Performance Evaluation 73 (2014) 110–132

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Operational versus weakest pre-expectation semantics for the probabilistic guarded command language

Friedrich Gretz[a,b,*], Joost-Pieter Katoen[a], Annabelle McIver[b]

[a] RWTH Aachen University, Aachen, Germany

[b] Macquarie University, Sydney, Australia

ARTICLE INFO

Article history:
Available online 17 December 2013

Keywords:
Expectation transformer semantics
Operational semantics
Markov decision process
Expected rewards

ABSTRACT

This paper proposes a simple operational semantics of pGCL, Dijkstra's guarded command language extended with probabilistic choice, and relates this to pGCL's *wp*-semantics by McIver and Morgan. Parametric Markov decision processes whose state rewards depend on the post-expectation at hand are used as the operational model. We show that the *weakest pre-expectation* of a pGCL-program w.r.t. a post-expectation corresponds to the *expected cumulative reward* to reach a terminal state in the parametric MDP associated to the program. In a similar way, we show a correspondence between weakest *liberal* pre-expectations and *liberal* expected cumulative rewards. The verification of probabilistic programs using *wp*-semantics and operational semantics is illustrated using a simple running example.

1. Introduction

Formal semantics of programming languages has been the subject of intense research in computer science for several decades. Various approaches have been developed for the description of program semantics. Structured operational semantics defines the meaning of a program by means of an abstract machine where states correspond to program configurations (typically consisting of a program counter and a variable valuation) and transitions model the evolution of a program by executing statements. Program executions are then the possible runs of the abstract machine. Denotational semantics maps a program onto a mathematical object that describes for instance its input–output behaviour. Finally, axiomatic semantics provides the program semantics in an indirect manner by describing its properties. A prominent example of the latter are Hoare triples in which annotations, written in predicate logic, are associated to control points of the program.

The semantics of Dijkstra's seminal guarded command language [1] from the seventies is given in terms of weakest preconditions. It is in fact a predicate transformer semantics, i.e. a total function between two predicates on the state of a program. The predicate transformer $E = wp(P, F)$ for program P and postcondition F yields the weakest precondition E on the initial state of P ensuring that the execution of P terminates in a final state satisfying F. There is a direct relation with axiomatic semantics: the Hoare triple $\langle E\rangle\, P\, \langle F\rangle$ holds for total correctness if and only if $E \Rightarrow wp(P, F)$. The weakest *liberal* precondition $wlp(P, F)$ yields the weakest precondition for which P either does not terminate or establishes F. It does not ensure termination and corresponds to Hoare logic in partial correctness. Although providing an operational semantics for the guarded command language is rather straightforward, it was not until the early nineties that Lukkien [2,3] provided a formal connection between the predicate transformer semantics and the notion of a computation.

* Corresponding author at: Macquarie University, Sydney, Australia.
E-mail addresses: friedrich.gretz@students.mq.edu.au, fgretz@cs.rwth-aachen.de (F. Gretz), katoen@cs.rwth-aachen.de (J.-P. Katoen), annabelle.mciver@mq.edu.au (A. McIver).

0166-5316/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.peva.2013.11.004

Qualitative annotations in predicate calculus are often insufficient for probabilistic programs as they cannot express quantities such as expectations over program variables. To that end, McIver and Morgan [4] generalised the methods of Dijkstra and Hoare to probabilistic programs by making the annotations real-valued expressions – referred to as expectations – in the program variables. Expectations are the quantitative analogue of predicates. This yields an expectation transformer semantics of the probabilistic guarded command language (pGCL, for short), an extension of Dijkstra's language with a probabilistic choice operator. An expectation transformer is a total function between two expectations on the state of a program. The expectation transformer $e = wp(P, f)$ for pGCL-program P and post-expectation f over final states yields the least expected value e on P's initial state ensuring that P's execution terminates with a value f. The annotation $\langle e \rangle\ P\ \langle f \rangle$ holds for total correctness if and only if $e \leq wp(P, f)$, where $\leq$ is to be interpreted in a point-wise manner. The weakest *liberal* pre-expectation $wlp(P, f)$ yields the least expectation for which P either does not terminate or establishes f. It does not ensure termination and corresponds to partial correctness.

This paper provides a simple operational semantics of pGCL using parametric Markov decision processes (pMDPs), a slight variant of MDPs in which probabilities may be parameterised [5,6]. Our main contribution in this paper is a formal connection between the *wp*- and *wlp*-semantics of pGCL by McIver and Morgan and the operational semantics of pGCL. This provides a clean and insightful relationship between the abstract expectation transformer semantics that has been proven useful for formal reasoning about probabilistic programs, and the notion of a computation in terms of the operational model, a pMDP. In order to establish this connection we equip pMDPs with state rewards that depend on the post-expectation at hand. Intuitively speaking, we decorate a terminal state in the operational model of a program with a reward that corresponds to the value of the post-expectation. All other states are assigned reward zero. We then show that the *weakest pre-expectation* of a pGCL-program P w.r.t. a post-expectation corresponds to the *expected cumulative reward* to reach a terminal state in the pMDP associated to P. In a similar way, we show that weakest *liberal* pre-expectations correspond to *liberal* expected cumulative rewards. The proofs are by induction on the structure of our probabilistic programs using standard results from fixed point theory. This paper thus yields a correspondence theorem that enables us to understand the mathematically involved expectation transformers intuitively using only first principles of Markov decision processes with rewards. In addition, for finite-state programs (or program fragments), our result implies that algorithms for computing expected accumulated rewards in MDPs – for which efficient algorithms and tools based on linear programming exist – can be employed for computing weakest pre-expectations. Finally we recall the notion of probabilistic invariants [4] and apply our correspondence theorem to find an operational characterisation of invariants (which originally are defined in terms of expectation transformers).

1.1. Related work

The MDP semantics of pGCL in this paper bears strong resemblance to the operational semantics of similar languages. To mention a few, Baier et al. [7] provide an MDP semantics of a probabilistic version of Promela, the modelling language of the SPIN model checker. Di Pierro et al. [8] give a semantics to a very similar programming language without non-determinism. The seminal work by Kozen [9] provides two semantics of a deterministic variant of pGCL and shows their correspondence. Kozen interprets probabilistic programs as partial measurable functions on a measurable space, and as continuous linear operators on a Banach space of measures. He et al. [10] provide a mapping from a semantics based on a probabilistic complete partial order which contains non-determinism à la Jones [11] to a semantics which is a mapping from initial states to sets of probability distributions over final states. To our knowledge, our results on relating weakest pre-expectations of pGCL and an operational semantics are novel. Our set-up and results can be considered as a probabilistic analogue of the work by Lukkien [2,3] who provided a formal connection between the predicate transformer semantics of Dijkstra's guarded command language and the operational notion of a computation.

More examples of how to discover and apply invariants when reasoning about probabilistic programs can be found at [12]. There we also describe Prinsys, a tool for semi-automatic invariant generation.

1.2. Structure of this paper

The rest of the paper is divided as follows. In Section 2 we introduce the probabilistic programming language pGCL. Parametric Markov decision processes with rewards are introduced in Section 3. Section 4 recaps the denotational semantics of pGCL [4] and introduces operational semantics for this language. Then the main result is established, namely that the two semantics are equivalent. Section 5 provides an example of reasoning over pGCL programs. Finally, Section 6 introduces invariants and uses our main result to give an operational characterisation for them.

This paper is an extended version of the conference paper [13]. This version contains a generalised version of the proofs of Theorems 23 and 24, a new section on invariants and an Appendix with a new proof for continuity of $wp(P, \cdot)$ and $wlp(P, \cdot)$.

2. Probabilistic programs

Our programming language pGCL [4] is an extension of Dijkstra's guarded command language [1]. Besides a non-deterministic choice operator, denoted [], and a conditional choice, it incorporates a probabilistic choice operator, denoted

Listing 1: The duelling cowboys, cf. [4].

```
(t := A [] t := B);
c := 1;
while (c = 1) {
  if (t = A) {
    (c := 0 [a] t := B);
  } else {
    (c := 0 [b] t := A);
  }
}
```

[p], where p is a real parameter (or constant) whose value lies in the range [0, 1]. pGCL is a language to model sequential programs containing randomised assignments. For instance, the assignment ($x := 2 \cdot x\,[0.75]\,x := x + 1$) doubles the value of x with probability $\frac{3}{4}$ and increments it by one with the remaining probability $\frac{1}{4}$.

Definition 1 (*Syntax of* pGCL)**.** Let P, P_1, P_2 be pGCL-programs, p a probability variable, x a program variable, E an expression, and G a Boolean expression. The syntax of a pGCL program P adheres to the following grammar:

$$\mathsf{skip} \mid \mathsf{abort} \mid x := E \mid P_1; P_2 \mid P_1\,[]\,P_2 \mid P_1\,[p]\,P_2 \mid$$
$$\mathsf{if}(G)\{P_1\}\,\mathsf{else}\,\{P_2\} \mid \mathsf{while}(G)\{P\}. \quad \Box$$

skip stands for the empty statement, abort for abortion, and $x := E$ for an assignment of the value of expression E (over the program variables) to variable x. The sequentially composed program $P_1; P_2$ behaves like P_1 and subsequently like P_2 on the successful termination of P_1. The statement $P_1\,[]\,P_2$ denotes a non-deterministic choice; it behaves like either P_1 or P_2. The statement $P_1\,[p]\,P_2$ denotes a probabilistic choice. It behaves like P_1 with probability p and like P_2 with probability $1 - p$. The remaining two statements are standard: conditional choice and while-loop. Throughout this paper, we assume that pGCL-programs are well-typed. This entails that for assignments of the form $x := E$ we assume that x and E are of the same type. We assume G to denote a Boolean expression and variable p to denote a probability in the real interval [0, 1].

Example 2 (*Duelling Cowboys [4]*)**.** The pGCL program in Lst. 1 models the following situation: There are two cowboys, A and B, who are fighting a classical duel. They take turns, shooting at each other until one of them is hit. If A (resp. B) shoots then he hits B (resp. A) with probability a (resp. b). We assume that either cowboy A or B is allowed to start; the choice of who will start is resolved nondeterministically. Variable t keeps track of the turns, while c determines whether the duel continues or someone is hit. Note that it is a distinctive feature that we do not have to specify exact probabilities and instead allow arbitrary parameters such as a and b. $\Box$

3. Markov decision processes

This section introduces the basics of Markov decision processes (MDPs) [5,6] enriched with state rewards. We first recall the definition of an MDP with a countable state space and define elementary notions such as paths and policies. Subsequently, we introduce reward-MDPs in which states are equipped with a real valued reward and focus on reachability objectives, in particular (liberal) expected cumulative rewards to reach a set of states. These measures are later shown to closely correspond to the weakest (liberal) pre-condition semantics of pGCL-programs.

3.1. Preliminaries

Let *Var* be a set of variables. In the following we consider a countable state space S where each state s is a valuation which maps variables to real values

$$S : Var \to \mathbb{R}.$$

This approach later allows us to nicely connect states of a program to states of a transition system like an MDP (as defined below). Additionally we fix a set of parameters *Par* which are independent of states. These parameters will represent probabilities which can be left unspecified. So each parameter represents a value from the interval [0, 1]. Let $V(Par)$ denote the set of expressions over *Par*.

Definition 3 (*Parametric Distribution*)**.** A parametric distribution μ is a function that maps states to probabilities. The probabilities are real values in [0, 1] or expressions over *Par*:

$$\mu : S \to V(Par) \cup [0, 1] \quad \text{with} \sum_{s \in S} \mu(s) = 1.$$

The set of all parametric distributions over state space S is denoted Dist(S). $\Box$

Example 4 (*Parametric Distribution*)**.** Consider $S = \{s_0, s_1, s_2\}$. Then a parametric distribution μ might be: $\mu(s_0) = p$, $\mu(s_1) = 1 - p$ and $\mu(s_2) = 0$ where $p \in V(Par)$ is an expression that consists of just a single parameter. The expression's value is fixed but unknown. The parameter p can be refined by any value from $[0, 1]$. □

It is important to stress that Definition 3 requires that for any state s the resulting expression $\mu(s)$ is a probability, i.e. an expression that evaluates to a value in $[0, 1]$ for all possible states s and all possible parameter valuations.

Definition 5 (*Parametric Markov Decision Process*)**.** A pMDP $\mathcal{M}$ is a tuple $(S, S_0, \rightarrow)$ where S is a countable set of states with initial state-set $S_0 \subseteq S$ where $S_0 \neq \emptyset$, and $\rightarrow \subseteq S \times Dist(S)$ is a transition relation from a state to a set of parametric distributions over states. □

Let $s \rightarrow \mu$ denote $(s, \mu) \in \rightarrow$ and $s \rightarrow t$ denote $s \rightarrow \mu$ with $\mu(t) = 1$. We define $Dist(s) = \{ \mu \mid s \rightarrow \mu \}$ to be the set of *enabled distributions* in state s. In the following we do not emphasise the parametric nature of our structures and use the term "Markov decision process" (MDP) synonymously. The intuitive operational behaviour of an MDP $\mathcal{M}$ is as follows. First, non-deterministically select some initial state $s_0 \in S_0$. In state s with $Dist(s) \neq \emptyset$, non-deterministically select $\mu \in Dist(s)$. The next state t is randomly chosen with probability $\mu(t)$. If $Dist(t) = \emptyset$, exit; otherwise continue as for state s.

Remark 6 (*Countability of Paths*)**.** In the context of this paper we are only interested in finitely branching Markov decision processes with bounded non-determinism. Therefore $|Dist(s)| < \infty$ for all $s \in S$ and all distributions are assumed to have finite support. Consequently every state has finitely many successor states. Hence there are countably many (finite) paths between any two states. This property is crucial for Definition 12 and Lemma 21 later on. □

A *path* π of MDP $\mathcal{M}$ is a maximal alternating sequence of states and distributions, written $\pi = s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} \ldots$ such that $\mu_i \in Dist(s_i)$ and $\mu_i(s_{i+1}) > 0$ for all $i \geq 0$. As any path is a maximal sequence, it is either infinite or ends in a state s with $Dist(s) = \emptyset$. The set of all paths in $\mathcal{M}$ is denoted $Paths(\mathcal{M})$. Reasoning about probabilities on sets of paths of an MDP relies on the resolution of non-determinism. This resolution is performed by a policy[1] that selects one of the enabled distributions in a state. In general a policy may base its decision in state s on the path fragment from $s_0 \in S_0$ to s. However in the context of this paper we are interested in computing expectations in MDPs and so it suffices to consider positional policies as shown in [6].

Definition 7 (*Positional Policy*)**.** A function $\mathfrak{P} : S \rightarrow Dist(S)$ with $\mathfrak{P}(s) \in Dist(s)$ for all $s \in S$ is called a positional policy for MDP $\mathcal{M} = (S, S_0, \rightarrow)$. □

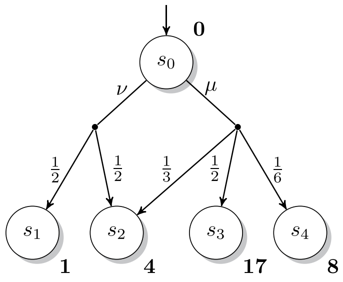
A positional policy deterministically selects an enabled distribution based on the current state s only. As in the rest of this paper we only consider positional policies, we call them simply policies. The path fragment leading to s does not play any role. The path $\pi = s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} \ldots$ is called a $\mathfrak{P}$-path if it is induced by the policy $\mathfrak{P}$, that is, $\mathfrak{P}(s_i) = \mu_i$ for all $i \geq 0$. Let $Paths^{\mathfrak{P}}(s)$ denote the set of $\mathfrak{P}$-paths starting from state s. A policy of an MDP $\mathcal{M}$ induces a Markov chain $\mathcal{M}^{\mathfrak{P}}$ with the same state space as $\mathcal{M}$ and transition probabilities $\mathfrak{P}(s)(t)$ for states s and t. For finite path fragment $\widehat{\pi} = s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_{k-1}} s_k$ of a $\mathfrak{P}$-path, let $\mathbf{P}^{\mathfrak{P}}(\widehat{\pi})$ denote the probability of $\widehat{\pi}$ which is defined by $\mu_0(s_1) \times \cdots \times \mu_{k-1}(s_k) = \prod_{i=1}^{k} \mu_{i-1}(s_i)$. Let $Pr_s^{\mathfrak{P}}(\Pi)$ denote the probability of the set of paths Π all starting in s under policy $\mathfrak{P}$. This probability measure is defined in the standard way using a cylinder set construction on the induced Markov chain $\mathcal{M}^{\mathfrak{P}}$ [14]. In the following we drop the subscript s whenever it is clear from the context. Note that the measure $\mathbf{P}^{\mathfrak{P}}(\pi)$ of a path π starting in state s is 0 if $\pi \notin Paths^{\mathfrak{P}}(s)$.

To compare our operational semantics of pGCL with its *wp*- and *wlp*-semantics, we use rewards (or, dually costs).

Definition 8 (*MDP with Rewards*)**.** An MDP with *rewards* (also called reward-MDP, or shortly RMDP) is a pair $(\mathcal{M}, r)$ with $\mathcal{M}$ an MDP with state space S and $r : S \rightarrow \mathbb{R}_{\geq 0}$ a function assigning a real reward to each state. □

Intuitively, the reward $r(s)$ stands for the reward earned on entering state s. The cumulative reward of a finite path fragment $s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} \cdots s_k$ is the sum of the rewards in all states that have been visited, i.e., $r(s_0) + \cdots + r(s_k)$.

Example 9 (*RMDP, Cumulative Reward of a Path*)**.**



[1] Also called scheduler, strategy or adversary.

Assume a policy $\mathfrak{P}$ with $\mathfrak{P}(s_0) = \mu$. Then $\pi = s_0 \xrightarrow{\mu} s_3 \in Paths^{\mathfrak{P}}(s_0)$ is a possible path that is taken under policy $\mathfrak{P}$ with probability $\mathbf{P}^{\mathfrak{P}}(\pi) = 0.5$ and has cumulative reward $r(\pi) = 17$. □

3.2. Reachability objectives

We are interested in reachability events in RMDPs. Let $T \subseteq S$ be a set of target states. The event $\Diamond T$ stands for the reachability of some state in T, i.e., $\Diamond T$ is the set of paths in MDP $\mathcal{M}$ that hit some state $s \in T$. Formally $\Diamond T = \{\pi \in Paths(\mathcal{M}) \mid \exists i \geq 0.\pi[i] \in T\}$ where $\pi[i]$ denotes the ith state visited along π. We write $\pi \models \Diamond T$ whenever π belongs to $\Diamond T$. It follows by standard arguments that $\Diamond T$ is a measurable event. Its measure depends on the chosen initial state and policy. In order to define the cumulative reward for this event we need to introduce the cumulative reward along a path.

Definition 10 (*Cumulative Reachability Reward*)**.** Let $\pi = s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} \cdots$ be a maximal path in RMDP $(\mathcal{M}, r)$ and $T \subseteq S$ a set of target states. If $\pi \models \Diamond T$, the *cumulative reward* along π before reaching T is defined by: $r_T(\pi) = r(s_0) + \cdots + r(s_k)$ where $s_i \notin T$ for all $i < k$ and $s_k \in T$. If $\pi \not\models \Diamond T$, then $r_T(\pi) = 0$. □

Stated in words, the cumulative reward for a path π to reach T is the cumulative reward of the minimal prefix of π satisfying $\Diamond T$. In case π never reaches a state in T, the cumulative reward is defined to be zero. We denote by $Paths^{\mathfrak{P}}(s, \Diamond T)$ the set of paths starting in s that eventually reach T under policy $\mathfrak{P}$.

Remark 11 (*Reward for Paths that Fail to Reach an Objective*)**.** One can argue that the choice of zero as the reward for never reaching T is arbitrary and that this reward could alternatively be defined as e.g., any constant or even infinity. This depends on the purpose of rewards. Later we will reward states that correspond to the terminal states of a program. If an execution fails to reach a terminal state, then we treat this as "undesired" behaviour that has reward zero. This agrees with the previous definition. □

We can now define the *expected* reward for reachability. Usually an expectation of a real valued random variable X with a density function $p(x)$ is defined as

$$E(X) = \int_x x \cdot p(x)dx.$$

Our random variable is the reward but it is not continuous. Even though a reward function maps states to non-negative real values, there are only countably many different rewards that can be accumulated on the way from a state s to a target set T. This is because there are only countably many finite prefixes of paths that lead from a state s to states in T. Hence the random variable – the reward – can assume only countably many distinct values. We can therefore define a discrete probability distribution, which assigns each given reward c the probability of all finite path prefixes that run from s to T and have the cumulative reachability reward c.

Definition 12 (*Expected Reward for Reachability*)**.** Let $(\mathcal{M}, r)$ be an RMDP with state space S and $T \subseteq S$ and $s \in S$. Further let $\mathfrak{C}$ denote the set of all cumulative reachability reward values that can be accumulated by paths from s to T in $(\mathcal{M}, r)$. The *minimal expected reward* until reaching $T \subseteq S$ from $s \in S$, denoted $ExpRew^{(\mathcal{M},r)}(s \models \Diamond T)$, is defined by:

$$\inf_{\mathfrak{P}} \sum_{c \in \mathfrak{C}} c \cdot Pr^{\mathfrak{P}}\{\pi \in Paths^{\mathfrak{P}}(s, \Diamond T) \mid r_T(\pi) = c\}.$$

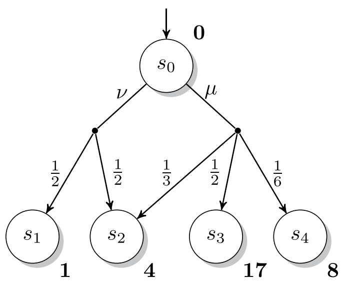
The minimal *liberal* expected reward until reaching some state in T from s, denoted $LExpRew^{(\mathcal{M},r)}(s \models \Diamond T)$, is defined by:

$$\inf_{\mathfrak{P}} \left\{ Pr^{\mathfrak{P}}(s \not\models \Diamond T) + \sum_{c \in \mathfrak{C}} c \cdot Pr^{\mathfrak{P}}\{\pi \in Paths^{\mathfrak{P}}(s, \Diamond T) \mid r_T(\pi) = c\} \right\}.$$

We omit the superscript $(\mathcal{M}, r)$ when the underlying model is clear from the context. □

The expected reward in s to reach some state in T is the expected cumulative reward over all paths (reaching T) induced under a demonic policy. A demonic policy resolves non-determinism such that the expected reward is minimised. The motivation to consider a demonic and not an angelic (maximising) policy lies in the relationship between demonic choice and the notion of *weakest* pre-expectation of pGCL programs. As we will see later, a non-deterministic choice in pGCL will be resolved in such a way that the pre-expectation is minimised. Note that in case T is not reachable with positive probability from s under a demonic policy, $ExpRew(s \models \Diamond T) = 0$. $LExpRew(s \models \Diamond T)$ is the expected reward to reach T or never reach it from s. In case there is no policy under which T can be reached from s with positive probability, we have that $LExpRew(s \models \Diamond T) = 1$. This measure is motivated by reasoning about partial correctness where termination is not guaranteed and we will later see the relationship to the weakest *liberal* pre-expectation in Theorem 24. Note that *ExpRew* and *LExpRew* coincide if T is reached with probability one under all policies. For finite MDPs without parameters, expected and liberal expected rewards for reachability objectives can be obtained by solving a linear programming problem, cf. [6].

Example 13 (*Expected Rewards*).



Let $T = \{s_2, s_3\}$. Then $ExpRew(s_0 \models \Diamond T) = \min\{4 \cdot \frac{1}{2}, 4 \cdot \frac{1}{3} + 17 \cdot \frac{1}{2}\} = \min\{2, \frac{59}{6}\} = 2$. And $LExpRew(s_0 \models \Diamond T) = \min\{\frac{1}{2} + 4 \cdot \frac{1}{2}, \frac{1}{6} + 4 \cdot \frac{1}{3} + 17 \cdot \frac{1}{2}\} = \min\{2.5, 10\} = 2.5$. $\square$

4. pGCL semantics

This section describes an expectation transformer semantics of pGCL, as well as an operational semantics using MDPs. The main result of this section is a formal connection between these two semantics.

4.1. Denotational semantics

When probabilistic programs are executed they determine a probability distribution over final values of program variables. For instance, on termination of

$$(x := 1\,[0.75]\,x := 2);$$

the final value of x is 1 with probability $\frac{3}{4}$ and 2 with probability $\frac{1}{4}$. An alternative way to characterise that probabilistic behaviour is to consider the expected values over random variables with respect to that distribution. For example, to determine the probability that x is set to 1, we can compute the expected value of the random variable "x is 1" which is $\frac{3}{4} \cdot 1 + \frac{1}{4} \cdot 0 = \frac{3}{4}$. Similarly, to determine the average value of x, we compute the expected value of the random variable "x" which is $\frac{3}{4} \cdot 1 + \frac{1}{4} \cdot 2 = \frac{5}{4}$.

More generally, rather than a distribution-centred approach [15,16], we take an "expectation transformer" [4] approach. We annotate probabilistic programs with *expectations*. As before we assume a state space S, a set of parameters Par and a set of expressions $V(Par)$ over it.

Definition 14 (*Expectation*). Expectations are functions which map program states to (non-negative) real values or expressions over parameters. The set of expectations over state space S is then

$$\mathbf{E} = \{f \mid f : S \to \mathbb{R}_{\geq 0} \cup V(Par)\}. \quad \square$$

Note that every expectation f maps to a non-negative real value or an expression that is non-negative for all possible evaluations of its parameters. Expectations are the quantitative analogue to Hoare's predicates for non-probabilistic programs. An expectation transformer is a total function between two expectations. The expectation transformer $wp(P, f)$ for program P and post-expectation f yields the least expected value e on P's initial state ensuring that P's execution terminates with a value f. Annotation $\langle e \rangle\, P\, \langle f \rangle$ holds for total correctness if and only if $e \leq wp(P, f)$ where $\leq$ is to be interpreted in a point-wise manner. Intuitively, implication between predicates is generalised to pointwise inequality between expectations. For convenience we use square brackets to cast Boolean truth values to numbers and by convention [true] = 1 and [false] = 0.

Definition 15 (*wp-semantics of* pGCL *[4]*). Let P and Q be pGCL-programs, f a post-expectation, x a program variable, E an expression, and G a Boolean expression. The *wp*-semantics of a program is defined by structural induction over the program as follows:

- $wp(\mathsf{skip}, f) = f$
- $wp(\mathsf{abort}, f) = 0$
- $wp(x := E, f) = f[x/E]$
- $wp(P; Q, f) = wp(P, wp(Q, f))$
- $wp(\mathsf{if}(G)\{P\}\mathsf{else}\{Q\}, f) = [G] \cdot wp(P, f) + [\neg G] \cdot wp(Q, f)$
- $wp(P\,[]\,Q, f) = \min\,(wp(P, f), wp(Q, f))$
- $wp(P\,[p]\,Q, f) = p \cdot wp(P, f) + (1 - p) \cdot wp(Q, f)$
- $wp(\mathsf{while}(G)\{P\}, f) = \mu X.\,([G] \cdot wp(P, X) + [\neg G] \cdot f)$.

Here $f[x/E]$ denotes a function that is obtained from f by replacing every occurrence of x by E. The least fixed point operator μ is used w.r.t. the ordering $\leq$ on expectations. The existence of the fixed point can be seen from Theorem 30 in the Appendix. We refer to [4] for more details. □

If program P does not contain a probabilistic choice, then this *wp* is isomorphic to Dijkstra's *wp* [4]. A weakest liberal pre-expectation $wlp(P, f)$ yields the least expectation for which P either does not terminate or establishes f.

Definition 16 (*wlp-semantics of* pGCL)**.** *wlp*-semantics differs from *wp*-semantics only for while and abort:

- $wlp(\mathsf{abort}, f) = 1$
- $wlp(\mathsf{while}(G)\{P\}, f) = \nu X.\, ([G] \cdot wlp(P, X) + [\neg G] \cdot f)$.

Here ν is the greatest fixed point operator w.r.t. the ordering $\leq$ on expectations. □

While expectations do not need to be bounded from above in general, an upper bound is required for the definition of *wlp*. This is because non-terminating programs produce the maximal pre-expectation which has to be well-defined. In this paper we set this upper bound of *wlp* to one. In [17] a more general approach is discussed.

As the previous definitions indicate, the difference between *wp* and *wlp* lies in the handling of non-termination. The expectation transformers *wp* and *wlp* coincide for programs that terminate with probability one as is the case for *ExpRew* and *LExpRew*.

Example 17 (*Application of wp-semantics*)**.** Consider again the duelling cowboys example. Assume we are given the post-expectation:

$$f(c,t) = [t = A \wedge c = 0] + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab}.$$

The post-expectation f gives the probability that A wins, depending on the state that the program is in. The states are characterised by the predicates in square brackets. Let us compute the weakest pre-expectation of the loop body from Lst. 1 w.r.t. the post-expectation f. This yields:

$$\begin{aligned}
& wp(\mathsf{if}(t = A)\{(c := 0\,[a]\,t := B);\,\}\mathsf{else}\{(c := 0\,[b]\,t := A);\,\}, f) \\
&= [t = A] \cdot wp((c := 0\,[a]\,t := B), f) + [t \neq A] \cdot wp((c := 0\,[b]\,t := A), f) \\
&= [t = A] \cdot (a \cdot wp(c := 0, f) + (1-a) \cdot wp(t := B, f)) + [t \neq A] \cdot (b \cdot wp(c := 0, f) + (1-b) \cdot wp(t := A, f)) \\
&= [t = A] \cdot \left(a \cdot \left([t = A \wedge 0 = 0] + [t = A \wedge 0 = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge 0 = 1] \cdot \frac{(1-b)a}{a+b-ab}\right)\right. \\
&\qquad\qquad \left. + (1-a) \cdot \left([B = A \wedge c = 0] + [B = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [B = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab}\right)\right) \\
&\quad + [t \neq A] \cdot \left(b \cdot \left([t = A \wedge 0 = 0] + [t = A \wedge 0 = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge 0 = 1] \cdot \frac{(1-b)a}{a+b-ab}\right)\right. \\
&\qquad\qquad \left. + (1-b) \cdot \left([A = A \wedge c = 0] + [A = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [A = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab}\right)\right) \\
&= [t = A] \cdot \left(a \cdot [t = A] + (1-a) \cdot [c = 1] \cdot \frac{(1-b)a}{a+b-ab}\right) \\
&\quad + [t \neq A] \cdot \left(b \cdot [t = A] + (1-b)\left([c = 0] + [c = 1] \cdot \frac{a}{a+b-ab}\right)\right) \\
&= [t = A] \cdot a + [t = A \wedge c = 1] \cdot \frac{(1-a)(1-b)a}{a+b-ab} \\
&\quad + [t \neq A \wedge c = 0] \cdot (1-b) + [t \neq A \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab} \\
&= [t = A \wedge c \neq 1] \cdot a + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} \\
&\quad + [t \neq A \wedge c = 0] \cdot (1-b) + [t \neq A \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab}.
\end{aligned}$$

In the last step we use the fact that $[t = A] \cdot a$ can be split into

$$[t = A \wedge c = 1] \cdot a + [t = A \wedge c \neq 1] \cdot a.$$

This computation tells us that if, say f describes the probability after one iteration of the loop that cowboy A wins, then the computed expression gives the probability that A wins before that iteration of the loop. The result of this example will be used in Section 5. □

Table 1
Inference rules for pGCL programs.

$$\langle \text{skip}, \eta\rangle \to \langle \text{exit}, \eta\rangle$$
$$\langle \text{abort}, \eta\rangle \to \langle \text{abort}, \eta\rangle$$
$$\langle x := E, \eta\rangle \to \langle \text{exit}, \eta[x := [\![E]\!]_\eta]\rangle$$
$$\frac{\langle P, \eta\rangle \to \mu}{\langle P; Q, \eta\rangle \to \nu} \quad \text{with } \nu(\langle P'; Q, \eta'\rangle) = \mu(\langle P', \eta'\rangle) \text{ where exit; } Q = Q.$$
$$\langle P \, [] \, Q, \eta\rangle \to \langle P, \eta\rangle \qquad \langle P \, [] \, Q, \eta\rangle \to \langle Q, \eta\rangle$$
$$\langle P \, [p] \, Q, \eta\rangle \to \mu \quad \text{with } \mu(\langle P, \eta\rangle) = p \quad \text{and} \quad \mu(\langle Q, \eta\rangle) = 1 - p$$
$$\frac{\eta \models G}{\langle \text{if}(G)\{P\} \text{ else } \{Q\}, \eta\rangle \to \langle P, \eta\rangle}$$
$$\frac{\eta \not\models G}{\langle \text{if}(G)\{P\} \text{ else } \{Q\}, \eta\rangle \to \langle Q, \eta\rangle}$$
$$\frac{\eta \models G}{\langle \text{while}(G)\{P\}, \eta\rangle \to \langle P; \text{while}(G)\{P\}, \eta\rangle}$$
$$\frac{\eta \not\models G}{\langle \text{while}(G)\{P\}, \eta\rangle \to \langle \text{exit}, \eta\rangle}$$

4.2. Operational semantics

Our aim is to model the stepwise behaviour of a pGCL-program P by an MDP denoted $\mathcal{M}[\![P]\!]$. This MDP represents the operational interpretation of the program P and intuitively acts as an abstract machine for P. This is done as follows. Let η be a variable valuation of the program variables. That is, η is a mapping from the program variables onto their (possibly infinite) domains. For variable x, $\eta(x)$ denotes the value of x under η. For expression E, let $[\![E]\!]_\eta$ denote the value of E under valuation η. This is defined in the standard way, e.g., for $E = 2{\cdot}x{+}y$ with $\eta(x) = 3$ and $\eta(y) = 7$, we have $[\![E]\!]_\eta = 2{\cdot}\eta(x){+}\eta(y) = 13$. We use the distinguished semantic construct exit to denote the successful termination of a program. States in the MDP are of the form $\langle Q, \eta\rangle$ with Q a pGCL-statement or Q = exit and η a variable valuation. For instance, the execution of the assignment $x := 2 \cdot x + y$ under valuation η with $\eta(x) = 3$ and $\eta(y) = 7$ results in the state $\langle \text{exit}, \eta'\rangle$ where η' is the same as η except that $\eta'(x) = 13$. Initial states of program P are tuples $\langle P, \eta\rangle$ where η is arbitrary.

Definition 18 (*Operational Semantics of* pGCL)**.** The operational semantics of pGCL-program P, denoted $\mathcal{M}[\![P]\!]$, is the MDP $(S, S_0, \to)$ where:

- S is the set of pairs $\langle Q, \eta\rangle$ with Q a pGCL-program or Q = exit, and η is a variable valuation of the variables occurring in P,
- $S_0 = \{ \langle P, \eta\rangle \}$ where η is arbitrary, and
- $\to$ is the smallest relation that is induced by the inference rules in Table 1. □

Each rule tells us how to obtain the successors from a state. For example, in a state with probabilistic choice the MDP will make a transition to the parametric distribution μ. Then a successor state is chosen according to μ. We omit the distribution when it is obvious and write an arrow to the successor instead. For instance the rules for non-deterministic choice are a shorthand for

$$\langle P \, [] \, Q, \eta\rangle \to \{\mu, \nu\}$$
$$\text{with } \mu(\langle P, \eta\rangle) = 1 \quad \text{and} \quad \nu(\langle Q, \eta\rangle) = 1.$$

A premise is used to enable or disable transitions depending on the variable valuation of the current state. Consider the last two rules: if the system evolves from a state that represents a loop, it will proceed to a state where the loop body has to be executed once before going back to the loop header *provided* that the current variable valuation η satisfies the loop's guard G. If it does not, the last rule dictates that the loop is terminated, i.e. the system moves to an exit state.

Example 19 (*Operational Semantics*)**.** Fig. 1 depicts the MDP underlying the cowboy example. This MDP is parameterised with parameters a and b. Technically, for every possible initial variable evaluation there should be an initial state. However, a programmer usually has to initialise the program variables before they may be used in a computation. This is also the case in our example program from Lst. 1. Therefore it does not matter in which initial state we start as the initialisation steps will always take us to the states $(4, A, 1)$ or $(4, B, 1)$. This observation allows us to represent the program by an MDP with a finite state space where all initial states have been merged into one. A slight adaptation of our example program in which we keep track of the number of shots before one of the cowboys dies, yields an MDP with infinitely many states. The support of any distribution in this MDP is finite however. □

Let $P^{\surd}$ denote the set of states in MDP $\mathcal{M}[\![P]\!]$ of the form $\langle \text{exit}, \eta\rangle$ for arbitrary variable valuation η. Note that states in $P^{\surd}$ represent the successful termination of P. If $P^{\surd}$ is not reachable, program P diverges under all possible policies.

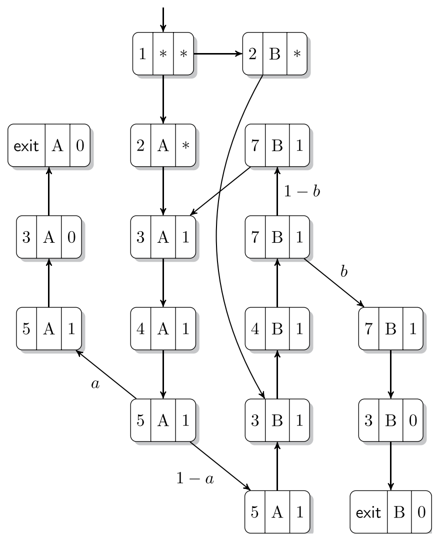


Fig. 1. MDP $\mathcal{M}$ for the duelling cowboys example. Each state is determined by a triple: (program location, value of t, value of c) where $*$ denotes an arbitrary value.

Definition 20 (*RMDP of a* pGCL*-program*)**.** Let P be a pGCL-program and f a post-expectation for P. The reward-MDP associated with P and f is defined as $\mathcal{R}_f[\![P]\!] = (\mathcal{M}[\![P]\!], r)$ with $\mathcal{M}[\![P]\!]$ the MDP of P as defined before and reward function r defined by $r(s) = f(\eta)$ if $s = \langle \text{exit}, \eta \rangle \in P^{\surd}$ and $r(s) = 0$ otherwise. □

Note that we use a special reward structure: only terminal states are assigned a reward which is not necessarily zero. All other states have a zero reward. The following lemma explains how expected rewards can be computed in $\mathcal{R}_f[\![P]\!]$.

Lemma 21 (*Characterising Expected Rewards*)**.** *For* pGCL *program P and a state $s = \langle P, \eta \rangle$, we have:*

$$ExpRew^{\mathcal{R}_f[\![P]\!]}(s \models \Diamond P^{\surd}) = \inf_{\mathfrak{P}} \sum_{\widehat{\pi} \in Paths^{\mathfrak{P}}_{\min}(s, \Diamond P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{P^{\surd}}(\widehat{\pi}),$$

where $Paths^{\mathfrak{P}}_{\min}(s, \Diamond P^{\surd})$ is the set containing all (finite) paths of the form $s_0 \ldots s_k$ with $s_0 = s$, $s_k \in P^{\surd}$ and $s_i \notin P^{\surd}$ for all $0 \le i < k$ that adhere to the policy $\mathfrak{P}$. □

Proof. Let $T = P^{\surd}$ for pGCL program P. The proof requires a property stated in Remark 6 namely that in an MDP there are only countably many finite paths that lead from one state to another. Consider the definition of expected reward:

$$\inf_{\mathfrak{P}} \sum_{c \in \mathfrak{C}} c \cdot Pr^{\mathfrak{P}}\{\, \pi \in Paths^{\mathfrak{P}}(s, \Diamond T) \mid r_T(\pi) = c \,\}.$$

Given that $Pr^{\mathfrak{P}}(\pi \models \Diamond T) = \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})$ where prefix $\widehat{\pi}$ of π is minimal and ends in T, the above term equals:

$$\inf_{\mathfrak{P}} \sum_{c \in \mathfrak{C}} c \cdot \mathbf{P}^{\mathfrak{P}}\{\widehat{\pi} \in Paths^{\mathfrak{P}}_{\min}(s, \Diamond T) \mid r_T(\widehat{\pi}) = c\}.$$

As in $\mathcal{R}_f[\![P]\!]$ the number of finite path prefixes $\widehat{\pi}$ that reach T and accumulate a reward c is countable we can rewrite the sum into:

$$\inf_{\mathfrak{P}} \sum_{\widehat{\pi} \in \text{Paths}^{\mathfrak{P}}_{\min}(s, \Diamond T)} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_T(\widehat{\pi}). \quad \square$$

Lemma 21 expresses the expected reward in terms of paths that reach an exit state and their cumulative rewards. This provides a straightforward way to calculate expected rewards (for finite systems). In the next subsection Lemma 21 will be helpful in the proofs of our main results.

Analogously we obtain:

Lemma 22 (*Characterising Liberal Expected Rewards*). *For* pGCL *program P and variable valuation η, we have:*

$$LExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P,\eta\rangle \models \Diamond P^{\surd}) = \inf_{\mathfrak{P}}\left\{Pr^{\mathfrak{P}}(\langle P,\eta\rangle \not\models \Diamond P^{\surd}) + \sum_{\widehat{\pi}\in Paths^{\mathfrak{P}}_{\min}(s,\Diamond P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{P^{\surd}}(\widehat{\pi})\right\}. \quad \square$$

Proof. Follows immediately from Definition 12 and Lemma 21. $\square$

4.3. Correspondence between operational and expectation transformer semantics

We now present the main results of this paper: a formal relationship between the *wp*-semantics of pGCL-program P and its operational semantics in terms of a RMDP, and similarly for the *wlp*-semantics. We first consider the *wp*-semantics.

Theorem 23 (*Correspondence Theorem*). *For* pGCL*-program P, variable valuation η, and post-expectation f:*

$$wp(P,f)(\eta) = ExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P,\eta\rangle \models \Diamond P^{\surd}). \quad \square$$

Proof. By structural induction over the pGCL program P. We write paths as sequences of states and leave out the distribution in between each pair of states for the ease of presentation. In this proof we use the alternative definition for expected rewards given in Lemma 21.

Induction base:

- For $P = \mathsf{skip}$ we use the fact that skip does not change the post-expectation. We derive:

$$\begin{aligned} ExpRew^{\mathcal{R}_f[\![\mathsf{skip}]\!]}(\langle \mathsf{skip},\eta\rangle \models \Diamond \mathsf{skip}^{\surd}) &= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathsf{Paths}^{\mathfrak{P}}_{\min}(\langle \mathsf{skip},\eta\rangle,\Diamond \mathsf{skip}^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{\mathsf{skip}^{\surd}}(\widehat{\pi}) \\ &= \inf_{\mathfrak{P}} \mathbf{P}^{\mathfrak{P}}(\langle \mathsf{skip},\eta\rangle\langle \mathsf{exit},\eta\rangle)\cdot f(\eta) \\ &= 1\cdot f(\eta) \\ &= f(\eta) \\ &= wp(\mathsf{skip},f)(\eta). \end{aligned}$$

- For $P = \mathsf{abort}$ we use the fact that it fails to terminate and has a pre-expectation of zero. We derive:

$$\begin{aligned} ExpRew^{\mathcal{R}_f[\![\mathsf{abort}]\!]}(\langle \mathsf{abort},\eta\rangle \models \Diamond \mathsf{abort}^{\surd}) &= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathsf{Paths}^{\mathfrak{P}}_{\min}(\langle \mathsf{abort},\eta\rangle,\Diamond \mathsf{abort}^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{\mathsf{abort}^{\surd}}(\widehat{\pi}) \\ &= 0 \\ &= wp(\mathsf{abort},f)(\eta) \end{aligned}$$

  as there is no path starting from $\langle \mathsf{abort},\eta\rangle$ that reaches an exit-state.
- Let P be the assignment $x := E$. For this case we apply the substitution:

$$\begin{aligned} ExpRew^{\mathcal{R}_f[\![x:=E]\!]}(\langle x := E,\eta\rangle \models \Diamond (x := E)^{\surd}) &= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathsf{Paths}^{\mathfrak{P}}_{\min}(\langle x:=E,\eta\rangle,\Diamond (x:=E)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{(x:=E)^{\surd}}(\widehat{\pi}) \\ &= \inf_{\mathfrak{P}} \mathbf{P}^{\mathfrak{P}}(\langle x := E,\eta\rangle\langle \mathsf{exit},\eta[x/E]\rangle)\cdot f(\eta[x/E]) \\ &= 1\cdot f(\eta[x/E]) \\ &= f(\eta[x/E]) \\ &= f[x/E](\eta) \\ &= wp(x := E,f)(\eta). \end{aligned}$$

Induction hypothesis: assume that for program P (and analogously for Q)

$$wp(P,f)(\eta) = ExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P,\eta\rangle \models \Diamond P^{\surd}).$$

Induction step:

- Consider the probabilistic choice $P\,[p]\,Q$ (this also covers conditional choice since it can be written as $P\,[[G]]\,Q$).[2] The idea is to represent the expected reward as a weighted sum of the expected rewards computed from successor states. This corresponds to the weighted sum for the weakest pre-expectation:

$$\begin{aligned}
&\mathit{ExpRew}^{\mathcal{R}_f[\![P\,[p]\,Q]\!]}(\langle P\,[p]\,Q,\eta\rangle \models \Diamond(P\,[p]\,Q)^{\surd})\\
&= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P\,[p]\,Q,\eta\rangle,\Diamond(P\,[p]\,Q)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{(P\,[p]\,Q)^{\surd}}(\widehat{\pi})\\
&= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P,\eta\rangle,\Diamond P^{\surd})} p\cdot \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{P^{\surd}}(\widehat{\pi}) + \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle Q,\eta\rangle,\Diamond Q^{\surd})} (1-p)\cdot \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{Q^{\surd}}(\widehat{\pi})\\
&\overset{*}{=} p\cdot \inf_{\mathfrak{P}_1} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}_1}_{\min}(\langle P,\eta\rangle,\Diamond P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{P^{\surd}}(\widehat{\pi}) + (1-p)\cdot \inf_{\mathfrak{P}_2} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}_2}_{\min}(\langle Q,\eta\rangle,\Diamond Q^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{Q^{\surd}}(\widehat{\pi})\\
&= p\cdot \mathit{ExpRew}^{\mathcal{R}_f[\![P]\!]}(\langle P,\eta\rangle \models \Diamond P^{\surd}) + (1-p)\cdot \mathit{ExpRew}^{\mathcal{R}_f[\![Q]\!]}(\langle Q,\eta\rangle \models \Diamond Q^{\surd})\\
&\overset{I.H.}{=} p\cdot \mathit{wp}(P,f)(\eta) + (1-p)\cdot \mathit{wp}(Q,f)(\eta)\\
&= \mathit{wp}(P\,[p]\,Q,f)(\eta).
\end{aligned}$$

In $*$ we use the fact that the policy for paths starting in $\langle P,\eta\rangle$ is independent of the policy for paths starting in $\langle Q,\eta\rangle$ because they are positional policies.

- Consider the non-deterministic choice $P\,[]\,Q$ which is analogous to probabilistic choice, except that min replaces the weighted sum:

$$\begin{aligned}
&\mathit{ExpRew}^{\mathcal{R}_f[\![P\,[]\,Q]\!]}(\langle P\,[]\,Q,\eta\rangle \models \Diamond(P\,[]\,Q)^{\surd})\\
&= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P\,[]\,Q,\eta\rangle,\Diamond(P\,[]\,Q)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{(P\,[]\,Q)^{\surd}}(\widehat{\pi})\\
&= \min\left\{ \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P,\eta\rangle,\Diamond P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{P^{\surd}}(\widehat{\pi}),\ \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle Q,\eta\rangle,\Diamond Q^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{Q^{\surd}}(\widehat{\pi}) \right\}\\
&= \min\{\mathit{ExpRew}^{\mathcal{R}_f[\![P]\!]}(\langle P,\eta\rangle \models \Diamond P^{\surd}), \mathit{ExpRew}^{\mathcal{R}_f[\![Q]\!]}(\langle Q,\eta\rangle \models \Diamond Q^{\surd})\}\\
&\overset{I.H.}{=} \min\{\mathit{wp}(P,f), \mathit{wp}(Q,f)\}\\
&= \mathit{wp}(P[]Q,f)(\eta).
\end{aligned}$$

- Consider the sequential composition $P;Q$. The idea is break up each path into a prefix that corresponds to the execution of P and a suffix that corresponds to the execution of Q. We can then compute the expected reward over the suffixes and use this intermediate result to compute the expected reward over the prefixes which corresponds to the nesting of weakest pre-expectations:

$$\begin{aligned}
&\mathit{ExpRew}^{\mathcal{R}_f[\![P;Q]\!]}(\langle P;Q,\eta\rangle \models \Diamond(P;Q)^{\surd})\\
&= \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P;Q,\eta\rangle,\Diamond(P;Q)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r_{(P;Q)^{\surd}}(\widehat{\pi})\\
&\overset{*}{=} \inf_{\mathfrak{P}} \sum_{\widehat{\pi}\in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P;Q,\eta\rangle,\Diamond P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi})\cdot r^{q}_{P^{\surd}}(\widehat{\pi}) \text{ where } r^{q}_{P^{\surd}}(\widehat{\pi}) \text{ is the sum of rewards } r_q \text{ along } \widehat{\pi} \text{ with}\\
&\qquad r_q(s) = \inf_{\mathfrak{P}'}\left(\sum_{\widehat{\pi}'\in \mathrm{Paths}^{\mathfrak{P}'}_{\min}(s,\Diamond Q^{\surd})} \mathbf{P}^{\mathfrak{P}'}(\widehat{\pi}')\cdot r_{Q^{\surd}}(\widehat{\pi}') \right) \text{ if } s\in P^{\surd} \text{ and } r_q(s)=0 \text{ otherwise}\\
&= \mathit{ExpRew}^{\mathcal{R}_g[\![P]\!]}(\langle P,\eta\rangle \models \Diamond P^{\surd}) \text{ where } g(\eta) = \mathit{ExpRew}^{\mathcal{R}_f[\![Q]\!]}(\langle Q,\eta\rangle \models \Diamond Q^{\surd})\\
&\overset{I.H.}{=} \mathit{wp}(P;\mathit{wp}(Q,f))(\eta)\\
&= \mathit{wp}(P;Q,f)(\eta).
\end{aligned}$$

In $*$ we divide each path into the aforementioned pre- and suffixes. We use the positionality of policies as the policies according to which the suffixes are constructed are independent of the history, i.e. the prefix of those paths.

[2] The guard is enclosed in square brackets twice: the inner brackets cast the Boolean formula to a {0, 1}-valued function, the outer brackets are part of the probabilistic choice statement.

- Consider the loop while$(G)\{P\}$. For this case we show by induction that the two semantics correspond for every iteration that the loop performs. We rely on the previously shown cases for abort, skip and sequential composition. Let the bounded while-loop for $k > 0$ be

$$(\text{while}(G)\{P\})^{k+1} = \text{if}(G)\{P;\ (\text{while}(G)\{P\})^k\}\ \text{else}\ \{\text{skip}\}$$

where the base case is $(\text{while}(G)\{P\})^0 = \text{abort}$. We will show for every k that

$$wp((\text{while}(G)\{P\})^k, f)(\eta) = ExpRew^{\mathcal{R}_f[\![(\text{while}(G)\{P\})^k]\!]}(\eta). \tag{1}$$

Observe that

$$wp((\text{while}(G)\{P\})^{k+1}, f)(\eta) \geq wp((\text{while}(G)\{P\})^k, f)(\eta).$$

From the fixpoint Theorem 3 in [18] we know that the more iterations the bounded while loop is allowed to perform, the closer it approximates the fixpoint given in Definition 15. Formally this means

$$\lim_{k\to\infty} wp((\text{while}(G)\{P\})^k, f)(\eta) = wp(\text{while}(G)\{P\}, f)(\eta). \tag{2}$$

A thorough justification for (2) is given in Appendix A.
From (1) it follows that for every k, $ExpRew$ behaves identically to wp. Thus with (2) it follows that

$$wp(\text{while}(G)\{P\}, f)(\eta) = ExpRew^{\mathcal{R}_f[\![\text{while}(G)\{P\}]\!]}(\eta).$$

It remains to prove (1). This is done by induction on k. Base case ($k = 0$):

$$\begin{aligned} wp((\text{while}(G)\{P\})^0, f)(\eta) &= wp(\text{abort}, f)(\eta) \\ &\overset{*}{=} ExpRew^{\mathcal{R}_f[\![\text{abort}]\!]}(\eta) \\ &= ExpRew^{\mathcal{R}_f[\![(\text{while}(G)\{P\})^0]\!]}(\eta) \end{aligned}$$

$(*)$ was already shown earlier in the case abort.
Induction hypothesis: Eq. (1) holds for some unspecified but fixed value of k.
Induction step:

$$\begin{aligned} wp((\text{while}(G)\{P\})^{k+1}, f)(\eta) &= wp(\text{if}(G)\{P;\ (\text{while}(G)\{P\})^k\}\text{else}\{\text{skip}\}, f)(\eta) \\ &= \left([G] \cdot wp(P;\ (\text{while}(G)\{P\})^k, f) + [\neg G] \cdot wp(\text{skip}, f)\right)(\eta) \\ &\overset{*}{=} \left([G] \cdot ExpRew^{\mathcal{R}_f[\![P;(\text{while}(G)\{P\})^k]\!]} + [\neg G] \cdot ExpRew^{\mathcal{R}_f[\![\text{skip}]\!]}\right)(\eta) \\ &= ExpRew^{\mathcal{R}_f[\![\text{if}(G)\{P;(\text{while}(G)\{P\})^k\}\text{else}\{\text{skip}\}]\!]}(\eta) \\ &= ExpRew^{\mathcal{R}_f[\![(\text{while}(G)\{P\})^{k+1}]\!]}(\eta) \end{aligned}$$

$(*)$ follows from the induction hypothesis and the previously shown cases for skip and sequential composition. □

Thus, $wp(P, f)$ evaluated at η is the least expected value of f over any of the result distributions of P.

Theorem 24 (*Correspondence Theorem for Liberal Semantics*). *For* pGCL*-program P, variable valuation η, and post-expectation f:*

$$wlp(P, f)(\eta) = LExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P, \eta\rangle \models \Diamond P^{\surd}). \quad \square$$

Proof. By structural induction over the pGCL program P (analogously to the proof of Theorem 23). Similarly we apply Lemma 22 here. To avoid repetition we skip the base cases which are rather simple.
Induction hypothesis: assume that for program P (and analogously for Q)

$$wlp(P, f)(\eta) = LExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P, \eta\rangle \models \Diamond P^{\surd}).$$

Induction step:

- Consider the probabilistic choice $P\,[p]\,Q$ (again, this covers conditional choice):

$$\begin{aligned} & LExpRew^{\mathcal{R}_f[\![P\,[p]\,Q]\!]}(\langle P\,[p]\,Q, \eta\rangle \models \Diamond (P\,[p]\,Q)^{\surd}) \\ &= \inf_{\mathfrak{P}} Pr^{\mathfrak{P}}(\langle P\,[p]\,Q, \eta\rangle \not\models \Diamond (P\,[p]\,Q)^{\surd}) + \sum_{\widehat{\pi} \in \text{Paths}^{\mathfrak{P}}_{\min}(\langle P\,[p]\,Q, \eta\rangle, \Diamond (P\,[p]\,Q)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{(P\,[p]\,Q)^{\surd}}(\widehat{\pi}) \\ &= \inf_{\mathfrak{P}} \Bigg(p \cdot Pr^{\mathfrak{P}}(\langle P, \eta\rangle \not\models \Diamond P^{\surd}) + \sum_{\widehat{\pi} \in \text{Paths}^{\mathfrak{P}}_{\min}(\langle P, \eta\rangle, \Diamond P^{\surd})} p \cdot \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{P^{\surd}}(\widehat{\pi}) \end{aligned}$$

$$+ (1-p) \cdot Pr^{\mathfrak{P}}(\langle Q, \eta\rangle \not\models \lozenge Q^{\surd}) + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle Q,\eta\rangle, \lozenge Q^{\surd})} (1-p) \cdot \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{Q^{\surd}}(\widehat{\pi}) \Bigg)$$

$$= p \cdot \inf_{\mathfrak{P}_1} Pr^{\mathfrak{P}_1}(\langle P, \eta\rangle \not\models \lozenge P^{\surd}) + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}_1}_{\min}(\langle P,\eta\rangle, \lozenge P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{P^{\surd}}(\widehat{\pi})$$

$$+ (1-p) \cdot \inf_{\mathfrak{P}_2} Pr^{\mathfrak{P}_2}(\langle Q, \eta\rangle \not\models \lozenge Q^{\surd}) + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}_2}_{\min}(\langle Q,\eta\rangle, \lozenge Q^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{Q^{\surd}}(\widehat{\pi})$$

$$= p \cdot LExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P, \eta\rangle \models \lozenge P^{\surd}) + (1-p) \cdot LExpRew^{\mathcal{R}_f[\![Q]\!]}(\langle Q, \eta\rangle \models \lozenge Q^{\surd})$$

$$\overset{I.H.}{=} p \cdot wlp(P, f)(\eta) + (1-p) \cdot wlp(Q, f)(\eta)$$

$$= wlp(P\,[p]\,Q, f)(\eta).$$

- Consider the non-deterministic choice $P\,[]\,Q$:

$$LExpRew^{\mathcal{R}_f[\![P\,[]\,Q]\!]}(\langle P\,[]\,Q, \eta\rangle \models \lozenge (P\,[]\,Q)^{\surd})$$

$$= \inf_{\mathfrak{P}} Pr^{\mathfrak{P}}(\langle P\,[]\,Q, \eta\rangle \not\models \lozenge (P\,[]\,Q)^{\surd}) + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P\,[]\,Q,\eta\rangle, \lozenge (P\,[]\,Q)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{(P\,[]\,Q)^{\surd}}(\widehat{\pi})$$

$$= \min \left\{ \inf_{\mathfrak{P}} \left(Pr^{\mathfrak{P}}(\langle P, \eta\rangle \not\models \lozenge P^{\surd}) + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P,\eta\rangle, \lozenge P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{P^{\surd}}(\widehat{\pi}) \right), \right.$$

$$\left. \inf_{\mathfrak{P}} \left(Pr^{\mathfrak{P}}(\langle Q, \eta\rangle \not\models \lozenge Q^{\surd}) + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle Q,\eta\rangle, \lozenge Q^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{Q^{\surd}}(\widehat{\pi}) \right) \right\}$$

$$= \min\{LExpRew^{\mathcal{R}_f[\![P]\!]}(\langle P, \eta\rangle \models \lozenge P^{\surd}), LExpRew^{\mathcal{R}_f[\![Q]\!]}(\langle Q, \eta\rangle \models \lozenge Q^{\surd})\}$$

$$\overset{I.H.}{=} \min\{wlp(P, f)(\eta), wlp(Q, f)(\eta)\}$$

$$= wlp(P[]Q, f)(\eta).$$

- Consider the sequential composition $P; Q$:

$$LExpRew^{\mathcal{R}_f[\![P;Q]\!]}(\langle P; Q, \eta\rangle \models \lozenge (P; Q)^{\surd})$$

$$= \inf_{\mathfrak{P}} \left(Pr^{\mathfrak{P}}\{\langle P; Q, \eta\rangle \not\models \lozenge (P; Q)^{\surd}\} + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P;Q,\eta\rangle, \lozenge (P;Q)^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r_{(P;Q)^{\surd}}(\widehat{\pi}) \right)$$

$$\overset{*}{=} \inf_{\mathfrak{P}} \left(Pr^{\mathfrak{P}}\{\langle P, \eta\rangle \not\models \lozenge P^{\surd}\} + \sum_{\widehat{\pi} \in \mathrm{Paths}^{\mathfrak{P}}_{\min}(\langle P;Q,\eta\rangle, \lozenge P^{\surd})} \mathbf{P}^{\mathfrak{P}}(\widehat{\pi}) \cdot r^{q}_{P^{\surd}}(\widehat{\pi}) \right)$$

where $r^{q}_{P^{\surd}}(\widehat{\pi})$ is the sum of rewards r_q along $\widehat{\pi}$ with

$$r_q(s) = \inf_{\mathfrak{P}'} \left(Pr^{\mathfrak{P}'}\{\langle Q, \eta'\rangle \not\models \lozenge Q^{\surd}\} + \sum_{\widehat{\pi}' \in \mathrm{Paths}^{\mathfrak{P}'}_{\min}(s, \lozenge Q^{\surd})} \mathbf{P}^{\mathfrak{P}'}(\widehat{\pi}') \cdot r_{Q^{\surd}}(\widehat{\pi}') \right) \text{if } s \in P^{\surd} \quad \text{and} \quad r_q(s) = 0 \text{ otherwise}$$

$$= LExpRew^{\mathcal{R}_g[\![P]\!]}(\langle P, \eta\rangle \models \lozenge P^{\surd}) \text{ where } g(\eta) = LExpRew^{\mathcal{R}_f[\![Q]\!]}(\langle Q, \eta\rangle \models \lozenge Q^{\surd})$$

$$\overset{I.H.}{=} wlp(P; wlp(Q, f))(\eta)$$

$$= wlp(P; Q, f)(\eta).$$

In $*$ we again rewrite each path into a prefix and a suffix and use positionality of policies. Additionally, observe that diverging paths are also split up into paths that already diverge before reaching an exit state of P and paths that do reach the end of P but diverge before reaching an exit state of Q. The probability of the former is captured by $Pr^{\mathfrak{P}}\{\langle P, \eta\rangle \not\models \lozenge P^{\surd}\}$ and the probability of the latter is the product of the probability of the prefix and the suffix whose probability is captured by $r^{q}_{P^{\surd}}$.

- Consider the loop while$(G)\{P\}$. Again we prove this case by induction on the number of iterations that a while-loop performs. Let $(\mathrm{while}(G)\{P\})^k$ be defined as in the proof of the previous theorem. We show for every k that

$$wlp((\mathrm{while}(G)\{P\})^k, f)(\eta) = LExpRew^{\mathcal{R}_f[\![(\mathrm{while}(G)\{P\})^k]\!]}(\eta). \qquad (3)$$

The only difference is now that

$$wlp((\mathsf{while}(G)\{P\})^{k+1}, f)(\eta) \leq wlp((\mathsf{while}(G)\{P\})^{k}, f)(\eta).$$

Using this we again know that the bounded while loop approximates the fixpoint given in Definition 16 (only this time from above). Formally this means

$$\lim_{k\to\infty} wlp((\mathsf{while}(G)\{P\})^{k}, f)(\eta) = wlp(\mathsf{while}(G)\{P\}, f)(\eta). \tag{4}$$

From (3) we know that for every k *LExpRew* behaves identically to *wlp*. Thus with (4) it follows that

$$wlp(\mathsf{while}(G)\{P\}, f)(\eta) = LExpRew^{\mathcal{R}_f[\![\, \mathsf{while}(G)\{P\} \,]\!]}(\eta).$$

It remains to prove (3). This is done by induction on k. Base case ($k = 0$):

$$\begin{aligned} wlp((\mathsf{while}(G)\{P\})^{0}, f)(\eta) &= wlp(\mathsf{abort}, f)(\eta) \\ &\overset{*}{=} LExpRew^{\mathcal{R}_f[\![\, \mathsf{abort} \,]\!]}(\eta) \\ &= LExpRew^{\mathcal{R}_f[\![\, (\mathsf{while}(G)\{P\})^0 \,]\!]}(\eta) \end{aligned}$$

$(*)$ was already shown earlier in the case abort.
Induction hypothesis: Eq. (3) holds for some unspecified but fixed value of k.
Induction step:

$$\begin{aligned} wlp((\mathsf{while}(G)\{P\})^{k+1}, f)(\eta) &= wlp(\mathsf{if}(G)\{P;\, (\mathsf{while}(G)\{P\})^{k}\}\mathsf{else}\{\mathsf{skip}\}, f)(\eta) \\ &= \big([G]\cdot wlp(P;\, (\mathsf{while}(G)\{P\})^{k}, f) + [\neg G]\cdot wlp(\mathsf{skip}, f)\big)(\eta) \\ &\overset{*}{=} \left([G]\cdot LExpRew^{\mathcal{R}_f[\![\, P;(\mathsf{while}(G)\{P\})^k \,]\!]} + [\neg G]\cdot LExpRew^{\mathcal{R}_f[\![\, \mathsf{skip} \,]\!]}\right)(\eta) \\ &= LExpRew^{\mathcal{R}_f[\![\, \mathsf{if}(G)\{P;(\mathsf{while}(G)\{P\})^k\}\mathsf{else}\{\mathsf{skip}\} \,]\!]}(\eta) \\ &= LExpRew^{\mathcal{R}_f[\![\, (\mathsf{while}(G)\{P\})^{k+1} \,]\!]}(\eta) \end{aligned}$$

$(*)$ follows from the induction hypothesis and the previously shown cases for skip and sequential composition. □

The weakest liberal pre-expectation $wlp(P, f)$ is thus the least expected value of f over any of the result distributions of P plus the probability that P does not terminate.

Example 25 (*Duelling Cowboys*)**.** Consider again the duelling cowboys example from Lst. 1. Assume we are interested in the probability that cowboy A wins the duel. In terms of the MDP semantics this means we are interested in

$$ExpRew^{(\mathcal{M},r)}(\langle 2, *, *\rangle \models \Diamond(\mathcal{M}, r)^{\surd})$$

where $\mathcal{M}$ is the MDP from Fig. 1 and r is the reward function that indicates whether cowboy A has won or not, i.e.

$$r(s) = \begin{cases} 1 & \text{if } s = \langle 11, A, 0\rangle \\ 0 & \text{otherwise.} \end{cases}$$

In this example the MDP is finite and this allows us to compute the desired expected cumulative reward easily. That is, cowboy A wins with probability at least

$$\frac{(1-b)a}{a+b-ab}.$$

Fig. 2 visualises this result. We nicely see how the expected winning chance depends on a and b. Parameterising our model allows us to carry out a calculation only once and make statements about all possible refinements of the system. According to Theorem 23 we can obtain the same result when applying the expectation transformer mechanism. The following section illustrates how this works. □

5. Analysis

Although the computation of (liberal) expected rewards on MDPs may be numerically involved, its basic idea is intuitive in principle. However, pGCL programs will often have an infinite state space in particular due to the infinite domain of the program variables. In that case it is not possible to compute the expected reward on the reward model in general. In contrast to this, the denotational semantics does not depend on the underlying state space but on the structure of the program. In this section we show how to determine a pre-expectation using *wp*-semantics.

Again let us determine the probability that cowboy A wins the duel. Therefore we choose $[t = A]$ as the post-expectation and determine $wp(\text{cowboyDuel}, [t = A])$.

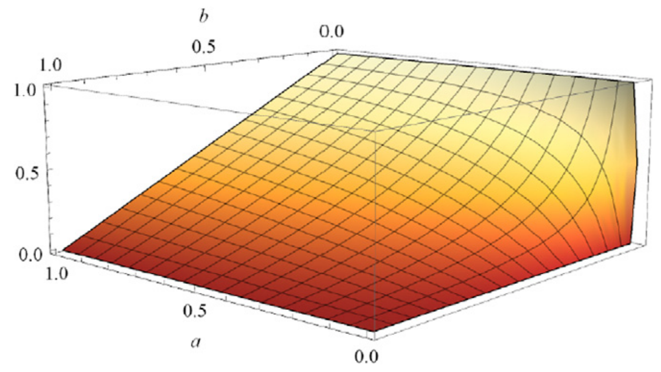


Fig. 2. Probability that A wins the duel, depending on *a* and *b*. Bear in mind that this is the least guaranteed probability that A wins. In the worst case (for A) cowboy B will shoot first and therefore as *b* tends to 1 the plot goes to 0, i.e. cowboy A has no chances. However for smaller values of *b* the influence of *a* increases.

Listing 2 shows the program cowboyDuel with annotations.

Listing 2: The duelling cowboys, annotated with expectations.

```
1   $\langle \frac{(1-b)a}{a+b-ab} \rangle$
2   $\langle \min\{\frac{a}{a+b-ab}, \frac{(1-b)a}{a+b-ab}\} \rangle$
3   (t := A [] t := B);
4   $\langle [t = A] \cdot \frac{a}{a+b-ab} + [t = B] \cdot \frac{(1-b)a}{a+b-ab} \rangle$
5   c := 1;
6   $\langle [t = A \wedge c = 0] \cdot 1 + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab} \rangle$
7   while (c = 1) {
8     $\langle [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab} \rangle$
9     $\langle [t = A \wedge c \neq 1] \cdot a + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 0] \cdot (1-b) + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab} \rangle$
10    if (t = A) {
11      (c := 0 [a] t := B);
12    } else {
13      (c := 0 [b] t := A);
14    }
15    $\langle [t = A \wedge c = 0] \cdot 1 + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab} \rangle$
16  }
17  $\langle [c \neq 1] \cdot \left( [t = A \wedge c = 0] \cdot 1 + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab} \right) \rangle$
18  $\langle [t = A] \rangle$
```

The program is annotated backwards according to the rules from Definition 15. In line 18 we start with the post-expectation that we are interested in. We finish with the sought probability in line 1. The only non-trivial step is to discover the so-called *invariant* which appears in lines 6 and 15. But let us assume for the moment that it is given. Then all other annotations are obtained by applying the syntactic rules from Definition 15 and rewriting. We can rewrite the expectation in line 9 into the expectation in line 8 because at this point the program must be in a state where $c = 1$ (the loop guard) holds and the expectations are equivalent for all these states. The same applies to expectations in lines 17 and 18 because at that point $c \neq 1$ holds. The calculation from line 15 to line 9 was already shown in Example 17. This means that once we have found the aforementioned invariant, the analysis can be automatically carried out by a computer — irrespective of the underlying state space size.

6. Invariants

The annotation in lines 6 and 15 of Lst. 2 which we call an invariant is an expectation that over-approximates the fixed point solution in Definition 15. In the following we use $\mathcal{I}$ for invariant expectations.

Definition 26 (*Probabilistic Invariant*)**.** An expectation $\mathcal{I}$ is called invariant for a loop while$(G)\{P\}$ if

$$\mathcal{I} \cdot [G] \leq wlp(P, \mathcal{I}). \quad \square \tag{5}$$

In our example, $\mathcal{I}$ is the expectation in line 6, G is the loop guard $c = 1$ and loop body P is the code in lines 10–14. In line 8, the expectation represents $\mathcal{I} \cdot [G]$ and line 9 is $wlp(P, \mathcal{I})$. Clearly, (5) is satisfied in our example.

6.1. Standard invariants

Before we elaborate more on invariants for probabilistic programs, let us consider invariants for traditional, non-probabilistic programs and how they are used. For this we remind the reader of the non-probabilistic version of Definition 26 along the lines of [1].

Definition 27 (*Standard Invariant*)**.** A predicate $\mathcal{I}$ is called invariant for a loop $\mathsf{while}(G)\{P\}$ if

$$\mathcal{I} \wedge G \Rightarrow wlp(P, \mathcal{I}). \quad \square \tag{6}$$

In this definition a single iteration of the loop body P is considered. The implication ensures that an execution of the loop body preserves the validity of $\mathcal{I}$. Note that G appears in the premise because we restrict our attention to states from which the loop will perform (at least) one iteration. States characterised by $\mathcal{I} \wedge \neg G$ are irrelevant because the loop will be skipped and one can trivially conclude that at the end of the loop's execution $\mathcal{I}$ is still true. Since (6) has to be satisfied on every iteration of the loop it follows that any execution beginning in a state that satisfies the invariant will terminate in a state that again satisfies the invariant (or the execution of the loop does not terminate). This property is colloquially summarised as "the set of states characterised by $\mathcal{I}$ is not left by the execution of the loop".

The key motivation for invariant annotations is that they establish the following relationship:

$$\mathcal{I} \Rightarrow wlp(\mathsf{while}(G)\{P\}, \mathcal{I} \wedge \neg G).$$

This relationship is called *partial correctness*. It means that every execution of the loop from a state satisfying the invariant can only terminate in a state that also satisfies the invariant and violates the guard G. The correctness is partial because it is possible that there are some executions which never terminate. In a separate proof, e.g. using a loop variant, one can establish that the loop terminates when started in some state in $\mathcal{I}$. This gives us *total correctness*:

$$\mathcal{I} \Rightarrow wp(\mathsf{while}(G)\{P\}, \mathcal{I} \wedge \neg G).$$

In practice one usually wants to prove that given some precondition *pre* before the beginning of the loop, the postcondition *post* will hold after the loop's execution. The straightforward way is to show this by directly applying *wp* semantics, i.e. proving

$$pre \Rightarrow wp(\mathsf{while}(G)\{P\}, post).$$

But it turns out to be hard because this requires to find the least fixed point of the loop with respect to *post*. Although that fixed point is mathematically well-defined it often is difficult to compute it in practice. Instead it usually[3] is easier to

1. find a predicate $\mathcal{I}$ such that

   $$pre \Rightarrow \mathcal{I} \quad \text{and} \quad \mathcal{I} \wedge \neg G \Rightarrow post,$$

2. show $\mathcal{I}$ is invariant for the loop $\mathsf{while}(G)\{P\}$, cf. Definition 27 and
3. prove that the loop terminates from any state in $\mathcal{I} \wedge G$.

Via this detour the same relation between *pre* and *post* is established as

$$pre \Rightarrow \mathcal{I} \Rightarrow wp(\mathsf{while}(G)\{P\}, \mathcal{I} \wedge \neg G) \quad \text{and} \quad \mathcal{I} \wedge \neg G \Rightarrow post.$$

6.2. Probabilistic invariants

Let us now return to probabilistic programs. In the probabilistic setting, we use an invariant $\mathcal{I}$ in the same way but this time it is an expectation. The post-expectation $\mathcal{I} \cdot [\neg G]$ has the pre-expectation $\mathcal{I}$.

However there is a crucial difference between non-probabilistic and probabilistic programs. Once we have shown that the loop in a non-probabilistic program terminates we have at the same time established that the set of reachable states is finite. This is because in a non-probabilistic program there may be several different executions from a given state due to non-determinism but the proof of termination shows that there are only finitely many emanating executions and each of them has finite length. For probabilistic programs the situation is somewhat different. Consider the example in Lst. 3 below.

[3] One can construct a loop and pre- and postconditions such that the alternative approach turns out to be as hard as finding the fixed point. In practice however there is a big difference.

Listing 3: Symmetric random walk over $\mathbb{N}$ with absorbing barrier at zero [4].

```
n := 1;
while(n != 0){
  (n := n - 1 [0.5] n := n + 1);
}
```

This loop will terminate with probability one[4] because the probability of an infinite walk is zero. However there exist infinitely many different walks of finite length from the initial state. Each of these walks has a positive probability. To convince ourselves that this indeed makes a difference we choose the invariant $\mathcal{I} = n$. It does satisfy Definition 26. Hence,

$$n \leq wlp(\mathsf{while}(n \neq 0)\{n := n - 1\,[0.5]\,n := n + 1\}, n \cdot [n = 0]).$$

And we have already shown that the loop terminates almost surely. We could therefore falsely conclude that

$$n \leq wp(\mathsf{while}(n \neq 0)\{n := n - 1\,[0.5]\,n := n + 1\}, n \cdot [n = 0]).$$

This would "prove" that the expected value of $n \cdot [n = 0]$ depends on the initial value of n and in the given example it is one. This of course is wrong as $n \cdot [n = 0]$ is zero everywhere. It is a nice exercise to compute the fixed point of this loop w.r.t. n. It gives a function that evaluates to zero for every n which coincides with the intuition that the expected outcome, in fact the only possible one, is zero. This also nicely shows that establishing termination is not the same as establishing termination with probability one.

In conclusion, given a probabilistic loop $\mathsf{while}(G)\{P\}$ and a post-expectation *post*, we can establish an upper bound for pre-expectation *pre* if we

1. find an expectation $\mathcal{I}$ such that

   $$pre \leq \mathcal{I} \quad \text{and} \quad \mathcal{I} \wedge \neg G \leq post,$$

2. show $\mathcal{I}$ is invariant for the loop $\mathsf{while}(G)\{P\}$, cf. Definition 26,
3. prove that the loop terminates from any state in G with probability one and
4. either show that from every initial state of the loop only a finite state space is reachable
   or make sure that $\mathcal{I}$ is bounded from above by some fixed constant
   or show that $wp(P, \mathcal{I} \cdot [G])$ tends to zero as the number of iterations tends to infinity.

The last item gives sufficient conditions to make reasoning with invariants sound for probabilistic programs [4, pp. 71–72].

Even though we argue that finding invariants and proving termination separately is easier than computing the least fixed point of the loop directly, it still remains a hard task to find a non-trivial invariant. Trivial invariants are constant functions such as 0 or 1 which are invariant for every loop but will hardly be useful for calculating an expectation. The invariant generation process is a topic on its own and beyond the scope of this paper. For probabilistic programs we have developed a constraint-based approach to generate invariants [19]. These ideas were implemented in our recently developed prototype tool PRINSYS [12]. It helps the user to find invariants for probabilistic programs semi-automatically. For instance, it was applied to our running example, the duelling cowboys, to calculate cowboy A's winning probability in Section 5.

In the rest of this section we apply our theoretical set-up to obtain an operational view of invariants for probabilistic systems.

6.3. Characterising invariants operationally

Recall that invariants for non-probabilistic programs are predicates, and they characterise the set of states that is never left by a loop as discussed before. This yields a straightforward operational characterisation of invariants: in a transition system that represents the loop we can identify a set of states with the property that one iteration of the loop started in such a state will end in this set again. If we can describe this set by a predicate this predicate will satisfy Definition 27.

For probabilistic programs the situation is more complicated. Invariants are expectations and not predicates and therefore do not necessarily characterise states. Instead they assign values to states such that an execution of the loop body started from a state with value e will *on average* end in a state with a value no less than e (if the execution terminates at all). Thus invariants establish lower bounds on pre-expectations. In fact, this idea generalises standard invariants which can be represented as $\{0, 1\}$-valued expectations.

Theorem 24 allows us to give an operational interpretation to invariants for probabilistic programs in terms of MDPs. Given an MDP for a while loop that is constructed according to the rules in Table 1 we can characterise an invariant in the following way:

[4] Also referred to as *almost* sure termination.

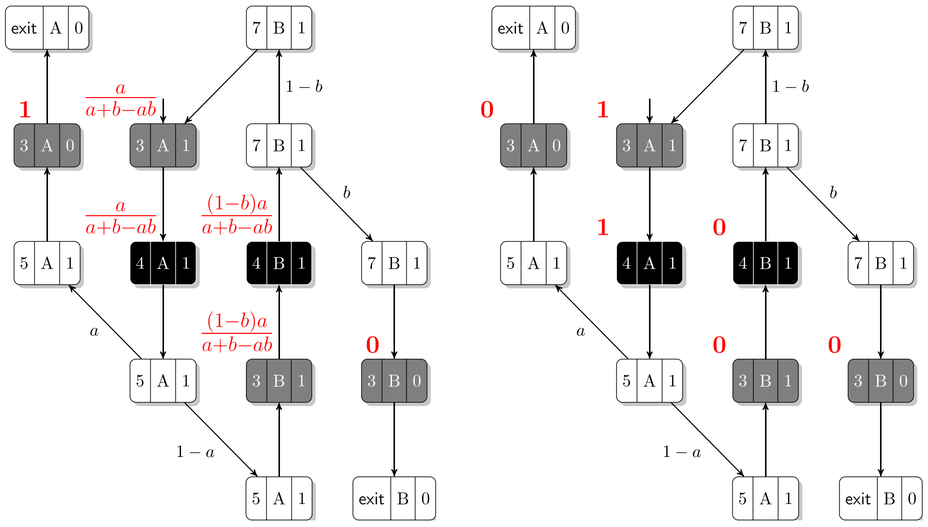
Corollary 28 (*Operational Interpretation of Quantitative Loop Invariants*). *An expectation $\mathcal{I}$ is invariant for the loop* $\mathsf{while}(G)\{P\}$ *if in $\mathcal{M}[\![\,\mathsf{while}(G)\{P\}\,]\!]$ for any state s of the form $\langle P; \mathsf{while}(G)\{P\}, \eta\rangle$:*

$$\mathcal{I}(\eta) \leq LExpRew^{\mathcal{R}_{\mathcal{I}}[\![P]\!]}(\langle P, \eta\rangle \models \Diamond P^{\surd}).$$

Intuitively this formula requires that the liberal expected reward w.r.t. the execution of the loop body P and the post-expectation $\mathcal{I}$ is bounded from below by $\mathcal{I}$. Corollary 28 is an immediate consequence of Definition 26 and Theorem 24. On the left hand side we evaluate $\mathcal{I}$ on η in those states where the loop has just been entered. This corresponds to the left hand side of the inequality in Definition 26. For the right hand side we have applied Theorem 24 to the right hand side of Definition 26. Thus the operational characterisation of invariants requires an expectation to meet a set of inequality constraints. Any expectation that is a solution to this set of constraints will satisfy Definition 26 as well. The difference between Definition 26 and Corollary 28 is that the former gives one inequality between two functions and this inequality is obtained by applying the *wlp* calculus while the latter gives one inequality for every possible initial state of the loop and each inequality is obtained by calculating a reachability objective in an MDP.

Below is an example for two different expectation functions, one of which is invariant and the other is not.

Example 29 (*Invariants*).



Both MDPs represent the loop from our running example in Lst. 1. Here the solid black states are of the form $\langle P; \mathsf{while}(G)\{P\}, \eta\rangle$ and the grey shaded states are the ones where one iteration of the loop has finished.

Let us once again consider the expectation

$$f = [t = A \wedge c = 0] \cdot 1 + [t = A \wedge c = 1] \cdot \frac{a}{a+b-ab} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a+b-ab}.$$

We evaluate this expectation in the black and grey states of the left MDP. According to Corollary 28 the following two inequalities have to be checked:

$$\begin{aligned} f(A,1) = \frac{a}{a+b-ab} &\leq \frac{a}{a+b-ab} \\ &= a \cdot 1 + (1-a) \cdot \frac{(1-b)a}{a+b-ab} \\ &= a \cdot f(A,0) + (1-a) \cdot f(B,1) \quad \checkmark \\ f(B,1) = \frac{(1-b)a}{a+b-ab} &\leq \frac{(1-b)a}{a+b-ab} \\ &= b \cdot 0 + (1-b) \cdot \frac{a}{a+b-ab} \\ &= b \cdot f(B,0) + (1-b) \cdot f(A,1) \quad \checkmark. \end{aligned}$$

Both are fulfilled and hence f is invariant.

Now let us choose a different expectation, for instance

$$g = [t = A] \cdot c.$$

The expectation g is evaluated on the black and grey states of the right MDP and the inequalities given by Corollary 28 become:

$$\begin{aligned} g(A,1) &= 1 \not\leq 0 \\ &= a\cdot 0 + (1-a)\cdot 0 \\ &= a\cdot g(A,0) + (1-a)\cdot g(B,1) \quad \times \\ g(B,1) &= 0 \leq 1-b \\ &= b\cdot 0 + (1-b)\cdot 1 \\ &= b\cdot g(B,0) + (1-b)\cdot g(A,1) \quad \checkmark. \end{aligned}$$

We find out that g is not invariant because the inequality corresponding to state $\langle 4, A, 1\rangle$ does not hold. □

In this section we have discussed the purpose of invariants and their properties. Furthermore we have shown an operational characterisation for invariants for probabilistic programs. As mentioned above, in other work [19] we have developed a technique to support the semi-automatic discovery of loop invariants which can be used to verify probabilistic properties of while loops.

7. Conclusion

This paper provides a formal connection between the expectation transformer semantics of pGCL by McIver and Morgan [4] and a simple operational semantics using (parametric) MDPs. This yields an insightful relationship between semantics used for deductive reasoning for probabilistic programs and the notion of a computation in terms of an MDP. Our approach assigns rewards to terminal states (only), and establishes that expected cumulative rewards correspond to *wp*-semantics. A slight variant of expected rewards yields a connection to the *wlp*-semantics.

The presented operational semantics provides a connection to model-checking algorithms for MDPs. In case a pGCL program yields a finite (parameterised) MDP, expected cumulative rewards (and thus weakest pre-expectations) can be computed by solving rational functions [20]. Future research will focus on exploiting results on the analysis of infinite MDPs (such as one-counter MDPs) to the verification of pGCL programs.

Acknowledgements

This work is supported by the DFG Research Training Group 1298 "AlgoSyn", the iMQRES scholarship, the EU FP7 project CARP (Correct and Efficient Accelerator Programming), the DFG–NWO bilateral ROCKS project and by Australian Research Council Discovery Grant DP1092464. The authors would like to thank Daniel Stan (ENS Cachan) who during his internship at the RWTH University contributed to the proof that appears in the Appendix. Additionally, we thank Tahiry Rabehaja and the anonymous reviewers for their valuable feedback.

Appendix. Proofs

In the proofs of Theorems 23 and 24 we rely on the fact that the *wp* and *wlp* semantics of a loop can be seen as the limit of a loop that may perform only n steps where n tends to infinity. This is a consequence of one of the renowned fixed point theorems. In [18] the authors discuss several similar fixed point theorems due to Kleene, Knaster and Tarski. We justify our proof by what they propose as the "folk theorem":

Theorem 30 (*Theorem 3 in [18]*). *Every continuous function F over a cpo has a least fixed point which is* $\mathrm{lub}_{n\in\mathbb{N}}\{F^n(\bot)\}$. □

Note that "continuous" means Scott-continuous here. In the following we define the necessary notions to understand the theorem and subsequently show that our expectation transformer $wp(P,\cdot)$ is indeed a Scott-continuous function. Hence we can characterise the behaviour of a loop by the supremum[5] over the behaviours of bounded loops. Finally, since $wp((\mathsf{while}(G)\{P\})^k,\cdot)$ is monotonically increasing we can exchange supremum for limit. Analogously, $wlp((\mathsf{while}(G)\{P\})^k,\cdot)$ is monotonically decreasing and the infimum can be replaced by the limit as well.

The first necessary notion is that of a directed set.

Definition 31 (*Directed Set*). A non-empty set D is directed if for all $x, y \in D$, there exists $z \in D$ such that $z \geq x$ and $z \geq y$. □

The expectation space $(\mathbf{E}, \leq)$ (cf. Definition 14, page 6) is directed because for any two expectations we can find an expectation which is (pointwise) greater than both, for example by taking their pointwise supremum.

Definition 32 (*Complete Partial Order*). A set C is a (directed) complete partial order (cpo) if for every directed subset $D \subseteq C$ the supremum of D exists and lies in C. □

5 Least upper bound (lub) and supremum (sup) mean the same.

$(\mathbf{E}, \leq)$ is not a cpo in general because there is no top element. Hence infinitely ascending chains are possible in $(\mathbf{E}, \leq)$. So for technical reasons we assume there is a top element ∞ in $\mathbf{E}$ such that for any directed subset D of expectations we can take the pointwise supremum which again is in $\mathbf{E}$. Alternatively we may restrict ourselves to pGCL programs for which we can prove that a bounded post-expectation will always be transformed to a bounded pre-expectation. This alternative approach is taken in [4].

Note that Theorem 30 above requires a bottom element $\perp$ in the cpo. In the domain of expectations this is the constant 0 function.

Definition 33 (*Scott-Continuous Function*)**.** Let C, C' be cpo's. A function $F : C \to C'$ is Scott-continuous if

- for all directed subsets $D \subseteq C$, the image $F(D)$ is directed and
- $F(\sup D) = \sup F(D)$. $\square$

This concludes our introduction of necessary definitions and we can now prove the following lemma that justifies the application of Theorem 30 in the proof of Theorem 23 (and similarly Theorem 24).

Lemma 34 (*Continuity of Expectation Transformers*)**.** *For every* pGCL *program P the expectation transformer $wp(P, \cdot)$ is a Scott-continuous function over $(\mathbf{E}, \leq)$. The same holds for $wlp(P, \cdot)$.* $\square$

Proof. The first point of Definition 33 follows immediately from monotonicity of $wp(P, \cdot)$.

It remains to show that for any directed subset D of expectations

$$wp(P, \sup_{f\in D} f) = \sup_{f\in D} wp(P, f).$$

The proof is carried out by induction on the structure of the program P.

Induction base:

- $P = \mathsf{skip}$:
$$wp(\mathsf{skip}, \sup_{f\in D} f) = \sup_{f\in D} f = \sup_{f\in D} wp(\mathsf{skip}, f).$$
- $P = \mathsf{abort}$:
$$wp(\mathsf{abort}, \sup_{f\in D} f) = 0 = \sup_{f\in D} wp(\mathsf{abort}, f).$$
- Let P be an assignment $x := E$:
$$\begin{aligned} wp(x := E, \sup_{f\in D} f) &= \left(\sup_{f\in D} f\right)[x/E] \\ &\overset{*}{=} \sup_{f\in D} (f[x/E]) \\ &= \sup_{f\in D} wp(x := E, f). \end{aligned}$$
For $(*)$ observe that one can construct the supremum expectation and then substitute x for the expression E or first do the substitution and then construct (the same) pointwise supremum.

Induction hypothesis: assume that for program P (and analogously for Q)

$$wp(P, \sup_{f\in D} f) = \sup_{f\in D} wp(P, f).$$

Induction step:

- Consider the sequential composition $P; Q$:
$$\begin{aligned} wp(P; Q, \sup_{f\in D} f) &= wp(P, wp(Q, \sup_{f\in D} f)) \\ &\overset{I.H.}{=} wp(P, \sup_{f\in D} wp(Q, f)) \\ &\overset{I.H.}{=} \sup_{f\in D} wp(P, wp(Q, f)). \end{aligned}$$

- Consider the probabilistic choice $P\,[p]\,Q$ (this also covers conditional choice because it can be rewritten as $P\,[[G]]\,Q$): A property of sup immediately provides an inequality:

$$\begin{aligned} wp(P\,[p]\,Q, \sup_{f\in D} f) &= p\cdot wp(P, \sup_{f\in D} f) + (1-p)\cdot wp(Q, \sup_{f\in D} f) \\ &\overset{I.H.}{=} \sup_{f\in D}\,(p\cdot wp(P,f)) + \sup_{f\in D}\,((1-p)\cdot wp(Q,f)) \\ &\geq \sup_{f\in D}\,(p\cdot wp(P,f) + (1-p)\cdot wp(Q,f)) \\ &= \sup_{f\in D} wp(P\,[p]\,Q, f). \end{aligned}$$

We can strengthen the inequality to equality. Assume for the purpose of contradiction that

$$\sup_{f\in D}\,(p\cdot wp(P,f) + (1-p)\cdot wp(Q,f)) < \sup_{f\in D}\,(p\cdot wp(P,f)) + \sup_{f\in D}\,((1-p)\cdot wp(Q,f))\,.$$

Then by definition of sup there must exist $g_1, g_2 \in D$ for which this strict inequality holds.

$$\begin{aligned} \sup_{f\in D}\,(p\cdot wp(P,f) + (1-p)\cdot wp(Q,f)) &< p\cdot wp(P,g_1) + (1-p)\cdot wp(Q,g_2) \\ &\overset{*}{\leq} p\cdot wp(P,h) + (1-p)\cdot wp(Q,h) \\ &\leq \sup_{f\in D}\,(p\cdot wp(P,f) + (1-p)\cdot wp(Q,f)) \quad \text{Contradiction!} \end{aligned}$$

Where in $(*)$ we use that D is directed and therefore there is an $h \in D$ with $g_1 \leq h$ and $g_2 \leq h$. And since $wp(P,\cdot)$ is monotonous for any P, the summands cannot decrease and hence the sum cannot decrease which gives the (non-strict) inequality.

- Consider the non-deterministic choice $P\,[]\,Q$. The proof for this case goes along the same lines as for probabilistic choice: A property of sup immediately provides an inequality:

$$\begin{aligned} wp(P\,[]\,Q, \sup_{f\in D} f) &= \min\left\{wp(P, \sup_{f\in D} f), wp(Q, \sup_{f\in D} f)\right\} \\ &\overset{I.H.}{=} \min\left\{\sup_{f\in D} wp(P,f), \sup_{f\in D} wp(Q,f)\right\} \\ &\geq \sup_{f\in D} \min\{wp(P,f), wp(Q,f)\} \\ &= \sup_{f\in D} wp(P\,[]\,Q, f). \end{aligned}$$

We can strengthen the inequality to equality. Assume for the purpose of contradiction that

$$\sup_{f\in D} \min\{wp(P,f), wp(Q,f)\} < \min\left\{\sup_{f\in D} wp(P,f), \sup_{f\in D} wp(Q,f)\right\}.$$

Then by definition of sup there must exist $g_1, g_2 \in D$ for which this strict inequality holds.

$$\begin{aligned} \sup_{f\in D} \min\{wp(P,f), wp(Q,f)\} &< \min\{wp(P,g_1), wp(Q,g_2)\} \\ &\leq \min\{wp(P,h), wp(Q,h)\} \\ &\leq \sup_{f\in D} \min\{wp(P,f), wp(Q,f)\} \quad \text{Contradiction!} \end{aligned}$$

- Consider the loop $\mathtt{while}(G)\{P\}$.

  In Definition 15, the semantics of the loop (w.r.t. a post-expectation f) were defined as the least fixed point of a function

$$F(X) = [G]\cdot wp(P,X) + [\neg G]\cdot f.$$

For the sake of notation we additionally define:

$$F_{\sup}(X) = [G]\cdot wp(P,X) + [\neg G]\cdot \sup_{f\in D} f.$$

By the induction hypothesis $F(X)$ is Scott-continuous in X. Therefore we can apply Theorem 30:

$$\mu X.F(X) = \sup_{n\in\mathbb{N}} F^n(0).$$

Using this we deduce:

$$\begin{aligned} wp(\mathsf{while}(G)\{P\}, \sup_{f\in D} f) &= \sup_{n\in\mathbb{N}} F^n_{\sup}(0) \\ &= \sup_{n\in\mathbb{N}} (\sup_{f\in D} F)^n(0) \\ &\overset{*}{=} \sup_{n\in\mathbb{N}} \sup_{f\in D} F^n(0) \\ &\overset{**}{=} \sup_{f\in D} \sup_{n\in\mathbb{N}} F^n(0) \\ &= \sup_{f\in D} wp(\mathsf{while}(G)\{P\}, f). \end{aligned}$$

For $(*)$ one can show that $(\sup_{f\in D} F)^n(0) = \sup_{f\in D} F^n(0)$ by induction on n using the definition of F, directedness of D and continuity of $wp(P, \cdot)$ as per induction hypothesis.

The equality in $(**)$ can be established by applying the suprema to $F^n(0)$ one by one and showing inequality in both directions. The existence of these suprema is a consequence of the induction hypothesis which gives us the continuity of F and Theorem 30. □

The *wlp*-semantics for a loop is defined as the greatest fixed point solution of an equation, cf. Definition 16. We can adapt the proof above to $wlp(P, \cdot)$, if we reverse the direction of the directed set of expectations and bound expectation functions from above by 1 (as in Definition 16). The semantics of the loop will be the limit of $F^n(1)$ (where the constant expectation function 1 is the top element in the reversed cpo).

Remark 35 (*Continuity Proofs*)**.** Continuity was proven for regular transformers of expectations over finite and countable state spaces [4]. Our result generalises to uncountable state spaces but restricts to pGCL programs which induce a subset of regular transformers. □

References

[1] E.W. Dijkstra, A Discipline of Programming, Prentice Hall, 1976.
[2] J.J. Lukkien, An operational semantics for the guarded command language, in: MPC, in: LNCS, vol. 669, Springer, 1992, pp. 233–249.
[3] J.J. Lukkien, Operational semantics and generalized weakest preconditions, Sci. Comput. Program. 22 (1–2) (1994) 137–155.
[4] A. McIver, C. Morgan, Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science), Springer, 2004.
[5] R.A. Howard, Dynamic Programming and Markov Processes, MIT Press, 1960.
[6] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley and Sons, 1994.
[7] C. Baier, F. Ciesinski, M. Größer, Probmela: a modeling language for communicating probabilistic processes, in: Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004, MEMOCODE'04, IEEE, 2004, pp. 57–66.
[8] A. Di Pierro, C. Hankin, H. Wiklicky, Program analysis probably counts, Comput. J. 53 (6) (2010) 871–880.
[9] D. Kozen, Semantics of probabilistic programs, J. Comput. System Sci. 22 (3) (1981) 328–350.
[10] J. He, K. Seidel, A. McIver, Probabilistic models for the guarded command language, Sci. Comput. Program. 28 (2–3) (1997) 171–192.
[11] C. Jones, Probabilistic non-determinism, Ph.D. thesis, University of Edinburgh, 1989.
[12] F. Gretz, J.-P. Katoen, A. McIver, Prinsys — on a quest for probabilistic Loop invariants, in: K.R. Joshi, M. Siegle, M. Stoelinga, P.R. D'Argenio (Eds.), QEST, in: Lecture Notes in Computer Science, vol. 8054, Springer, 2013, pp. 193–208.
[13] F. Gretz, J.-P. Katoen, A. McIver, Operational versus weakest precondition semantics for the probabilistic guarded command language, in: QEST, IEEE Computer Society, 2012, pp. 168–177.
[14] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
[15] J. den Hartog, E.P. de Vink, Verifying probabilistic programs using a hoare like logic, Internat. J. Found Comput. Sci. 13 (3) (2002) 315–340.
[16] G. Barthe, B. Köpf, F. Olmedo, S.Z. Béguelin, Probabilistic relational reasoning for differential privacy, in: J. Field, M. Hicks (Eds.), POPL, ACM, 2012, pp. 97–110.
[17] O. Celiku, A. McIver, Cost-based analysis of probabilistic programs mechanised in HOL, Nord. J. Comput. 11 (2) (2004) 102–128.
[18] J.-L. Lassez, V.L. Nguyen, L. Sonenberg, Fixed point theorems and semantics: a folk tale, Inf. Process. Lett. 14 (3) (1982) 112–116.
[19] J.-P. Katoen, A. McIver, L. Meinicke, C.C. Morgan, Linear-invariant generation for probabilistic programs: - automated support for proof-based methods, in: R. Cousot, M. Martel (Eds.), SAS, in: Lecture Notes in Computer Science, vol. 6337, Springer, 2010, pp. 390–406.
[20] E.M. Hahn, H. Hermanns, L. Zhang, Probabilistic reachability for parametric Markov models, STTT 13 (1) (2011) 3–19.

Friedrich Gretz received his diploma in computer science from the RWTH Aachen University in 2011. Since then he is a doctoral student under the joint supervision of Joost-Pieter Katoen and Annabelle McIver. Friedrich is a member of the DFG research training group AlgoSyn at the RWTH and an iMQRES scholarship holder at Macquarie University. His research interest is focused on semantics and verification of probabilistic systems.

Joost-Pieter Katoen is a distinguished professor at the RWTH Aachen University (since 2004) and is part-time affiliated to the University of Twente. His research interests are concurrency theory, model checking, timed and probabilistic systems, and semantics. He co-authored more than 200 journal and conference papers, and co-authored the book "Principles of Model Checking". He is a member of the Academia Europaea, IFIP WG 1.8 and WG 2.2, a member of the steering committees of the conferences ETAPS (chair), CONCUR, FORMATS, and QEST (chair), and a member of the editorial board of STTT.

Annabelle McIver holds degrees in mathematics from Cambridge (BA) and Oxford (D.Phil.), and has worked in industry. Currently she is an associate professor in the Department of Computing at Macquarie University, Sydney. She has published widely in formal methods on topics ranging from highly theoretical (quantitative modal algebra) to extremely practical (automatic correctness verifiers for probabilistic systems). Her (joint) text book "Abstraction, refinement and proof for probabilistic systems" is the only full research text on probabilistic program refinement theory.