on a formal semantics for pGCL programs, which we define by Markov Decision Processes (MDP) [10]. The MDP semantics of pGCL is used to define the satisfaction relation for pDL. The satisfaction relation is modeled after PCTL [11], but extended from a propositional to a first-order setting of dynamic logic, embedding program fragments in the modalities. The formalization of pDL allows us to study basic properties of specifications, such as weakening and distribution, which are properties underlying proof systems. Finally, we demonstrate how pDL can be used to specify and reason about program behavior. The main contributions of this paper are:

- We define a simple MDP semantics for specific pGCL programs (along the lines of [12]), thereby fixing the semantics of pGCL in terms of MDPs;
- We define the specification language pDL to syntactically express probabilistic properties of programs written in pGCL;
- We define the semantics of pDL over the MDPs of pGCL programs; the satisfaction relation is modeled after PCTL, but extended from a propositional to a first-order setting of dynamic logics with embedded pGCL programs; and
- We study basic properties of pDL and demonstrate how pDL can be used to specify and reason about pGCL programs.

Our motivation for this work is ultimately to define a proof system which allows us to mechanically verify high-level properties for programs written in probabilistic programming languages. Dynamic logic has proven to be a particularly successful logic for such verification systems in the case of regular (non-probabilistic) programs; in particular, KeY [13], which is based on forward reasoning over DL formulas, has been used for breakthrough results such as the verification of the TimSort algorithm [14]. The specification language introduced in this paper constitutes a step in this direction, especially by embedding probabilistic programs into the modalities of the specification language. Further, the semantic properties of the specification language form a semantic basis for proof rules, to be formalized, proven correct, and implemented in future work.

2 Preliminaries

We review the basic semantic notions used in the main part of the paper.

Definition 1 (Markov Decision Process). *A* Markov Decision Process *(MDP) is a tuple* $M = (\mathit{State}, \mathit{Act}, \mathbf{P})$ *where (i) State is a countable set of states, (ii) Act is a countable set of actions, and (iii)* $\mathbf{P} : \mathit{State} \times \mathit{Act} \to \mathit{Dist}(\mathit{State})$ *is a partial transition probability function.*

Let σ denote the states and a the actions of an MDP. A state σ is *final* if no further transitions are possible from it, i.e. $(\sigma, a) \notin \mathrm{dom}(\mathbf{P})$ for any a. A *path*, denoted $\overline{\sigma}$, is a sequence of states $\sigma_1, \ldots, \sigma_n$ such that σ_n is final and there are actions $a_1, \ldots, a_{n-1}$ such that $\mathbf{P}(\sigma_i, a_i)(\sigma_{i+1}) \geq 0$ for $1 \leq i < n$. Let $\mathrm{final}(\overline{\sigma})$ denote the final state of a path $\overline{\sigma}$.

For a given state, the set of applicable actions of $\mathbf{P}$ defines the *demonic choices* between successor state distributions. A *positional policy* π is a function that maps states to actions, so $\pi : \mathit{State} \to \mathit{Act}$. We assume π to be consistent with $\mathbf{P}$, so $\mathbf{P}(\sigma, \pi(\sigma))$ is defined. Given a policy π, we define a transition relation $\dot{\to}_\pi \subseteq \mathit{State} \times [0;1] \times \mathit{State}$ on states that resolves all the demonic choices in $\mathbf{P}$ and write:

$$\sigma \xrightarrow{p}_\pi \sigma' \quad \text{ iff } \quad \mathbf{P}(\sigma, \pi(\sigma))(\sigma') = p. \tag{1}$$

For a given policy π, we let $\xrightarrow{p}{}^*_\pi \subseteq \mathit{State} \times [0;1] \times \mathit{State}$ denote the reflexive and transitive closure of the transition relation, and define the probability of a path $\overline{\sigma} = \sigma_1, \ldots, \sigma_n$ by

$$\Pr(\overline{\sigma}) = 1 \cdot p_1 \cdots p_n \quad \text{ where } \sigma_1 \xrightarrow{p_1}_\pi \cdots \xrightarrow{p_n}_\pi \sigma_n. \tag{2}$$

Thus, a path with no transitions consists of a single state σ, and $\Pr(\sigma) = 1$. Let $\mathrm{paths}_\pi(\sigma)$ denote the set of all paths with policy π from σ to final states.

In this paper we assume that MDPs (and the programs we derive them from) arrive at final states with probability 1 under all policies. This means that the logic pDL that we will be defining and interpreting over these MDPs can only talk about properties of almost surely terminating programs, so in general it cannot be used to reason about termination without adaptation. This is what corresponds to the notion of partial correctness in non-probabilistic proof systems.

An MDP may have an associated *reward function* $r : \mathit{State} \to [0;1]$ that assigns a real value $r(\sigma)$ to any final state $\sigma \in \mathit{State}$. (In this paper we assume that rewards are zero everywhere but in the final states.) We define the *expectation* of the reward starting in a state σ as the greatest lower bound on the expected value of the reward over all policies; so the real valued function defined as

$$\mathbf{E}_\sigma(r) = \inf_\pi \mathbb{E}_{\sigma,\pi}(r) = \inf_\pi \sum_{\overline{\sigma} \in \mathrm{paths}_\pi(\sigma)} \Pr(\overline{\sigma})\, r(\mathrm{final}(\overline{\sigma})) \quad , \tag{3}$$

where $\mathbb{E}_{\sigma,\pi}(r)$ stands for the *expected value* of the random variable induced by the reward function under the given policy, known as the *expected reward*. Note that the expectation $\mathbf{E}_\sigma(r)$ always exists and it is well defined. First, for a given policy the expected value $\mathbb{E}_{\sigma,\pi}(r)$ is guaranteed to exist, as we only consider terminating executions and our reward functions are bounded, non-negative, and non-zero in final states only. The set of possible positional policies that we are minimizing over might be infinite, but the values we are minimizing over are bounded from below by zero, so the set of expected values has a well defined infimum. Finally, because the MDPs considered here almost surely arrive at a final state, we do not need to condition the expectations on terminating paths to re-normalize probability distributions, which greatly simplifies the technical machinery.

To avoid confusing expectations and scalar values, we use bold font for expectations in the sequel. For instance, $\boldsymbol{p}$ represents an unknown expectation from the state space into [0;1], and $\mathbf{0}$ represents a constant expectation function, equal to zero everywhere.

$$
\begin{aligned}
v \;\; &::= \mathit{true} \mid \mathit{false} \mid 0 \mid 1 \mid \ldots \\
e \;\; &::= v \mid x \mid \mathit{op}\ e \mid e\ \mathit{op}\ e \\
\mathit{op} &::= + \;\mid\; - \;\mid\; * \;\mid\; / \;\mid > \mid == \mid \geq \\
s \;\; &::= s \sqcap s \mid s\ {}_{e}\!\oplus s \mid s; s \mid \texttt{skip} \mid x := e \mid \texttt{if}\ e\ \{s\}\ \texttt{else}\ \{s\} \mid \texttt{while}\ e\ \{s\}
\end{aligned}
$$

Fig. 1. The syntax of the probabilistic guarded command language pGCL

We use characteristic functions to define rewards for the semantics of pGCL programs, consistently with McIver & Morgan [7]. For a formula φ in some logic with the corresponding satisfaction relation, a characteristic function $[\![\varphi]\!]$, also known as a Boolean embedding or an indicator function, assigns 1 to states satisfying φ and 0 otherwise. In this paper, models will be program states, and also states of an MDP. In general, characteristic functions can be replaced by arbitrary real-valued functions [2], but this is not needed to interpret logical specifications, so we leave this to future work.

Finally, given a formula φ that can be interpreted over a state space of an MDP, we define the truncation of a reward function $\boldsymbol{p}$ as the function $(\boldsymbol{p}{\downarrow}\varphi)(\sigma) = \boldsymbol{p}(\sigma) \cdot [\![\varphi]\!](\sigma)$. The truncation of $\boldsymbol{p}$ to φ maintains the original value of $\boldsymbol{p}$ for states satisfying φ and gives zero otherwise. Note that $\boldsymbol{p}{\downarrow}\varphi_1$ remains a valid reward function if $\boldsymbol{p}$ was.

3 pGCL: A Probabilistic Guarded Command Language

The probabilistic guarded command language pGCL [7], extends Dijkstra's guarded command language [8] with probabilistic choice. Figure 1 gives the syntax of pGCL. We let x range over the set X of program variables, v over primitive values, and e over expressions Exp. Expressions e are constructed over program variables x and primitive values v by means of unary and binary operators op (including logical operators $\neg, \wedge, \vee$ and arithmetic operators $+, -, *, /$). Expressions are assumed to be well-formed.

Statements s include the non-deterministic (or demonic) choice $s_1 \sqcap s_2$ between the branches s_1 and s_2. We write $s\ {}_{e}\!\oplus s'$ for the probabilistic choice between the branches s and s'; if the expression e evaluates to a value p given the current values for the program variables, then s and s' have probability p and $1-p$ of being selected, respectively. In many cases e will be a constant, but in general it can be an *expression over the state variables* (i.e., $e \in \mathit{Exp}$), so its semantics will be an real-valued function. Sequential composition, **skip**, assignment, **if**-**then**-**else** and **while** are standard (e.g., [8]).

The semantics of pGCL programs s is defined as an MDP $\mathcal{M}_s$ (cf. [12]), and its executions are captured by the partial transition probability function for a given policy π, denoted $\xrightarrow{p}_{\pi}$ for some probability p, (Def. 1). A state σ of $\mathcal{M}_s$ is a pair of a *valuation* and a *program*, so $\sigma = \langle \varepsilon, s \rangle$ where the valuation ε is a mapping from all the program variables in s to concrete values (sometimes we omit the program part, if it is unambiguous in the context). The state $\langle \varepsilon, s \rangle$ represents an *initial state* of the program s given some initial valuation ε and the state $\langle \varepsilon, \texttt{skip} \rangle$ represents a *final state* in which the program has terminated with the valuation ε.

The rules defining the partial transition probability function for a given policy π are shown in Fig. 2. We denote by $\langle \varepsilon, s \rangle \xrightarrow{p}_{\pi} \langle \varepsilon', s' \rangle$ the transition from $\langle \varepsilon, s \rangle$ to $\langle \varepsilon', s' \rangle$ by

$$\text{(ASSIGN)}\quad \frac{\varepsilon' = \varepsilon[x \mapsto \varepsilon(e)]}{\langle \varepsilon, x := e\rangle \xrightarrow{1}_{\pi} \langle \varepsilon', \mathtt{skip}\rangle}$$

$$\text{(PROBCHOICE1)}\quad \frac{\varepsilon(e) = p \quad 0 \leq p \leq 1}{\langle \varepsilon, s_1 \;{}_e\!\oplus s_2\rangle \xrightarrow{p}_{\pi} \langle \varepsilon, s_1\rangle}$$

$$\text{(DEMCHOICE)}\quad \frac{i \in \{1,2\} \quad \pi\langle \varepsilon, s_1 \sqcap s_2\rangle = s_i}{\langle \varepsilon, s_1 \sqcap s_2\rangle \xrightarrow{1}_{\pi} \langle \varepsilon', s_i\rangle}$$

$$\text{(PROBCHOICE2)}\quad \frac{\varepsilon(e) = p \quad 0 \leq p \leq 1}{\langle \varepsilon, s_1 \;{}_e\!\oplus s_2\rangle \xrightarrow{1-p}_{\pi} \langle \varepsilon, s_2\rangle}$$

$$\text{(COMPOSITION1)}\quad \frac{\langle \varepsilon', s_2\rangle \xrightarrow{p}_{\pi} \langle \varepsilon'', s\rangle}{\langle \varepsilon, \mathtt{skip}; s_2\rangle \xrightarrow{p}_{\pi} \langle \varepsilon'', s\rangle}$$

$$\text{(IF1)}\quad \frac{\varepsilon(e) = \mathit{true}}{\langle \varepsilon, \mathtt{if}\ e\ \{s_1\}\ \mathtt{else}\ \{s_2\}\rangle \xrightarrow{1}_{\pi} \langle \varepsilon, s_1\rangle}$$

$$\text{(WHILE1)}\quad \frac{\varepsilon(e) = \mathit{true}}{\langle \varepsilon, \mathtt{while}\ e\ \{s\}\rangle \xrightarrow{1}_{\pi} \langle \varepsilon, s; \mathtt{while}\ e\ \{s\}\rangle}$$

$$\text{(COMPOSITION2)}\quad \frac{\langle \varepsilon, s_1\rangle \xrightarrow{p}_{\pi} \langle \varepsilon', s\rangle}{\langle \varepsilon, s_1; s_2\rangle \xrightarrow{p}_{\pi} \langle \varepsilon', s; s_2\rangle}$$

$$\text{(IF2)}\quad \frac{\varepsilon(e) = \mathit{false}}{\langle \varepsilon, \mathtt{if}\ e\ \{s_1\}\ \mathtt{else}\ \{s_2\}\rangle \xrightarrow{1}_{\pi} \langle \varepsilon, s_2\rangle}$$

$$\text{(WHILE2)}\quad \frac{\varepsilon(e) = \mathit{false}}{\langle \varepsilon, \mathtt{while}\ e\ \{s\}\rangle \xrightarrow{1}_{\pi} \langle \varepsilon, \mathtt{skip}\rangle}$$

Fig. 2. An MDP-semantics for pGCL.

action $\alpha = \pi(\langle \varepsilon, s\rangle)$, where p is the resulting probability. Note that for demonic choice, the policy π fixes the action choice between the distributions $0, 1$ and $1, 0$; for all others statements, there is already a single successor distribution. The transitive closure of this relation, denoted $\langle \varepsilon_0, s_0\rangle \xrightarrow{p}{}^{*}_{\pi} \langle \varepsilon_n, s_n\rangle$, expresses that there is a sequence of zero or more such transitions from $\langle \varepsilon_0, s_0\rangle$ to $\langle \varepsilon_n, s_n\rangle$ with corresponding actions $\alpha_i = \pi(\varepsilon_i, s_i)$ and probability p_i for $0 < i \leq n$, such that $p = 1 \cdot p_1 \cdots p_n$.

Remark that the rules in Fig. 2 allow programs to get stuck, for instance if an expression e evaluates to a value outside $[0;1]$ (PROBCHOICE). Since we are interested in partial correctness, we henceforth rule out such programs and only consider programs that successfully reduce to a single `skip` statement under all policies with probability 1.

4 Probabilistic Dynamic Logic

Formulas & Satisfiability. Given sets X of program variables and L of logical variables disjoint from X, let ATF denote the well-formed atomic formulae built using constants, program and logical variables. For every $l \in L$, let $\operatorname{dom} l$ denote the domain of l. We extend valuations to also map logical variables $l \in L$ to values in $\operatorname{dom} l$ and let $\varepsilon \models_{\mathsf{ATF}} \varphi$ denote standard satisfaction, expressing that $\varphi \in \mathsf{ATF}$ holds in valuation ε.

The formulas of probabilistic dynamic logic (pDL) are defined inductively as the smallest set generated by the following grammar:

$$\varphi \quad ::= \quad \mathsf{ATF} \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \forall l \cdot \varphi \ \mid \ [s]_{\boldsymbol{p}}\, \varphi \tag{4}$$

where φ ranges over pDL formulae, $l \in L$ over logical variables, s is a pGCL program with variables in X, and $\boldsymbol{p}$ is an expectation assigning values in $[0;1]$ to initial states of the program s. The logical operators $\rightarrow$, $\vee$ and $\exists$ are derived in terms of $\neg$, $\wedge$ and $\forall$ as usual.

The last operator in Eq. (4) is known as the box-operator in dynamic logics, but now we give it a probabilistic interpretation along with the name "p-box." Given a pGCL program s, we write $[s]_{\boldsymbol{p}}\,\varphi$ to express that the expectation that a formula φ holds after successfully executing s is at least $\boldsymbol{p}$; i.e., the function $\boldsymbol{p}$ represents the expectation for φ in the current state of $\mathcal{M}_s$ using $[\![\varphi]\!]$ as the reward function (see Sect. 2). For the reader familiar with the CTL/PCTL terminology, the p-box formulae are path formulae, and all other formulae are state formulae.

We define semantics of *well-formed formulae* in pDL, so formulae with no free logical variables—all occurrences of logical variables are captured by a quantifier. The definition extends the standard satisfaction relation of dynamic logic [9] to the probabilistic case:

Definition 2 (Satisfaction of pDL Formulas). *Let φ be a well-formed pDL formula, π range over policies, $l \in L$, $\boldsymbol{p}$: State $\rightarrow [0;1]$ be an expectation lower bound, and ε be a valuation defined for all variables mentioned in φ. The* satisfiability *of a formula φ in a model ε, denoted $\varepsilon \models \varphi$, is defined inductively as follows:*

$$\begin{array}{lll}
\varepsilon \models \varphi & \text{iff} & \varphi \in \mathsf{ATF} \quad \text{and} \quad \varepsilon \models_{\mathsf{ATF}} \varphi \\
\varepsilon \models \varphi_1 \wedge \varphi_2 & \text{iff} & \varepsilon \models \varphi_1 \quad \text{and} \quad \varepsilon \models \varphi_2 \\
\varepsilon \models \neg\varphi & \text{iff} & \text{not } \varepsilon \models \varphi \\
\varepsilon \models \forall l \cdot \varphi & \text{iff} & \varepsilon \models \varphi[l := v] \text{ for each } v \in \operatorname{dom} l \\
\varepsilon \models [s]_{\boldsymbol{p}}\varphi & \text{iff} & \boldsymbol{p}(\varepsilon) \leq \mathbf{E}_\varepsilon[\![\varphi]\!] \text{ where the expectation is taken in } \mathcal{M}_s
\end{array}$$

For $\varphi \in \mathsf{ATF}$, $\models_{\mathsf{ATF}}$ can be used to check satisfaction just against the valuation of program variables since φ is well-formed. In the case of universal quantification, the substitution replaces logical variables with constants. The last case (p-box) is implicitly recursive, since the characteristic function $[\![\varphi]\!]$ refers to the satisfaction of φ in the final states of s.

The satisfaction of a p-box formula $[s]_p\,\varphi$ captures a lower bound on the probability of φ holding after the program s. Consequently, pDL supports specification and reasoning about probabilistic reachability properties in almost surely terminating programs.

It is convenient to omit the valuation ε from the satisfaction judgement, meaning that the judgement holds for all valuations (validity):

$$\models [s]_{\boldsymbol{p}}\,\varphi \quad \text{iff} \quad \varepsilon \models [s]_{\boldsymbol{p}}\,\varphi \quad \text{for all valuations } \varepsilon \tag{5}$$

5 The p-box Modality and Logical Connectives

We begin our investigation of pDL by exploring how the p-box operator interacts with different expectations and the other connectives of pDL.

In a proof system, weakening is useful to allow adjusting proven facts to a format of a syntactic proof rule. Since all operators of pDL, with the exception of p-box, behave like in first order logic, the usual qualitative weakening properties apply for these operators at the top-level. For instance, $\varphi_1 \wedge \varphi_2$ can be weakened to φ_1. These properties follow directly from Def. 2. The following proposition states the key properties for p-box:

Proposition 3 (Weakening). *Let ε stand for a valuation, $\boldsymbol{p}, \mathbf{0} \in State \to [0;1]$ be expectation lower bounds, s a pGCL program, and $\varphi \in$ pDL. Then:*

1. *Universal lower bound: $\varepsilon \models [s]_{\mathbf{0}}\, \varphi$*
2. *Quantitative weakening: $\varepsilon \models [s]_{\boldsymbol{p}_1}\, \varphi$ then $\varepsilon \models [s]_{\boldsymbol{p}_2}\, \varphi$ if $\boldsymbol{p}_2 \leq \boldsymbol{p}_1$ everywhere*
3. *Weakening conjunctions: $\varepsilon \models [s]_{\boldsymbol{p}}\, (\varphi_1 \wedge \varphi_2)$ then $\varepsilon \models [s]_{\boldsymbol{p}}\, \varphi_i$ for $i = 1, 2$*
4. *Qualitative weakening: $\varepsilon \models [s]_{\boldsymbol{p}}\, \varphi_1$ and $\models \varphi_1 \to \varphi_2$ then $\varepsilon \models [s]_{\boldsymbol{p}}\, \varphi_2$.*

The first point states that there is a limit to the usefulness of weakening the expectation: if you cannot guarantee that the lower bound is positive, then you do not have any information at all. A zero lower-bound would hold for any property. The second property is a probabilistic variant of weakening, which follows directly from the last case of Def. 2; the lower bound on an expectation can always be lowered. The last two properties are the probabilistic counterparts of weakening in standard (non-probabilistic) dynamic logic; the third property is syntactic for conjunction, the last one is general.

When building proofs with pDL, the other direction of reasoning seems more useful: we would like to be able to derive a conjunction from two independently concluded facts. For state formulae, this holds naturally, like in first-order logic. For p-box formulae, we would like to use the expectations $\boldsymbol{p}_i$ of two formulas φ_i to draw conclusions about the expectation that their conjunction holds. It seems tempting to translate the intuitions from the Boolean lattice to real numbers, and to suggest that a minimum of the expectations for both formulae is a lower bound for their conjunction. To develop some intuition, let us first consider an incorrect proposal using the following counterexample:

Example 4. Consider the program ⚄, modeling a six-sided fair die:

$$⚄ \quad ::= \quad \texttt{x:=1}\ {}_{1/6}{\oplus}\ (\texttt{x:=2}\ {}_{1/5}{\oplus}\ (\texttt{x:=3}\ {}_{1/4}{\oplus}\ (\texttt{x:=4}\ {}_{1/3}{\oplus}\ (\texttt{x:=5}\ {}_{1/2}{\oplus}\ \texttt{x:=6})))) \tag{6}$$

Let 'odd' be an atomic formula stating that a value is odd, and 'prime' an atomic formula stating that it is prime. Since the die is fair, the expectations for each of these after ⚄ are:

$$\models [⚄]_{\mathbf{1/2}}\, \mathrm{odd}(x) \qquad \models [⚄]_{\mathbf{1/2}}\, \mathrm{prime}(x) \tag{7}$$

The minimum of the two expectations is a constant function which equals $\mathbf{1/2}$ everywhere, but the expectation bound in $[s]_{\boldsymbol{p}}(\mathrm{odd}(x) \wedge \mathrm{prime}(x))$ can be at most $\mathbf{1/3}$ since only two outcomes ($x \mapsto 3$ and $x \mapsto 5$) satisfy both predicates. Effectively, even if $\varepsilon \models [s]_{\boldsymbol{p}_1}\, \varphi_1$ and $\varepsilon \models [s]_{\boldsymbol{p}2}\, \varphi_2$ hold, we do not necessarily have $\varepsilon \models [s]_{\min(\boldsymbol{p}_1, \boldsymbol{p}_2)}\, \varphi_1 \wedge \varphi_2$. The reason is that the expectation bounds measure what is the lower bound on satisfaction of a property, but not where in the execution space this probability mass is placed. There is not enough information to see to what extent the two properties are overlapping. □

Similarly, $\boldsymbol{p}(\varepsilon) = \boldsymbol{p}_1(\varepsilon)\boldsymbol{p}_2(\varepsilon)$ is not a good candidate in Ex. 4, since it is only guaranteed to be a lower bound for a conjunction when φ_i are independent events. Unless $\boldsymbol{p}_1 = \boldsymbol{p}_2 = \mathbf{1}$, combining proven facts with conjunction (or disjunction) weakens the expectation:

Theorem 5. *Let ε be a valuation, $\boldsymbol{p}, \boldsymbol{p}_1, \boldsymbol{p}_2 \in State \to [0;1]$ expectation lower bounds, s a pGCL program, and $\varphi_1, \varphi_2 \in$ pDL. Then:*

1. *p-box conjunction: if* $\varepsilon \models [s]_{\boldsymbol{p}_1}\, \varphi_1$ *and* $\varepsilon \models [s]_{\boldsymbol{p}_2}\, \varphi_2$*, then* $\varepsilon \models [s]_{\boldsymbol{p}}\, (\varphi_1 \wedge \varphi_2)$ *where* $\boldsymbol{p} = \max(\boldsymbol{p}_1 + \boldsymbol{p}_2 - 1, 0)$ *everywhere.*
2. *p-box disjunction: if* $\varepsilon \models [s]_{\boldsymbol{p}_1}\, \varphi_1$ *or* $\varepsilon \models [s]_{\boldsymbol{p}_2}\, \varphi_2$*, then* $\varepsilon \models [s]_{\boldsymbol{p}}\, (\varphi_1 \vee \varphi_2)$ *where* $\boldsymbol{p} = \min(\boldsymbol{p}_1, \boldsymbol{p}_2)$ *everywhere.*

Note the asymmetry between these cases: reasoning about conjunctions of low probability properties using Thm. 5.1 is inefficient, and quickly arrives at the lower bound expectation **0**, which, as observed in Prop. 3, holds vacuously. If both properties have an expected probability lower than $1/2$, then pDL cannot really see (in a compositional manner) whether there is any chance that they can be satisfied simultaneously. In contrast, compositional reasoning about disjunctions makes sense both for low and high probability events. This is a consequence of using lower bounds on expectations.

The qualitative non-probabilistic specialization of Thm. 5.1 behaves reasonably: when φ_1 or φ_2 hold almost surely, then the theorem reduces to a familiar format:

$$\text{if } \varepsilon \models [s]_{\boldsymbol{p}}\varphi_1 \text{ and } \varepsilon \models [s]_{\mathbf{1}}\varphi_2 \text{ then } \varepsilon \models [s]_{\boldsymbol{p}}(\varphi_1 \wedge \varphi_2) \tag{8}$$

Theorem 6. *Let* ε *be a valuation,* $\boldsymbol{p} \in \mathit{State} \to [0;1]$ *an expectation lower bound,* s *a pGCL program, and* $\varphi \in$ pDL *a well-formed formula.*

1. *If* $\varepsilon \models [s]_{\boldsymbol{p}}\, \forall l \cdot \varphi$ *then* $\varepsilon \models \forall l \cdot [s]_{\boldsymbol{p}}\, \varphi$*, but not the other way around in general.*
2. *If* $\varepsilon \models \exists l \cdot [s]_{\boldsymbol{p}}\, \varphi$ *then* $\varepsilon \models [s]_{\boldsymbol{p}}\, \exists l \cdot \varphi$ *but not the other way around in general.*

The essence of the above two properties lies in the fact that quantifiers in pDL only affect logical variables, programs cannot access logical variables, and we do not allow quantification over expectation variables.

In a deductive proof system, one works with abstract states, not just concrete states. A state abstraction can be introduced as a precondition, a pDL property that captures the essence of an abstraction, and is satisfied by all the abstracted states sharing the property. If an abstract property is a precondition for a proof, it is naturally introduced using implication. However, implication is unwieldy in an expectation calculus, so it is practical to be able to eliminate it in the proof machinery. The following theorem explains how a precondition can be folded into an expectation function:

Theorem 7 (Implication Elimination). *Let* s *be a* pGCL *program,* φ_i *be* pDL *formulae, and* $\boldsymbol{p}$ *a lower-bound function for expectations. Then:*

$$\models \varphi_1 \to ([s]_{\boldsymbol{p}}\, \varphi_2) \quad \textit{iff} \quad \models [s]_{\boldsymbol{p}\downarrow\varphi_1}\varphi_2$$

Note that we use validity naturally when working with abstract states, as the state is replaced by the precondition in the formula.

Finally, negation in pDL is difficult to push over boxes. This is due to non-determinism and the lower bound semantics of expectations it enforces. A p-box property expresses a lower bound on probability of a post-condition holding after a program. Naturally, a negation of a p-box property will express an *upper-bound* on a property, but pDL has no upper-bound modality first-class. We return to this problem in Sect. 7, where we discuss reasoning about upper-bounds in non-deterministic and in purely probabilistic programs.

6 Expectations for Program Constructs

This section investigates how expectations are transformed by pGCL program constructs, as opposed to logical constructs discussed above. We begin by looking at the composite statements, which build the structure of the underlying MDP. The probabilistic choice introduces a small expectation update, consistent with an expectation of a Bernoulli variable (item 1). The demonic choice (item 2), requires that both sides provide the same guarantee, which is consistent with worst-case reasoning.

Theorem 8 (Expectation and Choices). *Let s_i be programs, φ a PDL formula, $\boldsymbol{p}_i$ lower bound functions for expectations into $[0;1]$, and ε a valuation of variables. Then:*

1. *If $\varepsilon \models [s_1]_{\boldsymbol{p}_1}\varphi$ and $\varepsilon \models [s_2]_{\boldsymbol{p}_2}\varphi$ then $\varepsilon \models [s_1 \,{}_e\!\oplus\, s_2]_{\boldsymbol{p}}\,\varphi$ with $\boldsymbol{p} = \varepsilon(e)\boldsymbol{p}_1 + (1-\varepsilon(e))\boldsymbol{p}_2$*
2. *$\varepsilon \models [s_1]_{\boldsymbol{p}}\,\varphi$ and $\varepsilon \models [s_2]_{\boldsymbol{p}}\,\varphi$ if and only if $\varepsilon \models [s_1 \sqcap s_2]_{\boldsymbol{p}}\,\varphi$*

Note that in the second case, demonic, we can always use weakening (Prop. 3.2) to equalize the left-hand-side expectation lower-bounds using a point-wise minimum, if the premises are established earlier for different lower bound functions.

For any program logic, it is essential that we can reason about composition of consecutive statements; allowing the post-condition of one to be used as a pre-condition for the other. The following theorem demonstrates that sequencing in pGCL corresponds to composition of expectations in the MDP domain. It uses implication elimination (Thm. 7) to compute a post-condition for a sequence of programs. Crucially, the new lower bound is computed using an expectation operation in the MDP of the first program, using the lower-bound of the second program as a reward function. Here, the expectation operation acts as a way to explore the program graph and accumulate values in final states.

Theorem 9 (Expectation and Sequencing). *Let s_i be pGCL programs, φ_i be pDL formulae, ε be a valuation, and $\boldsymbol{p}$ an expectation lower bound function.*

$$\text{If} \models \varphi_1 \to ([s_2]_{\boldsymbol{p}}\,\varphi_2) \text{ then } \varepsilon \models [s_1; s_2]_{\mathbf{E}_{\langle\varepsilon,s_1\rangle}(\boldsymbol{p}\downarrow\varphi_1)}\,\varphi_2 \quad ,$$

where the expectation $\mathbf{E}_{\langle\varepsilon,s_1\rangle}(\boldsymbol{p}\downarrow\varphi_1)$ is taken in $\mathcal{M}_{s_1}$ with $\boldsymbol{p}\downarrow\varphi_1$ as the reward function.

For a piece of intuition, note that the above theorem captures the basic step of a backwards reachability algorithm for MDPs, but expressed in pDL; it accumulates expectations backwards over s_1 from what is already known for s_2.

We now move to investigating how simple statements translate expectations:

Theorem 10 (Unfolding Simple Statements). *Let s be a pGCL program, φ a pDL formula, $\boldsymbol{p}$ a function into $[0;1]$, a lower bound on expectations, and ε a valuation. Then*

1. *$\varepsilon \models [\textbf{skip}]_{\mathbf{1}}\,\varphi$ iff $\varepsilon \models \varphi$*
2. *$\varepsilon \models [s]_{\boldsymbol{p}}\,\varphi$ iff $\varepsilon \models [\textbf{skip}; s]_{\boldsymbol{p}}\,\varphi$*
3. *$\varepsilon \models [\mathtt{x}{:=}e; s]_{\boldsymbol{p}}\,\varphi$ iff $\varepsilon[x \mapsto \varepsilon(e)] \models [s]_{\boldsymbol{p}}\,\varphi$*

The case of if-conditions below is rather classic (Thm. 11.3). For any given state, we can evaluate the head condition and inherit the expectation from the selected branch. For this to work we assume that the atomic formulae (ATF) satisfaction semantics in pDL is consistent with the expression evaluation semantics in pGCL. The case of while loops is much more interesting—indeed a plethora of works have emerged recently on proposing sound reasoning rules for while loop invariants, post-conditions and termination (see Sect. 9). In this paper, we show the simplest possible reasoning rule for loops that performs a single unrolling, exactly along the operational semantics. Of course, we are confident that many other rules for reasoning about while loops (involving invariants, prefixes, or converging chains of probabilities) can also be proven sound in pDL—left as future work.

Theorem 11 (Unfolding Loops and Conditionals). *Let e be a program expression (also an atomic* pDL *formula over program variables in X), φ be a* pDL *atomic formula, s_i be* pGCL *programs, $\boldsymbol{p}$ an expectation lower bound function, and ε a valuation. Then:*

1. *If* $\varepsilon \models e \wedge [s_1]_{\boldsymbol{p}}\, \varphi$ *then* $\varepsilon \models [\texttt{if}\ e\ \{\ s_1\ \}\ \texttt{else}\ \{\ s_2\ \}]_{\boldsymbol{p}}\, \varphi$
2. *If* $\varepsilon \models \neg e \wedge [s_2]_{\boldsymbol{p}}\, \varphi$ *then* $\varepsilon \models [\texttt{if}\ e\ \{\ s_1\ \}\ \texttt{else}\ \{\ s_2\ \}]_{\boldsymbol{p}}\, \varphi$
3. $\varepsilon \models [\texttt{if}\ e\,\{\ s;\ \texttt{while}\ e\ \{\ s\ \}\}\ \texttt{else}\ \{\texttt{skip}\}]_{\boldsymbol{p}}\, \varphi$ *iff* $\varepsilon \models [\texttt{while}\ e\,\{\ s\ \}]_{\boldsymbol{p}}\, \varphi$

7 Purely Probabilistic and Deterministic Programs

The main reason for the lower-bound expectation semantics in pDL (inherited from McIver&Morgan) is the presence of demonic choice in pGCL. With non-determinism in the language, calculating precise probabilities is not possible. However, this does not mean that pDL cannot be used to reason about upper-bounds. The following theorem explains:[4]

Theorem 12 (Joni's Theorem). *For a policy π, property φ, program s, and state ε: if $\varepsilon \models [s]_{\boldsymbol{p}_1} \varphi$ and $\varepsilon \models [s]_{\boldsymbol{p}_2} \neg\varphi$ then $\mathbb{E}_{\pi,\varepsilon}[\![\varphi]\!] \in [p_1, 1 - p_2]$.*

The theorem means that for a purely probabilistic program derived by fixing a policy for a pGCL program s, the expected reward is bounded from below by the expectation of this reward in s, and from above by the expectation of its negation in s. The theorem follows directly from Eq. (3) and the negation case in Def. 2.

For deterministic programs, some surprising properties, follow from interaction of probability and logics. For instance, we can conclude a conjunction of *expectations* from an expectation of a disjunction.

Theorem 13. *Let s be a purely probabilistic pGCL program (a program that does not use the demonic choice), let ε stand for a valuation, $\boldsymbol{p} \in State \to [0;1]$ be an expectation function, and $\varphi_i \in$ pDL properties. Then if $\varepsilon \models [s]_{\boldsymbol{p}}\, (\varphi_1 \vee \varphi_2)$ then there exist $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_1 + \boldsymbol{p}_2 \geq \boldsymbol{p}$ everywhere, such that $\varepsilon \models [s]_{\boldsymbol{p}_1}\, \varphi_1$ and $\varepsilon \models [s]_{\boldsymbol{p}_2}\, \varphi_2$.*

Intuitively, the property holds, because each of the measure of the space of final states of the disjointed properties can be separated between the disjuncts. This separation would not be possible with non-determinism, as shown in the following counterexample.

[4] The theorem is named as a tribute to the song *Both sides now* by Joni Mitchell.

Example 14. Consider the program $\textcircled{s} ::=$ `x := H` $\sqcap$ `x := T`. The following holds for any initial valuation ε:

$$\varepsilon \models [\textcircled{s}]_{\mathbf{1}}(x = \mathtt{H} \lor x = \mathtt{T})$$

This happens because disjunction is weakening and a weaker property is harder to avoid, here impossible to avoid, for an adversary minimizing an expectation satisfaction. However, at the same time: $\varepsilon \models [\textcircled{s}]_{\mathbf{0}}(x = \mathtt{H})$ and $\varepsilon \models [\textcircled{s}]_{\mathbf{0}}(x = \mathtt{T})$ and $\mathbf{0} + \mathbf{0} < \mathbf{1}$. Importantly, zero is the tightest expectation lower bound possible here. □

8 Case Studies

In this section, we apply pDL to reason about two illustrative case studies: the Monty Hall game (Sect. 8.1), and convergence of a Bernoulli random variable (Sect. 8.2).

8.1 Monty Hall Game

In this section, we use pDL to compute the probability of winning the *Monty Hall game*. In this game, a host presents 3 doors, one of which contains a prize and the others are empty, and a contestant must figure out the door behind which the prize is hidden. To this end, the host and contestant follow a peculiar sequence of steps. First, the location of the prize is non-deterministically selected by the host. Secondly, the contestant chooses a door. Then, the host opens an empty door from those that the contestant did not choose. Finally, the contestant is asked whether she would like to switch door. We determine, using pDL, what option increases the chances of winning the prize (switching or not).

Listing 1.1 shows a pGCL program, `Monty_Hall`, modeling the behavior of host and contestant. There are 4 variables in this program: `prize` (door containing the prize), `choice` (door selected by the contestant), `open` (door opened by the host), `switch` (Boolean indicating whether the user switches in the last step). Note that the variable `switch` is undefined in the program. The value of `switch` encodes the strategy of the contestant, so its value will be part of a pDL specification that we study below. Line 1 models the hosts's non-deterministic choice of the door for the prize. Line 2 models the door choice of the contestant (uniformly over the 3 doors). Lines 3-6 model the selection of the door to open, from the non-selected doors by the contestant. Lines 7-10 model whether the contestant switches door or not. For clarity and to reduce the size of the program, in lines 6 and 8, we use a shortcut to compute the door to open and to switch, respectively. Note that for $x, y \in \{0, 1, 2\}$ the expression $z = (2x - y) \mathbf{\ mod\ } 3$ simply returns $z \in \{0, 1, 2\}$ such that $z \neq x$ and $z \neq y$. Similarly, in line 4, the expressions $y = (x + 1) \mathbf{\ mod\ } 3, z = (x + 2) \mathbf{\ mod\ } 3$ ensure that $y \neq x$, $z \neq x$ and $y \neq z$. This shortcut computes the doors that the host may open when the contestant's choice (line 2) is the door with the prize.

Listing 1.1. Monty Hall Program (`Monty_Hall`)

```
1  prize := 0 ⊓ (prize := 1 ⊓ prize := 2);
2  choice := 0 1/3⊕ (choice:=1 1/2⊕ choice:=2);
3  if (prize = choice)
4     open := (prize+1)%3 ⊓ open := (prize+2)%3;
```

```
5   else
6      open := (2*prize-choice)%3;
7   if (switch)
8      choice := (2*choice-open)%3
9   else
10     skip
```

We use pDL to find out the probability of the contestant selecting the door with the prize. To this end, we check satisfaction of the following formula, and solve it for $\boldsymbol{p}$.

$$\varepsilon[\mathit{switch} \mapsto \mathit{true}] \models [\texttt{Monty\_Hall}]_{\boldsymbol{p}}(\mathit{choice} = \mathit{prize}). \tag{9}$$

First, we show that $\boldsymbol{p} = \min(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2)$ where each $\boldsymbol{p}_i$ is the probability for the different locations of the prize. Formally, we use Thm. 8.2 (twice) as follows

$$\begin{aligned}
&\varepsilon \models [\texttt{prize:=0;}\dots]_{\boldsymbol{p}_0}(\mathit{choice} = \mathit{prize}) \ \mathit{and}\\
&\varepsilon \models [\texttt{prize:=1;}\dots]_{\boldsymbol{p}_1}(\mathit{choice} = \mathit{prize}) \ \mathit{and}\\
&\varepsilon \models [\texttt{prize:=2;}\dots]_{\boldsymbol{p}_2}(\mathit{choice} = \mathit{prize}) \ \mathit{imply}\\
&\varepsilon \models [\texttt{Monty_Hall}]_{\min(\boldsymbol{p}_0,\boldsymbol{p}_1,\boldsymbol{p}_2)}(\mathit{choice} = \mathit{prize})
\end{aligned}$$

For each $\boldsymbol{p}_i$, we compute the probability for each branch of the probabilistic choice. To this end, we use Thm. 8.1 as follows:

$$\begin{aligned}
&\varepsilon \models [\texttt{choice:=0;}\dots]_{\boldsymbol{p}_{i0}}(\mathit{choice} = \mathit{prize}) \ \ \mathit{and}\\
&\varepsilon \models [\texttt{(choice:=1}\ {}_{1/2}\!\oplus \texttt{choice:=2);}\dots]_{\boldsymbol{p}_{i1}}(\mathit{choice} = \mathit{prize}) \ \ \mathit{imply}\\
&\varepsilon \models [\texttt{choice:=0}\ {}_{1/3}\!\oplus \texttt{(choice:=1}\ {}_{1/2}\!\oplus \texttt{choice:=2);}\dots]_{1/3\cdot\boldsymbol{p}_{i0}+2/3\cdot\boldsymbol{p}_{i1}}(\mathit{choice} = \mathit{prize}).
\end{aligned}$$

and apply it again for $\boldsymbol{p}_{i1}$ to resolve the inner probabilistic choice:

$$\begin{aligned}
&\varepsilon \models [\texttt{choice:=1;}\dots]_{\boldsymbol{p}_{i10}}(\mathit{choice} = \mathit{prize}) \ \ \mathit{and}\\
&\varepsilon \models [\texttt{choice:=2;}\dots]_{\boldsymbol{p}_{i11}}(\mathit{choice} = \mathit{prize}), \ \ \mathit{implies}\\
&\varepsilon \models [\texttt{(choice:=1}\ {}_{1/2}\!\oplus \texttt{choice:=2);}\dots]_{1/2\cdot\boldsymbol{p}_{i10}+1/2\cdot\boldsymbol{p}_{i11}}(\mathit{choice} = \mathit{prize})
\end{aligned}$$

These steps show that $\boldsymbol{p}_i = 1/3\cdot\boldsymbol{p}_{i0} + 2/3\cdot 1/2\cdot\boldsymbol{p}_{i10} + 2/3\cdot 1/2\cdot\boldsymbol{p}_{i11}$ where $\boldsymbol{p}_{i0}$, $\boldsymbol{p}_{i10}$ and $\boldsymbol{p}_{i11}$ are the probabilities for the paths with *choice* equals to 0, 1 and 2, respectively.

Let us focus on the case $\boldsymbol{p}_1$. This is the case when the prize is behind door 1, $\varepsilon[\mathit{prize} \mapsto 1]$. In what follows, we explore the three possible branches of the probabilistic choice. Consider the case where the user chooses door 1, i.e., $\varepsilon[\mathit{choice} \mapsto 1]$ and

$$\varepsilon \models [\texttt{if (prize = choice) \{}s_0\texttt{\} else \{}s_1\texttt{\};}\dots]_{\boldsymbol{p}_{110}}(\mathit{choice} = \mathit{prize})$$

where s_0 and s_1 correspond to lines 4 and 6 in Listing 1.1, respectively. Since $\varepsilon \models \mathit{prize} = \mathit{choice}$ holds and by Thm. 11.1 we derive that

$$\varepsilon \models [s_0\texttt{;}\dots]_{\boldsymbol{p}_{110}}(\mathit{choice} = \mathit{prize}).$$

Note that $\boldsymbol{p}_{110}$ remains unchanged. Statement s_1 contains a non-deterministic choice, so we apply Thm. 8.2 to derive $\boldsymbol{p}_{110} = \min(\boldsymbol{p}_{1100}, \boldsymbol{p}_{1101})$ where each $\boldsymbol{p}_{110i}$ correspond to the cases where $\varepsilon[\mathit{open} \mapsto 2]$ and $\varepsilon[\mathit{open} \mapsto 0]$, respectively. Since $\mathit{switch} = \mathit{true}$ both branches execute line 8, and the probabilities remain the same (Thm. 11.1). A simple calculation shows that after executing line 8 $\varepsilon \not\models (\mathit{prize} = \mathit{choice})$ for both cases. For instance, consider

$$\varepsilon[\mathit{open} \mapsto 0] \models [\texttt{choice := (2*choice-open)\%3}]_{\boldsymbol{p}_{1100}}(\mathit{prize} = \mathit{choice}).$$

By Thm. 10.3 $\varepsilon[\mathit{choice} \mapsto (2 * 1 - 0)\%3 = 2]$, which results in $\mathit{prize} \neq \mathit{choice}$. By the universal lower bound rule (Prop. 3.1) we derive $\boldsymbol{p}_{1100} = 0$. The same derivations show that $\boldsymbol{p}_{1101} = 0$, and, consequently, $\boldsymbol{p}_{110} = 0$.

The same reasoning shows that $\mathit{prize} = \mathit{choice}$ holds for the cases where $\mathit{choice} \neq 1$ in line 2, i.e., $\boldsymbol{p}_{i0}$ and $\boldsymbol{p}_{i11}$—we omit the details as they are analogous to the steps above. In these cases, by Thm. 10.1 we derive that $\boldsymbol{p}_{i0} = 1$ and $\boldsymbol{p}_{i11} = 1$. Recall that $\boldsymbol{p}_{110} = 0$ (see above), then we derive that $\boldsymbol{p}_1 = 1/3 \cdot 1 \,+\, 2/3 \cdot 1/2 \cdot 0 \,+\, 2/3 \cdot 1/2 \cdot 1$. Consequently, $\boldsymbol{p}_1 = 1/3 + 1/3 = 2/3$. Analogous reasoning shows that all $\boldsymbol{p}_i = 2/3$.

To summarize, the probability of choosing the door with the prize when switching is at most 2/3. In other words, we have proven that switching door maximizes the probability of winning the prize.

8.2 Convergence of a Bernoulli random variable

We use pDL to study the convergence of a program that estimates the expectation of a Bernoulli random variable. To this end, we compute the probability that an estimated expectation is above an error threshold $\delta > 0$. This type of analysis may be of practical value for verifying the implementation of estimators for statistical models.

Consider the following pGCL program for estimating the expected value of a Bernoulli random variable (Technically the program computes the number of successes out of n trials, and we will put the estimation into the post-condition):

Listing 1.2. Bernoulli Program (`Bernoulli`)

```
1  i := 0; c := 0;
2  while (i < n) {
3    s := 0 μ⊕ s := 1;
4    c := c + s;
5    i := i + 1
6  }
```

Intuitively, `Bernoulli` computes the average of n Bernoulli trials X_i with mean μ, i.e., $X = \sum_i X_i/n$. It is well-known that $\mathrm{E}[X] = \mu$ (e.g., [15]). Each X_i can be seen as a sample or measurement to estimate μ. A common way to study convergence is to check the probability that the estimated mean X is within some distance $\delta > 0$ of μ, i.e., $\Pr(|X - \mu| > \delta)$. In `Bernoulli`, a sample X_i corresponds to the execution of the probabilistic choice ${}_{\mu}\oplus$ in line 3 of Listing 1.2. After running all loop iterations, variable c contains the sum of all the samples, i.e., $c = \sum_i X_i$. Thus, X is equivalent to c/n and the specification of convergence can be written as $\Pr(|c/n - \mu| > \delta)$. Note that this specification is independent of the implementation of the program. The same specification can be used for any program estimating μ—by simply replacing X with the term estimating μ in the program.

In pDL, we can study the convergence of this estimator by checking

$$\varepsilon \models [\texttt{Bernoulli}]_{\boldsymbol{p}}(|c/n - \mu| > \delta)$$

for some value of $\mu \in [0;1]$, $\delta > 0$ and $n \in \mathbb{N}$. Note that, since the program contains no non-determinism, $\boldsymbol{p} = \Pr(|X - \mu| > \delta)$. We describe the reasoning to compute $\boldsymbol{p}$.

First, note that the while-loop in Bernoulli is bounded. Therefore, we can replace it with a sequence of n iterations of the loop body. Let s_i denote the ith iteration of the loop (lines 3-4 in Listing 1.2). We omit for brevity the assignments in line 1 of Listing 1.2 and directly proceed with a state $\varepsilon[c \mapsto 0, i \mapsto 0]$. Consider the first iteration of the loop, i.e., $i = 0$. By Thm. 11.3 we can derive

$$\varepsilon \models [\texttt{if (0 < n) \{}s_0\texttt{; while (i < n) \{}s_1\texttt{\}\} else \{skip\}\}}]_{\boldsymbol{p}}(|c/n - \mu| > \delta).$$

Assume $\varepsilon \models 0 < n$ holds, then by Thm. 11.1 we derive

$$\varepsilon \models [s_0\texttt{; while (i < n) \{}s_1\texttt{\}}]_{\boldsymbol{p}}(|c/n - \mu| > \delta).$$

By applying the above rules repeatedly we can rewrite Bernoulli as

$$\varepsilon \models [s_0\texttt{;}\ldots\texttt{;}s_{n-1}\texttt{;skip}]_{\boldsymbol{p}}(|c/n - \mu| > \delta)$$

with the **skip** added in the last iteration of the loop by Thm. 11.3 and 11.2.

Second, we compute the value of $\boldsymbol{p}$ for a possible path of Bernoulli. Consider the case when $c = 0$ after executing the program. That is,

$$\varepsilon \models [s_0\texttt{;}\ldots\texttt{;}s_{n-1}\texttt{;skip}]_{\boldsymbol{p}}(c = 0).$$

This only happens for the path where the probabilistic choice is resolved as `c:=0` for all loop iterations. Applying Thm. 9 we derive

$$\textit{If } \models (c = 0) \to [s_1\texttt{;}\ldots\texttt{;}s_{n-1}\texttt{;skip}]_{\boldsymbol{p}'}(c = 0), \textit{ then}$$
$$\varepsilon \models [\mathsf{s}_0;\ldots;\mathsf{s}_{n-1};\texttt{skip}]_{\mathbf{E}_\varepsilon \boldsymbol{p}' \downarrow (c=0)}(c = 0).$$

Here $\mathbf{E}_\varepsilon$ is computed over $\mathcal{M}_{s_0}$ (cf. Thm. 9). For Bernoulli, this expectation is computed over the two paths resulting from the probabilistic choice in Listing 1.2, line 3. Since only the left branch satisfies $c = 0$ and it is executed with probability μ, then $\mathbf{E}_\varepsilon \boldsymbol{p}' = \mu \boldsymbol{p}'$. Applying this argument for each iteration of the loop we derive that $\varepsilon \models [s_0\texttt{;}\ldots\texttt{;}s_{n-1}\texttt{;skip;}]_{\boldsymbol{p}}(c = 0)$ holds for $\boldsymbol{p} = \mu^n$. Similarly, consider the case where $c = 1$ after running all iterations of the loop, due to the first iteration resulting in `c:=1` and the rest `c:=0`. Then, we apply Thm. 9 as follows

$$\textit{If } \models (c = 1) \to [s_1\texttt{;}\ldots\texttt{;}s_{n-1}\texttt{;skip}]_{\boldsymbol{p}'}(c = 1), \textit{ then}$$
$$\varepsilon \models [\mathsf{s}_0;\ldots;\mathsf{s}_{n-1};\texttt{skip}]_{\mathbf{E}_\varepsilon \boldsymbol{p}' \downarrow (c=1)}(c = 1).$$

In this case, $\mathbf{E}_\varepsilon \boldsymbol{p}' = (1 - \mu)\boldsymbol{p}'$, as the probability of $c = 1$ is $(1 - \mu)$ (cf. Listing 1.2 line 3). Since, in the case, the remaining iterations of the loop result in `c:=0`, and from our reasoning above, we derive that $\boldsymbol{p}' = \mu^{\mu-1}$. Hence, $\boldsymbol{p} = (1 - \mu)\mu^{n-1}$. In general, by

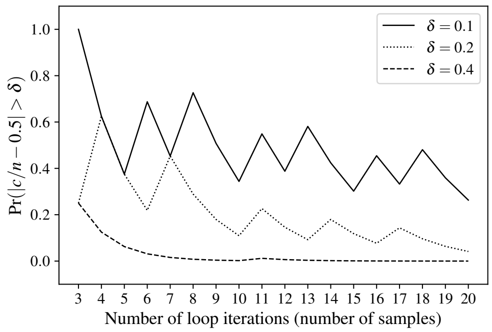

Fig. 3. Convergence of Bernoulli random variable with $\mu = 0.5$.

repeatedly applying these properties, we can derive that the probability of a path is $\mu^i(1-\mu)^j$ where i is the number loop iterations resulting in `c:=0` and j the number of loop iterations resulting in `c:=1`.

Now we return to our original problem $\varepsilon \models [\texttt{Bernoulli}]_{\boldsymbol{p}}(|c/n - \mu| > \delta)$. Recall from Def. 2 that $\boldsymbol{p}$ is the sum of the probabilities over all the paths that satisfy the post-condition. `Bernoulli` has 2^n paths (two branches per loop iteration). Therefore, we conclude that $\boldsymbol{p} = \sum_{i \in \Phi} \mu^{\mathit{zeros(i)}}(1-\mu)^{\mathit{ones(i)}}$ where $\mathit{zeros}(\cdot), \mathit{ones}(\cdot)$ are functions returning the number of zeros and ones in the binary representation of the parameter, respectively, and $\Phi = \{\, i \in 2^n \mid |\mathit{ones}(i)/n - \mu| > \delta \,\}$ enumerates all paths in program satisfying the post-condition. Note that the binary representation of $0, \ldots, 2^n$ conveniently captures each of the possible executions of `Bernoulli`.

The result above is useful to examine the convergence of `Bernoulli`. It allows us to evaluate the probability of convergence for increasing number of samples and different values of μ and δ. As an example, Fig. 3 shows the results for $\mu = 0.5, \delta \in \{0.1, 0.2, 0.4\}$ and up to $n = 20$ iterations of the loop. The dotted and dashed lines in the figure show that with 20 iterations the probability of having an error $\delta > 0.2$ is less than 5%. However, for an error $\delta > 0.1$ the probability increases to more than 20%.

9 Related Work

Probabilistic programming is applied in two main areas. First, probabilistic programs [16] are used to model complex probability distributions which are then analysed by statistical interference or Monte Carlo simulation to gain insights about the probability distribution. Second, probabilistic programs are used to express randomized algorithms [17]. In this work, we consider probabilistic programs of the second category. Verification of these probabilistic algorithms has been addressed with abstract interpretation [18], symbolic execution [19], or probabilistic model checking [20]. In this work, we focus on logical reasoning about probabilistic algorithms based on dynamic logic.

Existing dynamic logics for probabilistic programs are PPDL [21] and PrDL [22]. Kozen's PPDL [21] models states by probability distributions over variables, and propositions are measurable real-valued functions. The program semantics is a Markov chain where the transition relation is captured by transition probabilities, thus PPDL does not include demonic choice. Probabilistic Dynamic Logic PrDL [22] by Feldman and Harel relies on the same notion of state as PPDL, but introduces probabilistic transitions by a random choice operator. Similar to pDL, PPDL and PrDL use forward reasoning as standard in dynamic logics, but do not include non-determinism or an explicit probabilistic choice operator. Another major difference is that pDL models states as classical mappings of variables to values, and probabilities are introduced by probabilistic choice. Furthermore, pDL uses first-order logic to capture properties of program states.

The main alternative approach for logical reasoning about probabilistic programs is the weakest pre-expectation calculus proposed by McIver and Morgan in their probabilistic guarded command language (pGCL) [7]. pGCL contains explicit probabilistic and demonic choice. Program states are modeled by classical (non-probabilistic) variable assignments, and probabilities are introduced by an explicit probabilistic choice between states. Assertions over program states are real-valued functions capturing expectations, where a Boolean embedding is used to derive expectations from logical assertions. Reasoning in pGCL follows a backwards expectation transformer semantics. McIver and Morgan define an axiomatic semantics given by the weakest pre-expectation calculus over pGCL programs, but do not introduce an operational semantics for the language. Also they do not provide a specification language for pGCL assertions, i.e., real-valued functions, beyond the Boolean embedding (cf. [23]). pDL builds on pGCL using probabilistic and demonic choice, and relies on the same classical notion of program state. However, pDL goes beyond pGCL by providing an operational MDP semantics which naturally combines probabilistic and demonic choice, and by an explicit specification language based on dynamic logic interpreted over the MDP semantics. Furthermore, pDL allows forward reasoning for probabilistic programs.

Gretz et al. [12] proposes an MDP semantics for pGCL, similar to the MDP semantics underlying the semantics of pDL, where post-expectations are rewards in final states. Kaminski [2] presents a computation tree semantics of pGCL programs. However, both works apply those program semantics mainly for reasoning about the reachability of final states in termination proofs. Batz et al. [23] propose a specification language for real-valued functions that could be used to express assertions for pGCL programs and shows that this language is closed under the construction of weakest pre-expectations. Compared to this line of work, pDL is closer to a verification system, because it contains programs as a first class entity as in classic dynamic logic [9], in contrast to [23].

Termination analysis of probabilistic programs [1, 24] considers probabilistic reachability properties, but does not consider other program properties as pDL. Other extensions of program logics to probabilistic programs, such as separation logic for probabilistic programs [25], expected run-time analysis for probabilistic programs [26] and relational reasoning over probabilistic programs for sensitivity analysis [27], are orthogonal to pDL. Generally all these approaches rely on the backwards pre-expectation transformer semantics proposed by McIver and Morgan [7].

10 Conclusion

We have proposed pDL, a specification language for probabilistic programs that is the first dynamic logic for probabilistic programs written in pGCL. While pDL contains probabilistic and demonic choice (as in pGCL), pDL also includes programs as first-order entities in specifications and allows forward reasoning capabilities as usual in dynamic logic. We have defined the semantics of pDL based on classic MDP semantics and shown basic properties of the newly introduced p-box modality used to reason about pGCL programs. We demonstrated the reasoning capabilities with two well-known case examples. In the future, we plan to develop a deductive proof system for pDL supported by tools for (semi-)automated reasoning about pGCL programs.

References

1. Hark, M., Kaminski, B.L., Giesl, J., Katoen, J.: Aiming low is harder: induction for lower bounds in probabilistic program verification. Proc. ACM Program. Lang. **4**(POPL) (2020) 37:1–37:28
2. Kaminski, B.L.: Advanced weakest precondition calculi for probabilistic programs. PhD thesis, RWTH Aachen University, Germany (2019)
3. Kozen, D.: Semantics of probabilistic programs. In: Proc. 20th Annual Symposium on Foundations of Computer Science, IEEE Computer Society (1979) 101–114
4. Stein, D., Staton, S.: Compositional semantics for probabilistic programs with exact conditioning. In: Proc. 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021), IEEE (2021) 1–13
5. Smolka, S., Kumar, P., Foster, N., Kozen, D., Silva, A.: Cantor meets scott: semantic foundations for probabilistic networks. In Castagna, G., Gordon, A.D., eds.: Proc. 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017), ACM (2017) 557–571
6. Batz, K., Fesefeldt, I., Jansen, M., Katoen, J., Keßler, F., Matheja, C., Noll, T.: Foundations for entailment checking in quantitative separation logic. In Sergey, I., ed.: Proc. 31st European Symposium on Programming (ESOP 2022). Volume 13240 of Lecture Notes in Computer Science., Springer (2022) 57–84
7. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer (2005)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
9. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. Foundations of Computing. MIT Press (October 2000)
10. Puterman, M.L.: Markov Decision Processes. Wiley (2005)
11. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5) (1994) 512–535
12. Gretz, F., Katoen, J., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Perform. Evaluation **73** (2014) 110–132
13. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M., eds.: Deductive Software Verification - The KeY Book - From Theory to Practice. Volume 10001 of Lecture Notes in Computer Science. Springer (2016)
14. de Gouw, S., Rot, J., de Boer, F.S., Bubel, R., Hähnle, R.: OpenJDK's Java.utils.Collection.sort() is broken: The good, the bad and the worst case. In Kroening, D., Pasareanu, C.S., eds.: Proc. 27th International Conference on Computer Aided Verification (CAV 2015). Volume 9206 of Lecture Notes in Computer Science., Springer (2015) 273–289

15. Dekking, F.M., Kraaikamp, C., Lopuhaä, H.P., Meester, L.E.: A modern introduction to probability and statistics: understanding why and how. Springer texts in statistics. Springer (2005)
16. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Proceedings of the on Future of Software Engineering (FOSE 2014), ACM (2014) 167–181
17. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
18. Cousot, P., Monerau, M.: Probabilistic abstract interpretation. In: Programming Languages and Systems, Springer (2012) 169–193
19. Filieri, A., Pasareanu, C.S., Visser, W.: Reliability analysis in symbolic pathfinder. In: 35th International Conference on Software Engineering (ICSE'13), IEEE Computer Society (2013) 622–631
20. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: Ninth International Conference on Quantitative Evaluation of Systems (QEST 2012), IEEE Computer Society (2012) 203–204
21. Kozen, D.: A probabilistic PDL. J. Comput. Syst. Sci. **30**(2) (1985) 162–178
22. Feldman, Y.A., Harel, D.: A probabilistic dynamic logic. In: Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC), ACM (1982) 181–195
23. Batz, K., Kaminski, B.L., Katoen, J., Matheja, C.: Relatively complete verification of probabilistic programs: an expressive language for expectation-based reasoning. Proc. ACM Program. Lang. **5**(POPL) (2021) 1–30
24. McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.: A new proof rule for almost-sure termination. Proc. ACM Program. Lang. **2**(POPL) (2018) 33:1–33:28
25. Batz, K., Kaminski, B.L., Katoen, J., Matheja, C., Noll, T.: Quantitative separation logic: a logic for reasoning about probabilistic pointer programs. Proc. ACM Program. Lang. **3**(POPL) (2019) 34:1–34:29
26. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run-times of probabilistic programs. In Thiemann, P., ed.: Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016. Volume 9632 of Lecture Notes in Computer Science., Springer (2016) 364–389
27. Aguirre, A., Barthe, G., Hsu, J., Kaminski, B.L., Katoen, J., Matheja, C.: A pre-expectation calculus for probabilistic sensitivity. Proc. ACM Program. Lang. **5**(POPL) (2021) 1–28

A Proofs

A.1 Proofs and Auxiliary Properties for Sect. 2

Expectations in an MDP. We start with a few basic order and distribution properties of expectations that are useful in later proofs.

Lemma 15. *Given a program s, the MDP $\mathcal{M}_s$ representing its semantics, its state σ, and reward functions $\boldsymbol{r}$, $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ (into $[0;1]$ we have that*

1. $\mathbf{E}_\sigma(\boldsymbol{r})$ *is a non-negative function for any non-negative reward function* $\boldsymbol{r}$.
2. *If* $\boldsymbol{r}_1 \le \boldsymbol{r}_2$ *everywhere then* $\mathbf{E}_\sigma(\boldsymbol{r}_1) \le \mathbf{E}_\sigma(\boldsymbol{r}_2)$ *everywhere.*
3. $\mathbf{E}_\sigma(\boldsymbol{r}_1) + \mathbf{E}_\sigma(\boldsymbol{r}_2) \le \mathbf{E}_\sigma(\boldsymbol{r}_1 + \boldsymbol{r}_2)$ *everywhere.*
4. *For a constant* $c \in \mathbb{R}$ *we have that* $\max[\mathbf{E}_\sigma(\boldsymbol{r}) + c, 0] \le \mathbf{E}_\sigma[\max(\boldsymbol{r} + c, 0)]$
5. *For a constant* $c \in [0;1]$ *we have that* $c\mathbf{E}_\sigma(\boldsymbol{r}) = \mathbf{E}_\sigma[c\boldsymbol{r}]$
6. $(1 - \inf_\pi \boldsymbol{r}) = \sup_\pi (1 - \boldsymbol{r})$ *and* $(1 - \sup_\pi \boldsymbol{r}) = \inf_\pi (1 - \boldsymbol{r})$
7. $\mathbb{E}_{\sigma,\pi}(\lambda\sigma' \cdot \mathbb{E}_{\pi,\sigma'}(\boldsymbol{r})) = \mathbb{E}_{\sigma,\pi}(\boldsymbol{r})$
8. $\mathbf{E}_\sigma(\lambda\sigma' \cdot \mathbf{E}_{\sigma'}(\boldsymbol{r})) \le \mathbf{E}_\sigma(\boldsymbol{r})$

Proof (Lemma 15).

15.3 We have equality in this property if there is no non-determinism (a Markov Chain instead of an MDP). Then the property is just a regular property of expected value of random variables. For an MDP we have:

$$\begin{aligned}
\mathbf{E}_\sigma(\boldsymbol{r}_1) + \mathbf{E}_\sigma(\boldsymbol{r}_2) &= \left[\inf_\pi \mathbb{E}_{\sigma,\pi}(\boldsymbol{r}_1)\right] + \left[\inf_\pi \mathbb{E}_{\sigma,\pi}(\boldsymbol{r}_2)\right] && \text{(def. of expectation, Eq. (3))}\\
&\le \inf_\pi \left[\mathbb{E}_{\sigma,\pi}(\boldsymbol{r}_1) + \mathbb{E}_{\sigma,\pi}(\boldsymbol{r}_2)\right] && (\inf f + \inf g \le \inf(f+g))\\
&= \inf_\pi \mathbb{E}_{\sigma,\pi}(\boldsymbol{r}_1 + \boldsymbol{r}_2) && (\mathbb{E}\text{ distributes with +})\\
&= \mathbf{E}_\sigma(\boldsymbol{r}_1 + \boldsymbol{r}_2) && \text{(Eq. (3))}
\end{aligned}$$

15.4 We have equality if c is non-negative.

$$\begin{aligned}
\max\left(\mathbf{E}_\sigma(\boldsymbol{r}) + c, 0\right) &= \max\left[\left(\inf_\pi \mathbb{E}_{\sigma,\pi}(\boldsymbol{r})\right) + c, 0\right] && \text{(Eq. (3))}\\
&= \max\left[\inf_\pi \left(\mathbb{E}_{\sigma,\pi}(\boldsymbol{r}) + c\right), 0\right] && \text{(shift inf by a const.)}\\
&= \inf_\pi \max\left[\mathbb{E}_{\sigma,\pi}(\boldsymbol{r}) + c, 0\right] && \text{(argue by cases of max)}\\
&= \inf_\pi \max[\mathbb{E}_{\sigma,\pi}(\boldsymbol{r} + c), 0] && \text{(shift random variable by } c\text{)}\\
&\le \inf_\pi \max\left\{\mathbb{E}_{\sigma,\pi}[\max(\boldsymbol{r} + c, 0)], 0\right\} && \text{(truncate random var)}\\
&= \inf_\pi \mathbb{E}_{\sigma,\pi}[\max(\boldsymbol{r} + c, 0)] && (\mathbb{E}_{\sigma,\pi}[\max(\boldsymbol{r}+c,0)]\text{ nonneg.})\\
&= \mathbf{E}_\sigma[\max(\boldsymbol{r} + c, 0)] && \text{(Eq. (3))}
\end{aligned}$$

15.5 The equality follows from the fact that infimum and expected value both commute with a multiplication by a non-negative constant.

15.7 First a comment what the lambda notation in parentheses means: we mean that the random variable in the outer expected value is itself an expected value in final states of the outer Markov Chain (which are the initial states of the inner Markov Chain). As usual a random variable in a Markov Chain (or a reward in a Markov Chain) is a function that takes as an argument the state it is calculated from. We explicitly name this argument σ. Alternatively, we could have put a dot $(\cdot)$ instead of σ in the subscript of the inner expected value: $\lambda\sigma \cdot \mathbb{E}_{\pi,\sigma}(\boldsymbol{r})$ means the same as $\mathbb{E}_{\pi,\cdot}(\boldsymbol{r})$. Now that the notation is out of the way, let's prove the lemma by a sequence of equalities:

$$\begin{aligned}
&\mathbb{E}_{\sigma,\pi}(\lambda\sigma' \cdot \mathbb{E}_{\sigma',\pi}(\boldsymbol{r})) && \text{(assumption LHS)}\\
&= \sum_{\overline{\sigma}_1 \in \mathrm{paths}(\sigma)} \Pr(\overline{\sigma}_1)\mathbb{E}_{\mathrm{final}(\overline{\sigma}_1),\pi}(\boldsymbol{r}) && \text{(Eq. (3))}\\
&= \sum_{\overline{\sigma}_1 \in \mathrm{paths}(\sigma)} \Pr(\overline{\sigma}_1)\left[\sum_{\overline{\sigma}_2 \in \mathrm{paths}(\mathrm{final}(\overline{\sigma}_1))} \Pr(\overline{\sigma}_2)\boldsymbol{r}(\mathrm{final}(\overline{\sigma}_2))\right] && \text{(Eq. (3))}\\
&= \sum_{\overline{\sigma}_1 \in \mathrm{paths}(\sigma)} \sum_{\overline{\sigma}_2 \in \mathrm{paths}(\mathrm{final}(\overline{\sigma}_1))} \Pr(\overline{\sigma}_1)\Pr(\overline{\sigma}_2)\boldsymbol{r}(\mathrm{final}(\overline{\sigma}_2)) && \text{(distribute multiplication)}\\
&= \sum_{\overline{\sigma}_1\overline{\sigma}_2 \in \mathrm{paths}(\sigma)} \Pr(\overline{\sigma}_1\overline{\sigma}_2)\boldsymbol{r}(\mathrm{final}(\overline{\sigma}_1\overline{\sigma}_2)) && \text{(concatenate paths, now in } \mathcal{M}_{s_1;s_2})\\
&= \mathbb{E}_{\sigma,\pi}(\boldsymbol{r}) && \text{(Eq. (3))}
\end{aligned}$$

Note that when we expand expected values using Eq. (3) the policy π disappears in the notation—it is implicitly included in the probability mass function $\Pr$. We chose not to subscript it for readability. Also the paths set starting in σ in the penultimate row refer to the paths in $\mathcal{M}_{s_1;s_2}$ (so the set of the longer paths).

15.8 The use of the lambda notation is the same as in the previous point, so see above.

$$\begin{aligned}
&\mathbf{E}_{\sigma}(\lambda\sigma' \cdot \mathbf{E}_{\sigma'}(\boldsymbol{r})) && \text{(assumption LHS)}\\
&= \inf_{\pi} \mathbb{E}_{\sigma,\pi}\Big(\lambda\sigma' \cdot \inf_{\pi'} \mathbb{E}_{\sigma',\pi'}(\boldsymbol{r})\Big) && \text{(Eq. (3) twice)}\\
&\le \inf_{\pi} \mathbb{E}_{\sigma,\pi}(\lambda\sigma' \cdot \mathbb{E}_{\sigma',\pi}(\boldsymbol{r})) && \text{(infimum over a smaller set of policy pairs)}\\
&\le \inf_{\pi} \mathbb{E}_{\sigma,\pi}(\boldsymbol{r}) && \text{(Lemma 15.7 above)}\\
&= \mathbf{E}_{\sigma}(\boldsymbol{r}) && \text{(Eq. (3))}
\end{aligned}$$

□

Boolean embeddings (characteristic functions). For these properties it is useful to equate Boolean formulae with sets of states satisfying them (since these properties make sense for any classical logic, not necessarily PDL). The logical connectives then translate to set algebra in standard manner (conjunction is a intersection, etc.)

Lemma 16. *Consider formulae (sets of states), φ, φ_1 and φ_2. Then*

1. $[\![\varphi_1]\!] + [\![\varphi_2]\!] - 1 \leq [\![\varphi_1 \wedge \varphi_2]\!]$ *everywhere*
2. $[\![\neg\varphi]\!] = 1 - [\![\varphi]\!]$ *and* $[\![\varphi]\!] = 1 - [\![\neg\varphi]\!]$ *everywhere*

Proof (Lemma 16).

1. Consider an element of *State* which is both in φ_1 and in φ_2. Then the left hand side is 1, and so is the right hand side. The left hand side decreases by 1 or 2 for all other classes of elements, while the right hand side decreases by 1, so the inequality holds.
2. $[\![\varphi]\!]$ is a zero-one function, and the construction just inverts the assignment of zeroes and ones, which is exactly what a negation does.

□

A.2 Proofs for Sect. 5

Proof (Proposition 3).

3.1. By Lemma 15.1 and the last case of Def. 2.

3.2. By assumption $\boldsymbol{p}_2(\varepsilon) \leq \boldsymbol{p}_1(\varepsilon) \leq \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi]\!]$ for any state (the latter by the last case in Def. 2). Thus, by Def. 2, the last case again, $\varepsilon \models [s]_{\boldsymbol{p}_2}\, \varphi$.

3.3. Note that $\models \varphi_1 \wedge \varphi_2 \rightarrow \varphi_i$ and see the next point.

3.4. Note that $[\![\varphi_1]\!] \leq [\![\varphi_2]\!]$ everywhere. By the last case in Def. 2 and Lemma 15.2 we have $\boldsymbol{p}(\varepsilon) \leq \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_1]\!] \leq \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_2]\!]$, hence $\varepsilon \models [s]_{\boldsymbol{p}}\, \varphi_2$.

□

Proof (Theorem 5).

5.1. We show that satisfaction can be concluded with the last case of Def. 2:

$$\begin{aligned}
\boldsymbol{p}(\varepsilon) &= \max(\boldsymbol{p}_1(\varepsilon) + \boldsymbol{p}_2(\varepsilon) - 1, 0) \\
&\leq \max(\mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_1]\!] + \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_2]\!] - 1, 0) && (\boldsymbol{p}_i \text{ lowerbounds}) \\
&\leq \max(\mathbf{E}_{\langle\varepsilon,s\rangle}([\![\varphi_1]\!] + [\![\varphi_2]\!]) - 1, 0) && (\text{Lemma } 15.3) \\
&\leq \mathbf{E}_{\langle\varepsilon,s\rangle}(\max([\![\varphi_1]\!] + [\![\varphi_2]\!] - 1, 0)) && (\text{Lemma } 15.4) \\
&\leq \mathbf{E}_{\langle\varepsilon,s\rangle}(\max([\![\varphi_1 \wedge \varphi_2]\!], 0)) && (\text{Lemma } 16.1 \text{ and Lemma } 15.2) \\
&= \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_1 \wedge \varphi_2]\!] && (\max([\![\varphi_1 \wedge \varphi 2]\!], 0) = [\![\varphi_1 \wedge \varphi_2]\!])
\end{aligned}$$

The $\max$ operator is used to ensure that the obtained reward function is non-negative, which we wanted because we only work with rewards created by Boolean embeddings. This way the resulting expectation always has values in $[0;1]$.

5.2. Assume, without loss of generality, that $\varepsilon \models [s]_{\boldsymbol{p}_1}\, \varphi_1$ holds. We show that satisfaction can be concluded with the last case of Def. 2.

$$\begin{aligned}
\boldsymbol{p}(\varepsilon) = \min(\boldsymbol{p}_1(\varepsilon), \boldsymbol{p}_2(\varepsilon)) &\leq \boldsymbol{p}_1(\varepsilon) && (\text{minimum}) \\
&\leq \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_1]\!] && (\text{Def. } 2,\ \varepsilon \models [s]_{\boldsymbol{p}_1}\varphi_1) \\
&\leq \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\varphi_1 \vee \varphi_2]\!] && (\text{Prop. } 3.4)
\end{aligned}$$

□

Proof (Theorem 6).

6.1 First observe that for any value $v \in \operatorname{dom} l$ of a logical variable $l \in L$ we have the following *syntactic equality*:

$$([s]_{\boldsymbol{p}}\, \varphi)\, [l := v] \quad \equiv \quad [s]_{\boldsymbol{p}}\ (\varphi[l := v])$$

The two formulae are identical because the program s cannot refer to logical variables (l). Now we build the argument using this fact:

$$\begin{aligned}
&\varepsilon \models [s]_{\boldsymbol{p}}\ (\forall l \cdot \varphi) && \text{(assumption)}\\
&\text{implies } \varepsilon \models [s]_{\boldsymbol{p}}\, (\varphi[l := v]) && \text{(pick } v \in \operatorname{dom} l\text{, Lemma 15.2 as } [\![\forall l \cdot \varphi]\!] \leq [\![\varphi[l := v]]\!]\text{)}\\
&\text{iff } \varepsilon \models ([s]_{\boldsymbol{p}}\, \varphi)\, [l := v] && \text{(syntactic equality, above)}
\end{aligned}$$

Now observe that we have shown that $\varepsilon \models [s]_{\boldsymbol{p}}\ (\forall l \cdot \varphi)$ implies $\varepsilon \models ([s]_{\boldsymbol{p}}\, \varphi)\, [l := v]$ for arbitrary $v \in \operatorname{dom} l$. By Def. 2 this means that it also implies $\varepsilon \models \forall l \cdot [s]_{\boldsymbol{p}}\, \varphi$.

Consider a counter example for the opposite direction. Let ①be a program modeling a fair coin: ① ::= `x := H` $_{1/2}\oplus$ `x := T` and let $\operatorname{dom} l = \{\mathtt{H}, \mathtt{T}\}$, and the following two properties. The first property φ_1 holds in any environment, the second property φ_2 holds in no environment.

$$\varphi_1 \equiv \forall l \cdot [①]_{1/2}\, (x = l) \qquad\qquad \varphi_2 \equiv [①]_{1/2}(\forall l \cdot x = l) \tag{10}$$

6.2 We first prove the positive case of the theorem:

$$
\begin{array}{lr}
\varepsilon \models \exists l \cdot [s]_{\boldsymbol{p}}\, \varphi & \text{(assumption)}\\
\text{iff } \varepsilon \models \neg\forall l \cdot \neg[s]_{\boldsymbol{p}}\, \varphi & \text{(syntactic sugar)}\\
\text{iff not } \varepsilon \models \forall l \cdot \neg[s]_{\boldsymbol{p}}\, \varphi & \text{(Def. 2)}\\
\text{iff not for all } v \in \operatorname{dom} l\text{: } \varepsilon \models \neg[s]_{\boldsymbol{p}}\, (\varphi[l := v]) & \text{(Def. 2, synt. equality above)}\\
\text{iff not for all } v \in \operatorname{dom} l \text{ not: } \varepsilon \models [s]_{\boldsymbol{p}}\, (\varphi[l := v]) & \text{(Def. 2)}\\
\text{iff exists } v \in \operatorname{dom} l\text{: } \boldsymbol{p}(\varepsilon) \leq \inf_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\varphi[l := v]]\!] & \text{(Def. 2, meta-exists)}\\
\text{iff exists } v \in \operatorname{dom} l\text{: } \boldsymbol{p}(\varepsilon) \leq 1 - (1 - \inf_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\varphi[l := v]]\!]) & \text{(algebra)}\\
\text{iff exists } v \in \operatorname{dom} l\text{: } \boldsymbol{p}(\varepsilon) \leq 1 - \sup_{\pi} \big(1 - \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\varphi[l := v]]\!]\big) & \text{(Lemma 15.6)}\\
\text{iff exists } v \in \operatorname{dom} l\text{: } \boldsymbol{p}(\varepsilon) \leq 1 - \sup_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}(1 - [\![\varphi[l := v]]\!]) & \text{(sum expected vals)}\\
\text{iff exists } v \in \operatorname{dom} l\text{: } \boldsymbol{p}(\varepsilon) \leq 1 - \sup_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\neg\varphi[l := v]]\!] & \text{(Lemma 16.2)}\\
\text{implies exists } v \in \operatorname{dom} l\text{: } \boldsymbol{p}(\varepsilon) \leq 1 - \sup_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\forall l \cdot \neg\varphi]\!] & \\
 & \text{(as } [\![\forall l \cdot \neg\varphi]\!] \leq [\![\neg\varphi[l := v]]\!] \text{ for any } v)\\
\text{iff } \boldsymbol{p}(\varepsilon) \leq 1 - \sup_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\forall l \cdot \neg\varphi]\!] & \text{(drop free quantifier)}\\
\text{iff } \boldsymbol{p}(\varepsilon) \leq \inf_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}(1 - [\![\forall l \cdot \neg\varphi]\!]) & \text{(Lemma 15.6)}\\
\text{iff } \boldsymbol{p}(\varepsilon) \leq \inf_{\pi} \mathbb{E}_{\langle\varepsilon,s\rangle,\pi}[\![\neg\forall l \cdot \neg\varphi]\!] & \text{(Lemma 16.2)}\\
\text{iff } \boldsymbol{p}(\varepsilon) \leq \mathbf{E}_{\langle\varepsilon,s\rangle}[\![\exists l \cdot \varphi]\!] & \text{(syntactic sugar, Eq. (3))}\\
\text{iff } \varepsilon \models [s]_{\boldsymbol{p}}(\exists l \cdot \varphi) & \text{(Def. 2)}
\end{array}
$$

Consider a counter example for the opposite direction. Recall the program ⓢ modeling a "non-deterministic coin:" ⓢ ::= `x: = H` $\sqcup$ `x := T` and $\operatorname{dom} l = \{\mathtt{H}, \mathtt{T}\}$, and the following two properties. The first property φ_1 holds in any environment, the second property φ_2 holds in no environment.

$$\varphi_1 \equiv [ⓢ]_{\mathbf{1}}(\exists l \cdot x = l) \qquad \varphi_2 \equiv \exists l \cdot [ⓢ]_{\mathbf{1}}(x = l) \tag{11}$$

□

Proof (Theorem 7). We prove the co-occurrence of the two validities by splitting in cases, based on whether a particular valuation ε satisfies the precondition or not.

Case 1: $\varepsilon \models \varphi_1$ (works in both directions):

$$
\begin{array}{lr}
\varepsilon \models \varphi_1 \to ([s]_{\boldsymbol{p}}\, \varphi_2) & \text{(thm assumption)}\\
\varepsilon \models [s]_{\boldsymbol{p}}\, \varphi_2 & \text{(case)}\\
\varepsilon \models [s]_{\boldsymbol{p}\downarrow\varphi_1}\, \varphi_2 & ((\boldsymbol{p} \downarrow \varphi_1)(\varepsilon) = \boldsymbol{p}(\varepsilon))
\end{array}
$$

Case 2: not $\varepsilon \models \varphi_1$ then the left-hand-side holds vacuously. The right-hand-side also holds because the expectation is zero $(\boldsymbol{p} \downarrow \varphi_1)(\varepsilon) = \boldsymbol{p}(\varepsilon) \cdot ([\![\varphi_1]\!](\varepsilon)) = \boldsymbol{p}(\varepsilon) \cdot 0 = 0$ which means the formula holds by Prop. 3.1. □

A.3 Proofs for Sect. 6

Proof (Theorem 8).

8.1 Let i denote the program $s_1 \,{}_e\!\oplus\, s_2$. From left to right, we need to show that $\varepsilon(e)\boldsymbol{p}_1(\varepsilon)+(1-\varepsilon(e))\boldsymbol{p}_2(\varepsilon) \leq \mathbf{E}_{\langle\varepsilon,i\rangle}[\![\varphi]\!]$ where the expectation is taken in $\mathcal{M}_{s_1 {}_e\oplus s_2}$.

$$
\begin{aligned}
&\varepsilon(e)\boldsymbol{p}_1(\varepsilon) + (1-\varepsilon(e))\boldsymbol{p}_2(\varepsilon)\\
&\leq \varepsilon(e)\mathbf{E}_{\langle\varepsilon,s_1\rangle}[\![\varphi]\!] + (1-\varepsilon(e))\mathbf{E}_{\langle\varepsilon,s_2\rangle}[\![\varphi]\!] && \text{(left in } \mathcal{M}_{s_1}\text{, right in } \mathcal{M}_{s_2} \text{ by assumption)}\\
&= \mathbf{E}_{\langle\varepsilon,s_1\rangle}[\varepsilon(e)[\![\varphi]\!]] + \mathbf{E}_{\langle\varepsilon,s_2\rangle}[(1-\varepsilon(e))[\![\varphi]\!]] && \text{(Lemma 15.5 twice)}\\
&\leq \mathbf{E}_{\langle\varepsilon,i\rangle}\Big[\varepsilon(e)[\![\varphi]\!]^{(1)} + (1-\varepsilon(e))[\![\varphi]\!]^{(2)}\Big] && \text{(Lemma 15.3, } [\![\varphi]\!]^{(k)} \text{positive only for } s_k \text{ states)}\\
&= \mathbf{E}_{\langle\varepsilon,i\rangle}[\![\varphi]\!] && \text{(now in } \mathcal{M}_{s_1 {}_e\oplus s_2} \text{ extending expected value as per } \textsc{ProbChoice}1/2)
\end{aligned}
$$

8.2 Let i denote the program $s_1 \sqcap s_2$. For the proof from left to right, we need to show that $\boldsymbol{p}(\varepsilon) \leq \mathbf{E}_\varepsilon[\![\varphi]\!]$ where the expectation is taken in the $\mathcal{M}_{s_1 \sqcap s_2}$:

$$
\begin{aligned}
&\boldsymbol{p}(\varepsilon)\\
&\leq \min\{\mathbf{E}_{\langle\varepsilon,s_1\rangle}[\![\varphi]\!], \mathbf{E}_{\langle\varepsilon,s_2\rangle}[\![\varphi]\!]\} && \text{(left in } \mathcal{M}_{s_1}\text{, right in } \mathcal{M}_{s_2} \text{ by assumption)}\\
&= \min\{\inf_{\pi_1} \mathbb{E}_{\langle\varepsilon,s_1\rangle,\pi_1}[\![\varphi]\!], \inf_{\pi_2} \mathbb{E}_{\langle\varepsilon,s_2\rangle,\pi_2}[\![\varphi]\!]\} && \text{(left in } \mathcal{M}_{s_1}\text{, right in } \mathcal{M}_{s_2} \text{ by Eq. (3))}\\
&= \inf\{\inf_{\pi_1} \mathbb{E}_{\langle\varepsilon,s_1\rangle,\pi_1}[\![\varphi]\!], \inf_{\pi_2} \mathbb{E}_{\langle\varepsilon,s_2\rangle,\pi_2}[\![\varphi]\!]\} && \text{(inf is minimum in a finite set)}\\
&= \inf_{\pi} \mathbb{E}_{\langle\varepsilon,i\rangle,\pi}[\![\varphi]\!] && \text{(inf of infima, } \textsc{DemChoice} \text{ has two policies, no } \varepsilon \text{ change)}\\
&= \mathbf{E}_{\langle\varepsilon,i\rangle}[\![\varphi]\!] && \text{(in } \mathcal{M}_{s_1\sqcap s_2}\text{, Eq. (3))}
\end{aligned}
$$

The argument in the opposite direction follows by the same equalities backwards, and then the fact that a lower bound of a minimum is small than each element in a set.

□

Proof (Theorem 9). Let i denote the program $s_1; s_2$. To show that

$$\varepsilon \models [s_1; s_2]_{\mathbf{E}_{\langle\varepsilon,s_1\rangle}(\boldsymbol{p}\downarrow\varphi_1)}\, \varphi_2$$

we need to demonstrate that

$$\mathbf{E}_{\langle\varepsilon,s_1\rangle}(\boldsymbol{p} \downarrow \varphi_1) \leq \mathbf{E}_{\langle\varepsilon,i\rangle}[\![\varphi_2]\!] \quad ,$$

where the left expectation is taken in $\mathcal{M}_{s_1}$ and the right expectation is taken in $\mathcal{M}_{s_1;s_2}$. We use the theorem's assumption $\models \varphi_1 \rightarrow ([s_2]_{\boldsymbol{p}}\, \varphi_2)$ to get $\models [s_2]_{\boldsymbol{p}\downarrow\varphi_1}\, \varphi_2$ by Thm. 7. This in turn means that $(\boldsymbol{p} \downarrow \varphi_1)(\varepsilon') \leq \mathbf{E}_{\langle\varepsilon',s_2\rangle}[\![\varphi_2]\!]$ for all ε' (with the latter expectation taken in $\mathcal{M}_{s_2}$).

$$
\begin{aligned}
&\mathbf{E}_{\langle\varepsilon,s_1\rangle}(\boldsymbol{p} \downarrow \varphi_1)\\
&\leq \mathbf{E}_{\langle\varepsilon,s_1\rangle}\big(\lambda\varepsilon' \cdot \mathbf{E}_{\langle\varepsilon',s_2\rangle}[\![\varphi_2]\!]\big) && \text{(because } \models [s_2]_{\boldsymbol{p}\downarrow\varphi_1}\, \varphi_2 \text{ and Def. 2)}\\
&\leq \mathbf{E}_{\langle\varepsilon,i\rangle}[\![\varphi_2]\!] && \text{(Lemma 15.8)}
\end{aligned}
$$

□

Proof (Theorem 10).

10.1 Let us start from the left hand side, so assume that $\varepsilon \models [\texttt{skip}]_{\mathbf{1}}\, \varphi$. Then

$$\begin{aligned}
1 &= \mathbf{1}(\varepsilon) \\
&\le \mathbf{E}_{\langle \varepsilon, \texttt{skip} \rangle} [\![\varphi]\!] && \text{(Def. 2)} \\
&= \inf_{\pi} \mathbb{E}_{\langle \varepsilon, \texttt{skip} \rangle, \pi} [\![\varphi]\!] && \text{(Eq. (3))} \\
&= \mathbb{E}_{\langle \varepsilon, \texttt{skip} \rangle} [\![\varphi]\!] && \text{(}\texttt{skip}\text{ deterministic, single policy)} \\
&= \sum_{\overline{\sigma} \in \text{paths}(\langle \varepsilon, \texttt{skip} \rangle)} \Pr(\overline{\sigma}) \cdot ([\![\varphi]\!](\text{final}(\overline{\sigma}))) && \text{(Eq. (3))} \\
&= \sum_{\overline{\sigma} \in \text{paths}(\langle \varepsilon, \texttt{skip} \rangle)} \Pr(\overline{\sigma}) \cdot ([\![\varphi]\!](\varepsilon)) && \text{(definition of final, for pDL } [\![\varphi]\!](\langle \varepsilon, \texttt{skip} \rangle) = [\![\varphi]\!](\varepsilon)\text{)} \\
&= 1 \cdot [\![\varphi]\!](\varepsilon) && \text{(only one final path, singleton size)}
\end{aligned}$$

We have shown that $1 \le [\![\varphi]\!](\varepsilon)$, which by definition of characteristic functions means that $\varepsilon \models \varphi$. For the opposite direction, take $\varepsilon \models \varphi$ and use the above six equalities from the bottom to show that $\mathbf{1}(\varepsilon) = \mathbf{E}_{\langle \varepsilon, \texttt{skip} \rangle} [\![\varphi]\!]$, which means $\varepsilon \models [\texttt{skip}]_{\mathbf{1}}\, \varphi$.

10.2 One could argue from Thm. 9, but this is cumbersome, as it requires weakening the left-hand side to validity, which is not needed for **skip**, a special deterministic case. Thus it is better to prove directly from definition. The key step is that the infimum over all policies π for s is the same for $\texttt{skip}; s$, because skip does not change valuation and the set of policies, see rule COMPOSITION1; so in this case neither the reward valuation or the policy can be chosen differently. The argument works in both directions.

10.3 To prove this case we observe that executing a single assignment statement (see ASSIGN) does not change the probability of final paths in the MDP associated with the program, so it does not change the expectation of the formula as long as the program is executed in the same valuation that the assignment creates. This argument works in both directions:

$$\begin{aligned}
&\varepsilon \models [\mathtt{x}\!:=\!e; s]_{\boldsymbol{p}}\, \varphi \\
\text{iff } &\varepsilon[x \mapsto \varepsilon(e)] \models [\texttt{skip}; s]_{\boldsymbol{p}}\, \varphi && \text{(ASSIGN does not change probability of paths)} \\
\text{iff } &\varepsilon[x \mapsto \varepsilon(e)] \models [s]_{\boldsymbol{p}}\, \varphi && \text{(Thm. 10.2)}
\end{aligned}$$

□

Proof (Theorem 11).

11.1 Let's write i for the program $i = \texttt{if}\ e\ \{s_1\}\ \texttt{else}\ \{s_2\}$. We want to show that $\boldsymbol{p}(\varepsilon) \le \mathbf{E}_{\langle \varepsilon, i \rangle} [\![\varphi]\!]$ where the expectation is taken in the MDP $\mathcal{M}_i$. To do this we will reduce the calculation of the expectation to the MDPs $\mathcal{M}_{s_1}$ and $\mathcal{M}_{s_2}$ with the prefix for resolving the condition:

$$
\begin{aligned}
&\boldsymbol{p}(\varepsilon) \\
&\leq \mathbf{E}_{\langle\varepsilon,s_1\rangle}[\![\varphi]\!] && \text{(in } \mathcal{M}_{s_1}\text{, assumption and Def. 2 twice)} \\
&= \inf_{\pi} \mathbb{E}_{\langle\varepsilon,s_1\rangle}[\![\varphi]\!] && \text{(in } \mathcal{M}_{s_1}\text{, Eq. (3))} \\
&= \inf_{\pi} \sum_{\sigma\in\text{paths}(\langle\varepsilon,s_1\rangle)} \Pr(\sigma)\cdot[\![\varphi]\!](\text{final}(\sigma)) && \text{(Eq. (3), path prob. taken under } \pi\text{)} \\
&= \inf_{\pi} \left[1\cdot \sum_{\sigma\in\text{paths}(\langle\varepsilon,s_1\rangle)} \Pr(\sigma)\cdot[\![\varphi]\!](\text{final}(\sigma))\right] && \text{(trivial, also below)} \\
&= \inf_{\pi} \left[1\cdot \sum_{\sigma\in\text{paths}(\langle\varepsilon,s_1\rangle)} \Pr(\sigma)\cdot[\![\varphi]\!](\text{final}(\sigma)) + 0\cdot \sum_{\sigma\in\text{paths}(\langle\varepsilon,s_2\rangle)} \Pr(\sigma)\cdot[\![\varphi]\!](\text{final}(\sigma))\right] \\
&= \inf_{\pi} \sum_{\sigma\in\text{paths}(\langle\varepsilon,i\rangle)} \Pr(\sigma)\cdot[\![\varphi]\!](\text{final}(\sigma)) && \text{(I{\small F}1, assumption that } \varepsilon\models e\text{, Def. 2)} \\
&= \mathbf{E}_{\langle\varepsilon,i\rangle}[\![\varphi]\!] && \text{(in } \mathcal{M}_i\text{)}
\end{aligned}
$$

Notice the shift of the MDP in the penultimate line to $\mathcal{M}_i$ in the sum index. A line above, the assumption that the atomic formula e holds allows us to extend the expectation by the suitable choice of the if-branch. Since the if condition evaluates to true, the rule IF1 creates an MDP prefix advancing with probability one to state $\langle\varepsilon, s_1\rangle$ and with probability zero to the other state (we assume the same evaluation semantics for ATFin the logic and for expressions in pGCL). Afterwards we just observe that this is the same as calculating expected values directly from the if head.

11.2 The proof is symmetric to the previous case, just with the other IF rule.

11.3 This simple rule follows directly from operational semantics rules WHILE-1 and WHILE-2:

$$\boldsymbol{p} \leq \mathbf{E}_{\langle\varepsilon,i_1\rangle}[\![\varphi]\!] = \mathbf{E}_{\langle\varepsilon,i_2\rangle}[\![\varphi]\!] \tag{12}$$

where $i_1 =$ `if` e `{` s`;` `while` e `{` s `}}` `else {skip}` and $i_2 =$ `while` e `{` s `}`. The equality of expectations holds because the two MDPs are identical—represent the same tree of terminating expectations over the same states, with the same branching structure, and final states. (Recall we only reason about almost surely terminating programs.) The IF-1 rule on the first program has exactly the same effect (and successors) as WHILE-1 rule on the second program. Similarly both IF-2 on the first and WHILE-2 on the second program reduce to `skip` with the same valuation.

□

A.4 Proofs for Sect. 7

Proof (Theorem 13). Let s be a pGCL program, ε stand for a valuation, $\boldsymbol{p} \in \mathit{State} \to [0; 1]$ be an expectation function, π a policy resolving non-determinism in s, and $\varphi_i \in$ pDL properties. The expected reward $\boldsymbol{p}$ after s for $[\![\varphi_1 \vee \varphi_2]\!]$ under the scheduler π is a lower-bound for the sum of the separate expectations of φ_1 and φ_2, formally:

$$\boldsymbol{p} = \mathbb{E}_{\langle \varepsilon, s \rangle, \pi}[\![\varphi_1 \vee \varphi_2]\!] \leq \mathbb{E}_{\langle \varepsilon, s \rangle, \pi}[\![\varphi_1]\!] + \mathbb{E}_{\langle \varepsilon, s \rangle, \pi}[\![\varphi_2]\!] \ . \tag{13}$$

This is because in the rightmost sum, some of the states can be counted twice: once for φ_1 and once for φ_2.

Note that, if s is purely probabilistic (does not use the demonic choice operator) then the above can be written equivalently as follows: If $\varepsilon \models [s]_{\boldsymbol{p}} (\varphi_1 \vee \varphi_2)$ then there exist $\boldsymbol{p}_1, \boldsymbol{p}_2$ such that $\varepsilon \models [s]_{\boldsymbol{p}_1} \varphi_1$ **and** $\varepsilon \models [s]_{\boldsymbol{p}_2} \varphi_2$ and $\boldsymbol{p}_1 + \boldsymbol{p}_2 \geq \boldsymbol{p}$ everywhere. $\square$