Conditioning in Probabilistic Programming[*]

FEDERICO OLMEDO, Department of Computer Science, University of Chile, Chile
FRIEDRICH GRETZ, Bosch Corporate Research, Germany
NILS JANSEN, Radboud University Nijmegen, The Netherlands
BENJAMIN LUCIEN KAMINSKI, RWTH Aachen University, Germany
JOOST-PIETER KATOEN, RWTH Aachen University, Germany
ANNABELLE MCIVER, Macquarie University, Sydney, Australia

This paper investigates the semantic intricacies of conditioning, a main feature in probabilistic programming. Our study is based on an extension of the imperative probabilistic guarded command language pGCL with conditioning. We provide a weakest pre–condition (wp) semantics and an operational semantics. To deal with possibly diverging program behaviour we consider liberal pre–conditions. We show that diverging program behaviour plays a key role when defining conditioning. We establish that weakest pre–conditions coincide with conditional expected rewards in Markov chains—the operational semantics—and that the wp–semantics conservatively extends the existing semantics of pGCL (without conditioning). An extension of these results with non–determinism turns out to be problematic: although an operational semantics using Markov decision processes is rather straightforward, we show that providing an inductive wp–semantics in this setting is impossible. Finally, we present two program transformations which eliminate conditioning from any program. The first transformation hoists conditioning while updating the probabilistic choices in the program, while the second transformation replaces conditioning—in the same vein as rejection sampling—by a program with loops. In addition, we present a last program transformation that replaces an independent identically distributed loop with conditioning.

CCS Concepts: • **Theory of computation → Semantics and reasoning**; **Probabilistic computation**;

Additional Key Words and Phrases: Probabilistic programming, conditioning, weakest pre–condition semantics, operational semantics.

ACM Reference format:
Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle McIver. 2017. Conditioning in Probabilistic Programming. *ACM Trans. Program. Lang. Syst.* 9, 4, Article 39 (March 2017), 50 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

Probabilistic programs support random choices like "execute program c_1 with probability $1/3$ and program c_2 with probability $2/3$". Probabilistic programs are ordinary sequential programs describing posterior probability distributions. Describing randomised algorithms has been the classical

[*]This work was supported by the Excellence Initiative of the German federal and state government and the CDZ project CAP (GZ 1023).

0164-0925/2017/3-ART39 $15.00
https://doi.org/0000001.0000001

application of these programs. Applications in biology, machine learning, quantum computing, security, and so on, have led to a rapidly growing interest in probabilistic programs in the last decade [24]. Several probabilistic programming languages have been recently developed such as *Probabilistic C* [44], *Rely* [11], *Figaro* [45], *ProbLog* [20], *Tabular* [23], *webPPL* [22] and *R2* [42]. Most of these languages feature, in addition to sampling from probability distributions, the ability to *condition* values of variables in a program. Conditioning allows for adding information about observed events into the program that may influence the posterior distribution. It is one of the key features in Bayesian networks which rely on Bayes' rule as the basis for updating information. It is this feature that distinguishes modern probabilistic programming languages from those in the early days describing randomised algorithms.

The semantics of probabilistic programs without conditioning is rather well–understood. The seminal work by Kozen [37] provides a denotational semantics of a simple imperative probabilistic programming language. A probabilistic extension of propositional dynamic logic for fully probabilistic programs was provided in [38]. This was extended in McIver and Morgan [40] with a weakest pre–condition (wp) semantics covering demonic non–determinism and proof rules for loops. Proof rules for handling mixed–sign random variables are presented in [33]. In those approaches, one takes into account that due to its random nature, the final state of a program on termination is not unique. Thus, rather than a mapping from inputs to outputs—as in Dijkstra's approach—probabilistic programs map initial states to a *distribution* on their possible final states. More precisely, one obtains *sub*–distributions where the "missing" probability mass represents the likelihood of divergence. Given a random variable f and an initial state s, a central issue is to determine f's expected value upon the probabilistic program's termination. The wp–approach has been automated in the theorem provers HOL and Isabelle [16, 29]. [27] has pursued a similar approach, while [25] showed the relation between an operational semantics using Markov decision processes and the wp–semantics of [40]. Other related directions include Hoare logics [18] and semantics of constraint probabilistic programming languages [26]. These existing works do not consider the notion of conditioning. *A primary goal of this paper is to study the* wp*–approach for a probabilistic programming language with conditioning, and if possible, demonic non–determinism.*

The treatment of conditioning as present in modern probabilistic programming languages does impose several challenging problems. We discuss these intricacies in the setting of a simple imperative language, a probabilistic variant of Dijkstra's guarded command language, referred to as pGCL [40]. Admittedly, this is not a language used nowadays in probabilistic programming, but due to its simplicity it can be considered as a "core" language in which the semantic intricacies of conditioning can be properly illustrated. Its main restriction is that it does not support continuous distributions whereas the aforementioned languages such as R2 [42] and webPPL [22] do. The problems discussed here do, however, also occur when considering such distributions. On the other hand, we also consider an extension with a non–deterministic choice. This is essential for considering probabilistic programs at different abstraction levels—abstraction of a program variable naturally gives rise to non–determinism [36]—and for including multi–threading. We focus on conditioning as expressed by means of so–called observe statements [9, 15, 28, 42].

The Semantic Intricacies of Conditioning

We discuss the main semantic intricacies of conditioning by means of small examples.

When to observe? Consider the program snippet

$$c\colon \quad \{x \coloneqq 0\}\ [1/2]\ \{x \coloneqq 1\};\ \texttt{observe}\ (x = 1)\ ,$$

which assigns zero to the variable x with probability $1/2$ (modelled by a probabilistic choice) while x is assigned one with the same likelihood, after which we condition to the outcome of x to being one. The `observe` statement blocks all runs violating its condition and renormalises the probabilities of the remaining (called: valid) runs. The interpretation of the program is the expected outcome conditioned on the valid runs. For c, this yields for the value of the variable x (the outcome) one after conditioning.

Consider now:

$$\{x := 0;\ \texttt{observe}\,(x = 1)\}\ [1/2]\ \{x := 1;\ \texttt{observe}\,(x = 1)\}$$

The left branch of the probabilistic choice is infeasible as it has no valid runs. Is this program equivalent to c? In our approach they are. Setting an infeasible program into context thus can render it feasible.

The interference with non–termination. Consider

$$x := 2 \qquad\qquad \text{and} \qquad\qquad \{x := 2\}\ [1/2]\ \{\texttt{abort}\}\ .$$

Both programs assign two to x, but the right one aborts with probability $1/2$. Should these two programs be considered equivalent or not? Some semantics such as [42] do not distinguish them, as they assume programs to almost–surely terminate, i.e. terminate with probability one. This may make sense for programs in certain application domains. But can we really require a “probabilistic programmer" to only write almost–surely terminating programs? Sure, one can (syntactically) prevent a programmer from writing programs containing `abort` statements, but one cannot avoid divergence—programs with loops may easily not terminate. We advocate a semantics that can distinguish almost–surely terminating programs from those having a positive probability to diverge. The above two programs are thus distinguished. Such semantics is also needed to analyse termination, a key question in program termination. This is a non-trivial analysis aspect as checking almost–sure termination of probabilistic programs is “more undecidable” than termination for ordinary programs [32].

Observations inside loops. Consider the two programs:

```
repeat {                    repeat {
    x := 1                      {x := 1} [1/2] {x := 0};
} until (x = 0)                 observe (x = 1)
                            } until (x = 0)
```

The left program certainly diverges. For the program on the right, things are not so clear any more: On the one hand, the only non–terminating run is the one in which in every iteration x is set to 1. This event of setting x infinitely often to 1, however, has probability 0. So the probability of non–termination would be 0. On the other hand, the *global* effect of the observe statement within the loop is to condition on exactly this event, which occurs with probability 0. The principle for deciding on a semantics that makes sense is that the results should be consistent with the usual definition of conditional probabilities. For the program on the right the semantics should be equivalent to conditioning on the event “observe $x = 1$ infinitely often”, an event with probability 0, and it is for this reason that the semantics for this program should be undefined. Note that programs with (probabilistic) assertions must be loop–free to avoid similar problems [48]; other approaches insist on the absence of diverging loops [13]. While in this sample program it is immediate to see that the event to which we condition has probability zero, in general it might be highly non–trivial to identify this. Demanding from a “probabilistic programmer" to condition only to events with non–zero probability would thus be just as (if not even more) far–fetched as requiring an “ordinary programmer" to write only terminating programs. Therefore, a semantics for conditioning has to take the possibility of conditioning to zero–probability events into account. We propose such a semantics and it distinguishes the two programs with loops above.

The interference with non–determinism. The following example blurs the situation even further. Consider the program:

```
repeat {
    {x := 1} [1/2] {x := 0};
    {x := 1} □ {observe (x = 1)}
} until (x = 0)
```

This program first randomly sets x to 1 or 0. Then it either sets x to 1 or conditions to the event that x was set to 1 in the previous probabilistic choice. The latter choice is made non–deterministically and therefore the semantics of the entire program is not clear: If in line 3, the oracle to resolve the non–determinism always chooses $x := 1$, then this results in certain non–termination. If, on the other hand, the oracle always chooses observe $(x = 1)$, then the global effect of the observe statement is a conditioning to this zero–probability event. Which behaviour of the oracle is more demonic? We take the point of view that certain non–termination is a more well–behaved phenomenon than conditioning to a zero–probability event. Therefore a demonic oracle should prefer the latter.

Contributions of This Paper

This paper provides a semantics of pGCL with conditioning. This includes probabilistic choice, abortion and conditioning by means of observe statements. Given that this language is rather basic, our semantics can act as a backbone for full–fledged imperative probabilistic programming languages with conditioning. We provide a wp–semantics in the style of [40, 41] and present an operational model based on Markov decision processes [46]. In the absence of non–determinism, this reduces to Markov chains. The crux of our semantics is to distinguish the violation of observe statements and possible divergence. The probability that a given outcome is obtained is normalised with respect to the probability that all observe statements are fulfilled, even when they pertain to infinitary events. The latter probability includes possibly diverging runs.

The proposed solution is to define the semantics of a program c with respect to random variable f by a pair, consisting of the wp–semantics of c with respect to f and its liberal wp–semantics with respect to $\mathbf{1}$, the constant function yielding one for each program state. The latter component stands for the probability of all valid runs. This includes valid diverging runs, too. We consider this as a key issue in our semantics. The incorporation of diverging program runs is the main difference to the semantics in languages such as R2 [42] or [15].

The soundness of the semantics is investigated in two directions. The wp–semantics is shown to be semantically equivalent to the operational model in the sense that (roughly speaking) *weakest pre–expectations correspond to conditional expected rewards* in Markov chains. Moreover, this semantics is *a conservative extension of McIver, Seidel and Morgan's semantics* [41] in the sense that our semantics of programs without conditioning coincides[1]. To be more precise, this latter soundness result only holds for programs without non–determinism. In fact, it turns out that *combining non–determinism and conditioning cannot be treated* using the inductive style of the wp–semantics. The problem is that the resolution of non–deterministic choices needs to depend on the context of these choices, rendering a definition by structural induction on programs—as is *the* standard approach for defining

[1] Given that their semantics conservatively extends Dijkstra's guarded command language, we consider this as a desirable property.

wp–semantics—impossible. We treat this problem in detail in the paper, and provide an operational semantics for non–deterministic programs using Markov decision processes.

As an application of our semantics, we treat three program transformations. The first transformation removes `observe` statements from a program by hoisting them through the probabilistic choices in the program. This technique thus modifies the likelihood of probabilistic choices in the program based on the Boolean conditions in its `observe` statements. The result is a program without conditioning. This transformation is similar in nature to the one in [42], where all programs are assumed to be terminating. Due to the treatment of possible divergence, in our setting the transformation to eliminate conditioning is different and more involved. This transformation is complemented by an alternative transformation for removing conditioning. Let c be a program with observations. We transform this program by repeatedly sampling executions from c until the sampled execution satisfies all its observations. If during a program execution we encounter that an `observe` is violated, we restart the program as being fresh. This comes at the expense of introducing a loop. This program transformation has similarities to the application of rejection sampling to conditional probabilities as described in, e.g., [49]. These two program transformations thus show that conditioning is syntactic sugar as it can be either resolved in the wp–calculations or be replaced by a loop. Our third and last program transformation goes in the reverse direction: in case the successive loop iterations are statistically independent, a loop can be replaced by an `observe` statement, which has the same effect.

Besides being of interest on their own right, a particularly appealing application of these transformations is to ease the reasoning about probabilistic program termination, problem that is known to be strictly harder than in the non–probabilistic case [32]. Since the presented transformations are valid irrespective of the termination probability of the original programs, we can use the transformed—possibly simpler—programs to reason about the termination probability of the original programs.

Organisation of the paper. Section 2 provides an informal introduction to our approach and introduces our running example for this paper. Section 3 introduces the imperative probabilistic programming language pGCL extended with conditions. Section 4 presents our wp–semantics, while Section 5 presents the operational semantics and the correspondence between both semantics. Section 6 extends the operational semantics for a language incorporating a non–deterministic choice and presents our impossibility result for combining conditioning and non–determinism in an inductive wp–semantics. Section 7 covers the three program transformations that remove conditioning, and that replace a loop by an `observe`. Section 8 discusses related work, whereas Section 9 concludes the paper. Omitted proofs from the main part of the paper are included in the Appendix.

This work builds on a previous work from the authors [31] and extends it with the following contributions: a proof rule for reasoning about the conditional pre–expectation of loops, a more thorough study of the properties of the conditional wp–transformer, a program transformation that replaces loops with no information flow across iterations by a simple observation and proofs of all the results. A high-level overview can be found in [34].

2 OVERVIEW

We provide an informal and high–level overview of our two semantic models for conditioned probabilistic programs. Further details are elaborated in Sections 4 and 5. As running example we use the "goldfish–piranha" problem from [51]:

> One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish

> bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?

We can formalise this problem in terms of the program in Figure 1. The translation is straightforward:

```
1  $f_1$ := gold [1/2] pir;
2  $f_2$ := pir;
3  $rem$ := $f_1$ [1/2] $f_2$;
4  observe ($rem$ = pir)
```

Fig. 1. Probabilistic program c_{fish} encoding the goldfish–piranha problem.

Variable f_1 represents the fish that was already in the fishbowl at the beginning, variable f_2 the (piranha) fish that was introduced afterwards, and variable *rem* the fish that was removed from the bowl at the end. The fact that this fish turned out to be a piranha is encoded using the observe statement in Line 4. To solve the problem, we must determine the probability that $f_1 = \text{pir}$ upon the program termination.

Despite being modelled by a four–line program, the goldfish–piranha problem is sophisticated enough to illustrate all the essential aspects of both our semantic models.

2.1 Operational Semantics

We present our operational model for c_{fish} first, as we believe it is the most intuitive and easiest to grasp. We model the program as a probabilistic transition system that reflects all possible program runs along with their probabilities. The transition system is depicted in Figure 2. States of the transition system represent states of the program execution; they are tagged with the program line after which they occur. For example, state $\boxed{3\,|\,\text{gold}\,|\,\text{pir}\,|\,\text{pir}}$ of the transition system reflects that the program state $\langle f_1 \mapsto \text{gold}, f_2 \mapsto \text{pir}, rem \mapsto \text{pir}\rangle$ is reached after Line 3 of the program execution. In particular, symbol "$\star$" in a variable slot indicates that the program has not set its value, yet. To reflect the random nature of the program, some transitions are probabilistic. In this case, a state includes multiple outgoing edges, each of them labelled with the respective probability.

The construction of the transition system is as follows: Before starting the program execution, the program state is unknown; in the transition system this is denoted by the initial state $\boxed{0\,|\,\star\,|\,\star\,|\,\star}$. In Line 1, the program sets f_1 to gold or to pir with the same likelihood, $1/2$; in the transition system we move, correspondingly, to states $\boxed{1\,|\,\text{gold}\,|\,\star\,|\,\star}$ and $\boxed{1\,|\,\text{pir}\,|\,\star\,|\,\star}$, with respective probabilities $1/2$. In Line 2, the program sets f_2 to pir; in the transition system we then move from the two previous states to states $\boxed{2\,|\,\text{gold}\,|\,\text{pir}\,|\,\star}$ and $\boxed{2\,|\,\text{pir}\,|\,\text{pir}\,|\,\star}$, respectively. The program then goes through Line 3 and the construction of the transition system proceeds as for Line 1. Finally, Line 4 of the program contains an observation. From state $\boxed{3\,|\,\text{gold}\,|\,\text{pir}\,|\,\text{gold}}$, the observation is violated; we signal this by transitioning to "undesired" state $\boxed{4\,|\,\lightning}$. The other two states reachable after Line 4, namely $\boxed{4\,|\,\text{gold}\,|\,\text{pir}\,|\,\text{pir}}$ and $\boxed{4\,|\,\text{pir}\,|\,\text{pir}\,|\,\text{pir}}$, represent, on the contrary, valid final program states as they passed the observation.

The transition system in Figure 2 describes the behaviour of program c_{fish}. From the system we can see that the program admits four runs. One of them is blocked because it violates the observation. The other three are valid program runs; two of them yield final state $\langle f_1 \mapsto \text{pir}, f_2 \mapsto \text{pir}, rem \mapsto \text{pir}\rangle$ and the remaining run yields state $\langle f_1 \mapsto \text{gold}, f_2 \mapsto \text{pir}, rem \mapsto \text{pir}\rangle$.

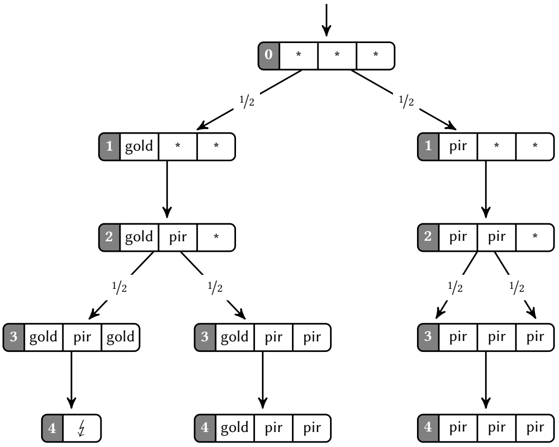


Fig. 2. Operational model for the program c_{fish} depicted in Figure 1. States are represented as rounded rectangles containing four elements: the program line, and the value of the program variables f_1, f_2 and *rem*, respectively

We can easily determine the probability $\Pr\left[c_{\mathit{fish}} \colon f_1 = \mathsf{pir}\right]$ that the program establishes $f_1 = \mathsf{pir}$ by examining the transition system. Due to the observation in Line 4, only the program runs that avoid the undesired state $\boxed{4 \mid ↯}$ remain. Their probabilities are normalised so that they sum up to one. We can thus compute $\Pr\left[c_{\mathit{fish}} \colon f_1 = \mathsf{pir}\right]$ as the quotient between

(1) the accumulated probabilities of all runs that elude "↯" *and* establish $f_1 = \mathsf{pir}$, and
(2) the accumulated probabilities of all runs that elude "↯".

This readily yields

$$\Pr\left[c_{\mathit{fish}} \colon f_1 = \mathsf{pir}\right] \;=\; \frac{1/2 \cdot (1/2 + 1/2)}{1/2 \cdot 1/2 \;+\; 1/2 \cdot (1/2 + 1/2)} \;=\; \frac{1/2}{3/4} \;=\; \frac{2}{3}\,, \tag{1}$$

and turning to our motivating problem, it says that the fish originally in the bowl happened to be a piranha with probability 2/3.

In Section 5 we will see that the transition system in Figure 2 slightly deviates from the actual transition system that we propose for program c_{fish} (cf. Figure 6). We deliberately did this to reduce technicalities and make the overview as accesible as possible. Despite these deviations, the model herein presented captures the essence of our operational semantics in a faithful and comprehensive manner.

2.2 Weakest Pre–Expectation Semantics

The other semantic model that we propose for conditioned programs is a quantitative extension of Dijkstra's weakest pre–condition semantics. There, the meaning of a classic sequential program c with state space $\mathbb{S}$ is given by the predicate transformer

$$\mathsf{wp}[c] \colon (\mathbb{S} \longrightarrow \{0,1\}) \longrightarrow (\mathbb{S} \longrightarrow \{0,1\})\;.$$

Given post–condition Q, the weakest pre–condition $\mathsf{wp}[c](Q)$ returns, for each initial state, 1 if the program establishes the post–condition and 0 if the program does not. For a probabilistic program, however, this binary outcome is not sufficient. Take for instance program c_{fish}. We can neither say that it establishes post–condition $f_1 = \mathsf{pir}$ nor that it fails to do so. Instead, it establishes the post–condition with a certain probability, 2/3. To handle probabilistic programs, it is thus necessary to consider quantitative pre– and post–conditions and extend the signature of transformer wp to

$$\mathrm{wp}[c] \colon (\mathbb{S} \longrightarrow [0,\, 1]) \longrightarrow (\mathbb{S} \longrightarrow [0,\, 1])\ .$$

A function of type $\mathbb{S} \longrightarrow [0,\, 1]$ is called *expectation* and, accordingly, we call $\mathrm{wp}[c](f)$ the *weakest pre–expectation* of (probabilistic) program c with respect to post–expectation f. For the current exposition, it suffices to consider only qualitative post–expectations of the form $[Q]$, where Q is a predicate over program states and $[Q]$ denotes its characteristic function. The treatment of arbitrary post–expectations is postponed to Section 4.

For an unconditioned program c, the transformer $\mathrm{wp}[c]$ can in fact be defined by induction on the structure of c [38]. For conditioned programs we observe, however, that this compositionality breaks down. To overcome this problem recall Equation (1). In a conditioned program, the probability of any post–expectation $[Q]$ can be computed as the quotient of two other probabilities. Our key observation here is that this pair of probabilities—in contrast to their quotient—do admit an inductive definition, following the program structure. To extend the notion of weakest pre–expectation to conditioned programs, we therefore propose the use of an inductive transformer $\underline{\mathrm{cw}}\mathrm{p}$ that operates over *pairs* of expectations. The transformer works as follows: As input, we provide a pair whose first component is the post–expectation $[Q]$ of interest, and whose second component is the constant post–expectation $\mathbf{1}$. The transformer then outputs a pair of pre–expectations, whose quotient yields the probability of establishing Q. The first component of the pair represents the probability that c passes all observations and establishes Q, whereas the second component represents the probability that c passes all observations (cf. the enumeration above Equation (1)). For instance, for our goldfish–piranha example we obtain

$$\underline{\mathrm{cw}}\mathrm{p}[c_{fish}]\big([f_1 = \mathrm{pir}],\, \mathbf{1}\big) \;=\; \big(\tfrac{1}{2},\, \tfrac{3}{4}\big)\ . \tag{2}$$

Transformer $\underline{\mathrm{cw}}\mathrm{p}$ is defined by induction on the program structure. Following the rules presented in Section 4, we can easily establish the above equation; detailed calculations are provided in Example 4.2.

3 THE PROGRAMMING LANGUAGE

For describing probabilistic programs, we employ the *conditional probabilistic guarded command language* (cpGCL for short), a simple—but powerful—imperative language extended with probabilistic choices and observe statements to endow it with a probabilistic behaviour. Formally, it is given by the grammar:

$$\begin{array}{rcll}
C & ::= & \mathtt{skip} & \text{no–op} \\
 & \mid & \mathtt{abort} & \text{abortion} \\
 & \mid & x := E & \text{assignment} \\
 & \mid & \mathtt{observe}\ (G) & \text{observation} \\
 & \mid & C;\, C & \text{sequential composition} \\
 & \mid & \mathtt{ite}\,(G)\,\{C\}\,\{C\} & \text{conditional branching} \\
 & \mid & \{C\}\ [p]\ \{C\} & \text{probabilistic choice} \\
 & \mid & \mathtt{while}\,(G)\,\{C\} & \text{repetition}
\end{array}$$

Here, x belongs to $\mathcal{V}$, the set of program variables; E is an expression over $\mathcal{V}$ and G denotes, in particular, an expression of Boolean type; p is a probability parameter in $[0,\, 1]$. Except for probabilistic choices and observations, all other language constructs are standard and require no further explanation. $\{c_1\}\ [p]\ \{c_2\}$ represents a *probabilistic choice* between programs c_1 and c_2, where c_1 is executed with probability p and c_2 with probability $1{-}p$. observe G represents a *conditioning* (in the sense of conditional probability) to the distribution of program runs. The effect of such an instruction is to block all program runs violating G and rescale the probability of the remaining runs so that they sum up to one.

Remark (Dynamical probabilities). In the probabilistic choices, instead of parameters $p \in [0,\ 1]$ we could have used arbitrary functions $p\colon \mathbb{S} \longrightarrow [0,\ 1]$ mapping the current program state to a probability as discussed e.g. in [54]. This would not change our semantics fundamentally. However, this would clutter the presentation and we will only need such constructs in the program transformation given in Section 7.1.

Example 3.1. To clarify this, consider the following two programs differing only in the presence of an observation and let us examine the probability that each of them establishes $x = 0$.

$$c_1\colon \quad \{x \coloneqq 0\}\ [1/3]\ \{x \coloneqq 1\};\ \{y \coloneqq 0\}\ [1/4]\ \{y \coloneqq -1\}$$

$$c_2\colon \quad \{x \coloneqq 0\}\ [1/3]\ \{x \coloneqq 1\};\ \{y \coloneqq 0\}\ [1/4]\ \{y \coloneqq -1\};\ \texttt{observe}\ (x{+}y = 0)$$

Program c_1 admits all (four) runs, two of which satisfy $x = 0$; for this program the probability that $x = 0$ is $1/3$. Program c_2—due to the observation requiring $x{+}y = 0$—admits only two runs, only one of them satisfying $x = 0$; for this program the probability that $x = 0$ is $\frac{1/3 \cdot 1/4}{1/3 \cdot 1/4 \,+\, 2/3 \cdot 3/4} = 1/7$. The normalisation factor in the denominator corresponds to the probability of a run that passes the `observe`–statement. △

A cpGCL program without observations such as c_1 will be called *unconditioned.* In the remainder we use syntactic sugar for describing programs like c_1 or c_2. Concretely, we abbreviate a probabilistic choice $\{x \coloneqq E_1\}\ [p]\ \{x \coloneqq E_2\}$ as $x \coloneqq\ E_1\,[p]\,E_2$ and, when possible, we collapse sequences of consecutive assignments like $x_1 \coloneqq E_1; \ldots; x_n \coloneqq E_n$ into a single compound assignment $x_1, \ldots, x_n \coloneqq E_1, \ldots, E_n$. This abbreviation was used before e.g. for describing program c_{fish}.

As for cpGCL semantics, program states correspond to variable valuations. That is, a (program) *state* s is a mapping from variables (in $\mathcal{V}$) to values and we call $\mathbb{S}$ the set of all program states. We assume that the set $\mathcal{V}$ of variables is finite, and that each variable can take countably many values, e.g., the rational numbers. By abuse of notation, we also write $s(E)$ for the value of expression E in state s.

Given the discrete nature of (binary) probabilistic choices, cpGCL induces only discrete distributions. In other words, the distribution of final states obtained by executing a cpGCL program from a given initial state is always discrete. The treatment of continuous distributions is out of the scope of this presentation.

4 WEAKEST PRE–EXPECTATION SEMANTICS

We now recall the weakest pre–expectation semantics of probabilistic programs and extend it to cpGCL to incorporate conditioning. We study some general properties of this semantic extension and present a proof rule to reason about loops.

4.1 Expectation Transformers for Unconditioned Programs

The weakest pre–expectation semantics generalises Dijkstra's original weakest pre–condition semantics to the setting of probabilistic programs. It was first introduced by [38] for fully probabilistic programs[2] with assertions (therein called *tests*) and then extended by [40] to incorporate non–determinism.

To accommodate probabilities, the weakest pre–expectation semantics extends the classic weakest pre–condition semantics twofold. First, instead of being predicates over program states, pre– and

[2] A probabilistic program is said to be *fully probabilistic* if it contains no non–deterministic choice.

post–conditions are now (non–negative) real–valued functions over program states. Second, instead of merely evaluating a (Boolean–valued) post–condition in the final state of a program, we now *measure* the expected value of a (real–valued) post–condition w.r.t. the distribution of final states. Formally, if $f\colon \mathbb{S} \longrightarrow \mathbb{R}^{\geq 0}$ we let

$$\mathsf{wp}[c](f) \;\triangleq\; \lambda s.\, \mathbf{E}_{[\![c]\!](s)}(f)\,,$$

where $[\![c]\!](s)$ denotes the distribution of final states from executing c in initial state s and $\mathbf{E}_{[\![c]\!](s)}(f)$ denotes the expected value of f w.r.t. the distribution of final states $[\![c]\!](s)$.[3] Consider, for instance, the program c_1 of Example 3.1 in Section 3. We have

$$\begin{aligned}\mathsf{wp}[c_1](f)(s) \;=\; \tfrac{1}{12}\, f(s[x,y/0{,}0]) \;+\; \tfrac{1}{4}\, f(s[x,y/0{,}{-1}])\\ +\; \tfrac{1}{6}\, f(s[x,y/1{,}0]) \;+\; \tfrac{1}{2}\, f(s[x,y/1{,}{-1}])\,,\end{aligned}$$

where $s[x_1,\ldots,x_n/v_1,\ldots,v_n]$ represents the state obtained by updating in s the value of variables $x_1,\ldots,x_n$ to $v_1,\ldots,v_n$, respectively.

Observe that, in particular, if $[A]$ denotes the characteristic function of a predicate A over program states, $\mathsf{wp}[c]([A])(s)$ gives the probability of (terminating and) establishing A after executing c from state s. For instance we can determine the probability that c_1 establishes $x + y = 0$ from state s through

$$\mathsf{wp}[c_1]([x{+}y = 0])(s) \;=\; \tfrac{1}{12}\,1 + \tfrac{1}{4}\,0 + \tfrac{1}{6}\,0 + \tfrac{1}{2}\,1 \;=\; \tfrac{7}{12}\,.$$

Moreover, for a deterministic, i.e. non–probabilistic, program c that from state s terminates in state s', $[\![c]\!](s)$ is the Dirac distribution that concentrates all its mass in s' and $\mathsf{wp}[c]([A])(s)$ reduces to $1 \cdot [A](s')$, which gives 1 if $s' \models A$ and 0 otherwise. In this way we recover Dijkstra's classic weakest pre–condition semantics of deterministic programs.

Transformer $\mathsf{wp}[\cdot]$ allows reasoning about total program correctness. To reason about partial program correctness, we define a liberal version of transformer $\mathsf{wp}[\cdot]$, namely $\mathsf{wlp}[\cdot]$. In the same vein as for ordinary sequential programs, $\mathsf{wp}[c]([A])(s)$ gives the probability that program c terminates and establishes event A from state s, while $\mathsf{wlp}[c]([A])(s)$ gives the probability that c terminates and establishes A, *or diverges*.

Formally, the transformer wp operates on unbounded, so–called *expectations* in $\mathbb{E} \triangleq \mathbb{S} \longrightarrow [0,\infty]$, while transformer wlp operates on bounded expectations in $\mathbb{E}_{\leq 1} \triangleq \mathbb{S} \longrightarrow [0,1]$. The reason wlp operates on bounded expectations is that wlp is only meaningful for reasoning about probabilities of events [38] and probabilities are always in range $[0,\ 1]$. Our expectation transformers have thus type

$$\mathsf{wp}[\cdot]\colon \mathbb{E} \longrightarrow \mathbb{E} \qquad \text{and} \qquad \mathsf{wlp}[\cdot]\colon \mathbb{E}_{\leq 1} \longrightarrow \mathbb{E}_{\leq 1}$$

and can be defined by induction on the program structure. To present the definition we require some notation related to expectations.

Notations. In the remainder we use bold fonts for constant expectations, e.g. $\mathbf{1}$ denotes the constant expectation $\lambda s.1$. Given an expression E over program variables, we simply write E for the expectation that in state s returns $s(E)$. Given a Boolean expression G over program variables, we use $[G]$ to denote the $\{0,1\}$–valued expectation that returns 1 if $s \models G$ and 0 otherwise. Finally, given expression E, program variable x and expectation f, we write $f[x/E]$ for the expectation that maps state s to $f(s[x/s(E)])$.

Having fixed the required notation, we present in Figure 3 (second column) the rules defining transformers wp and wlp. Transformer wlp differs from wp only in `abort` and `while`–loops. The

[3] Formally, $[\![c]\!](s)$ denotes a sub–distribution of total mass possibly less than one, where the missing mass represents the probability of c to diverge on input s.

$\boldsymbol{c}$	$\mathbf{wp}[\boldsymbol{c}](\boldsymbol{f})$	$\underline{\mathbf{cw}}\mathbf{p}[\boldsymbol{c}](\boldsymbol{f},\boldsymbol{g})$
`skip`	f	(f,g)
`abort`	$\mathbf{0}$	$(\mathbf{0},\mathbf{1})$
$x \coloneqq E$	$f[x/E]$	$(f,g)[x/E]$
`observe` (G)	—not defined—	$[G]\cdot(f,g)$
$c_1;\ c_2$	$(\mathrm{wp}[c_1]\circ\mathrm{wp}[c_2])(f)$	$(\underline{\mathrm{cw}}\mathrm{p}[c_1]\circ\underline{\mathrm{cw}}\mathrm{p}[c_2])(f,g)$
`ite` $(G)\,\{c_1\}\,\{c_2\}$	$[G]\cdot\mathrm{wp}[c_1](f)+[\neg G]\cdot\mathrm{wp}[c_2](f)$	$[G]\cdot\underline{\mathrm{cw}}\mathrm{p}[c_1](f,g)+[\neg G]\cdot\underline{\mathrm{cw}}\mathrm{p}[c_2](f,g)$
$\{c_1\}\ [p]\ \{c_2\}$	$p\cdot\mathrm{wp}[c_1](f)+(1{-}p)\cdot\mathrm{wp}[c_2](f)$	$p\cdot\underline{\mathrm{cw}}\mathrm{p}[c_1](f,g)+(1{-}p)\cdot\underline{\mathrm{cw}}\mathrm{p}[c_2](f,g)$
`while` $(G)\,\{c'\}$	$\mathit{lfp}_{\preceq}(\mathcal{F})$, where $\mathcal{F}(\hat{f})=[\neg G]\cdot f+[G]\cdot\mathrm{wp}[c'](\hat{f})$	$\mathit{lfp}_{\preceq,\succeq}(\mathcal{G})$, where $\mathcal{G}(\hat{f},\hat{g})=[\neg G]\cdot(f,g)+[G]\cdot\underline{\mathrm{cw}}\mathrm{p}[c'](\hat{f},\hat{g})$
$\boldsymbol{c}$	$\mathbf{wlp}[\boldsymbol{c}](\boldsymbol{f})$	$\underline{\mathbf{cwl}}\mathbf{p}[\boldsymbol{c}](\boldsymbol{f},\boldsymbol{g})$
`abort`	$\mathbf{1}$	$(\mathbf{1},\mathbf{1})$
`while` $(G)\,\{c'\}$	$\mathit{gfp}_{\preceq}(\mathcal{F}\ell)$, where $\mathcal{F}\ell$ is defined as $\mathcal{F}$, but using wlp instead	$\mathit{gfp}_{\preceq,\preceq}(\mathcal{G}\ell)$, where $\mathcal{G}\ell$ is defined as $\mathcal{G}$, but using $\underline{\mathrm{cwl}}\mathrm{p}$ instead

Fig. 3. Inductive definition of transformers w(l)p and $\underline{\mathrm{cw}}(\underline{\mathrm{l}})\mathrm{p}$. Transformer wlp ($\underline{\mathrm{cwl}}\mathrm{p}$) differs from wp ($\underline{\mathrm{cw}}\mathrm{p}$) in `abort` and `while`–loops. Substitution $(f,g)[x/E]$, multiplication $h\cdot(f,g)$ and addition $(f,g)+(f',g')$ are meant to be componentwise. $\mathit{lfp}_{\preceq,\succeq}$ represents the least fixed point over $\mathbb{E}\times\mathbb{E}_{\leq 1}$–transformers, where the first component of expectation pairs adopt the order $\preceq$ and the second component the reverse order $\succeq$; $\mathit{gfp}_{\preceq,\preceq}$ represents the greatest fixed point over $\mathbb{E}_{\leq 1}\times\mathbb{E}_{\leq 1}$–transformers, where both components of expectation pairs adopt the order $\preceq$. See Footnote 5 (p. 13) for a discussion about the origin of the order reversal in the definition of $\underline{\mathrm{cw}}\mathrm{p}[$`while` $(G)\,\{c'\}]$.

action of the transformers on loops is given as the fixed point of an expectation transformer. To guarantee that such fixed points exist, we exploit the fact that the set of expectations $\mathbb{E}$ and $\mathbb{E}_{\leq 1}$ form an ω–complete partial order (ω–cpo): Expectations are ordered pointwise, i.e. $f\ \preceq\ g$ iff $f(s)\leq g(s)$ for every state $s\in\mathbb{S}$. The least upper bound of ω–chains is also defined pointwise, i.e. $(\sup_n f_n)(s)\triangleq\sup_n f_n(s)$ for any ω–chain $f_1\preceq f_2\preceq\ldots$.

Throughout our presentation we follow McIver and Morgan's notation and terminology for the (weakest pre–condition) semantics of probabilistic programs, e.g. we use symbols w(l)p for the transformers and refer to pre– and post–conditions as pre– and post–expectations. In Kozen's original work, transformers wp[·] and wlp[·] are respectively denoted by $\langle\cdot\rangle$ and $[\cdot]$, and represent (dual) modalities of a propositional dynamic logic [38].

Program termination. Since the termination behavior of a program is given by the probability that it establishes true, we can readily use transformer wp[·] to reason about program termination. It suffices to consider the weakest pre–expectation of the program w.r.t. post–expectation $[\mathrm{true}]=\mathbf{1}$. Said otherwise, $\mathrm{wp}[c](\mathbf{1})(s)$ gives the termination probability of program c from state s. In particular, if the program terminates with probability 1, we say that it *terminates almost–surely.*

4.2 Conditional Expectation Transformers

The pre–expectation of an *unconditioned* program c in initial state s is given by the the expected value

$$\mathbf{E}_{[\![c]\!](s)}(f)$$

of the post–expectation with respect to the distribution of final states $[\![c]\!](s)$. If program c includes observations, we consider, instead, the *conditional* expected value to account for their effect. (Recall that the effect of an observation is to condition the distribution of program runs: runs violating the observation are blocked, while the probability of the unblocked runs is normalised.) This conditional expected value can be written as[4]

$$\frac{\mathbf{E}_{[\![c]\!]^{\checkmark}(s)}(f)}{\mathbf{E}_{[\![c]\!]^{\checkmark}(s)}(\mathbf{1})}\ ,$$

where $[\![c]\!]^{\checkmark}(s)$ is the sub–distribution of final states reached by unblocked runs, only. This quotient must be interpreted in the same way as the quotient $\frac{\mathsf{Pr}(A\cap B)}{\mathsf{Pr}(B)}$ encoding the conditional probability $\mathsf{Pr}(A|B)$, the only difference being that here we consider conditional expectations instead of mere conditional probabilities.

To extend the expectation transformer semantics to cpGCL we proceed in two steps. First, we introduce the subsidiary transformer

$$\underline{\mathsf{cw}}\mathsf{p}[\cdot]\colon \mathbb{E}\times\mathbb{E}_{\leq 1}\longrightarrow\mathbb{E}\times\mathbb{E}_{\leq 1}\ ,$$

which will capture the numerator and denominator of the above quotient. Then we define the *conditional weakest pre–expectation* $\mathsf{cwp}[c](f)$ of a cpGCL program c with respect to post–expectation f simply by

$$\mathsf{cwp}[c](f)\ \triangleq\ \frac{\underline{\mathsf{cw}}\mathsf{p}_1[c](f,\mathbf{1})}{\underline{\mathsf{cw}}\mathsf{p}_2[c](f,\mathbf{1})}\ ,$$

where $\underline{\mathsf{cw}}\mathsf{p}_1[c](f,g)$ (resp. $\underline{\mathsf{cw}}\mathsf{p}_2[c](f,g)$) denotes the first (resp. second) component of $\underline{\mathsf{cw}}\mathsf{p}[c](f,g)$. To reason about partial program correctness, transformer $\mathsf{cwp}[\cdot]$ admits a liberal version $\mathsf{cwlp}[\cdot]$, defined analogously, in terms of subsidiary transformer $\underline{\mathsf{cwl}}\mathsf{p}[\cdot]\colon \mathbb{E}_{\leq 1}\times\mathbb{E}_{\leq 1}\longrightarrow\mathbb{E}_{\leq 1}\times\mathbb{E}_{\leq 1}$. We follow this two–step–process because transformer $\mathsf{cw(l)p}[c]$ does not admit an inductive definition over the structure of c, while transformer $\underline{\mathsf{cw(l)}}\mathsf{p}[c]$ does.

Definition 4.1 (Conditional expectation transformers). Given program $c\in$ cpGCL and expectations $f\in\mathbb{E}$ and $g\in\mathbb{E}_{\leq 1}$ we let the *conditional weakest pre–expectation* $\mathsf{cwp}[c](f)$ of c with respect to f and the *conditional weakest liberal pre–expectation* $\mathsf{cwlp}[c](g)$ of c with respect to g be, respectively, defined as

$$\mathsf{cwp}[c](f)\ \triangleq\ \frac{\underline{\mathsf{cw}}\mathsf{p}_1[c](f,\mathbf{1})}{\underline{\mathsf{cw}}\mathsf{p}_2[c](f,\mathbf{1})}\qquad\text{and}\qquad \mathsf{cwlp}[c](g)\ \triangleq\ \frac{\underline{\mathsf{cwl}}\mathsf{p}_1[c](g,\mathbf{1})}{\underline{\mathsf{cwl}}\mathsf{p}_2[c](g,\mathbf{1})}\ ,$$

where transformers

$$\underline{\mathsf{cw}}\mathsf{p}[c]\colon \mathbb{E}\times\mathbb{E}_{\leq 1}\longrightarrow\mathbb{E}\times\mathbb{E}_{\leq 1}\qquad\text{and}\qquad \underline{\mathsf{cwl}}\mathsf{p}[c]\colon \mathbb{E}_{\leq 1}\times\mathbb{E}_{\leq 1}\longrightarrow\mathbb{E}_{\leq 1}\times\mathbb{E}_{\leq 1}$$

are defined by induction on the structure of c, following the rules in Figure 3 (third column), thoroughly discussed below. △

[4] In genereal, the conditional expected value $\mathbf{E}_{\mu}(f\,|B)$ of random variable f with respect to distribution μ is given by $\frac{\mathbf{E}_{\mu_{|B}}(f)}{\mathbf{E}_{\mu_{|B}}(\mathbf{1})}$, where $\mu_{|B}$ represents the restriction of μ to B.

As so defined, transformer $\mathrm{cwp}[\cdot]$ constitutes a simple extension of transformer $\mathrm{wp}[\cdot]$ to conditioned programs: $\mathrm{cwp}[c](f)(s)$ gives the expected value of f with respect to the distribution of final states obtained by executing c in state s, *given that all observations occurring along the runs of c were satisfied.*

In the definition of $\mathrm{cwp}[c](f)$, the scaling factor $\underline{\mathrm{cw}}\mathrm{p}_2[c](f, \mathbf{1})$ gives the probability that program c establishes all its observations, or, in other words, the overall probability of the set of unblocked runs, plus the probability of divergence. If for some initial state s, $\underline{\mathrm{cw}}\mathrm{p}_2[c](f, \mathbf{1})(s) = 0$, program c is said to be *infeasible* from state s, meaning that all its runs are blocked by observations. In this case, $\mathrm{cwp}[c](f)(s)$ is not well–defined. A similar phenomenon occurs for the liberal counterpart $\mathrm{cwlp}[c](g)(s)$.

Both subsidiary transformers $\underline{\mathrm{cw}}\mathrm{p}[\cdot]$ and $\underline{\mathrm{cwl}}\mathrm{p}[\cdot]$ are defined by induction on the program structure, following the rules in Figure 3 (third column). Let us briefly explain these rules. $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{skip}]$ behaves as the identity since `skip` has no effect. $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{abort}]$ maps any pair of post–expectations to the pair of constant pre–expectations $(\mathbf{0}, \mathbf{1})$. Assignments induce a substitution on expectations, i.e. $\underline{\mathrm{cw}}\mathrm{p}[x := E]$ maps (f, g) to pre–expectation $(f[x/E],\ g[x/E])$. $\underline{\mathrm{cw}}\mathrm{p}[c_1; c_2]$ is obtained as the functional composition (denoted $\circ$) of $\underline{\mathrm{cw}}\mathrm{p}[c_1]$ and $\underline{\mathrm{cw}}\mathrm{p}[c_2]$. $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{observe}\,(G)]$ restricts post–expectations to those states that satisfy G; states that do not satisfy G are mapped to 0. $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{ite}\,(G)\ \{c_1\}\ \{c_2\}]$ behaves either as $\underline{\mathrm{cw}}\mathrm{p}[c_1]$ or $\underline{\mathrm{cw}}\mathrm{p}[c_2]$ according to the evaluation of G. $\underline{\mathrm{cw}}\mathrm{p}[\{c_1\}\ [p]\ \{c_2\}]$ is obtained as a convex combination of $\underline{\mathrm{cw}}\mathrm{p}[c_1]$ and $\underline{\mathrm{cw}}\mathrm{p}[c_2]$, weighted according to p. $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\ (G)\ \{c'\}]$ is defined using standard fixed point techniques. The $\underline{\mathrm{cwl}}\mathrm{p}$ transformer follows the same rules as $\underline{\mathrm{cw}}\mathrm{p}$, except for the `abort` and `while` statements. $\underline{\mathrm{cwl}}\mathrm{p}[\mathtt{abort}]$ takes any post–expectation to pre–expectation $(\mathbf{1}, \mathbf{1})$; $\underline{\mathrm{cwl}}\mathrm{p}[\mathtt{while}\,(G)\ \{c\}]$ is defined in terms of a greatest rather than a least fixed point.[5]

Example 4.2. Consider again the goldfish–piranha problem from Section 2 and let us do the detailed calculations to establish Equation (2). Throughout the calculations we use c_{fish}^{i-j} to denote the fragment of program c_{fish} from line i to line j.

$$
\begin{aligned}
&\underline{\mathrm{cw}}\mathrm{p}[c_{\mathit{fish}}]([f_1 = \mathsf{pir}],\ \mathbf{1})\\
&\quad= \underline{\mathrm{cw}}\mathrm{p}\big[c_{\mathit{fish}}^{1-3}\big]\big(\underline{\mathrm{cw}}\mathrm{p}[\mathtt{observe}\,(\mathit{rem} = \mathsf{pir})]([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\\
&\quad= \underline{\mathrm{cw}}\mathrm{p}\big[c_{\mathit{fish}}^{1-2}\big]\Big(\underline{\mathrm{cw}}\mathrm{p}[\mathit{rem} := f_1\ [{}^1\!/_2]\ f_2]\big([\mathit{rem} = \mathsf{pir}] \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\Big)\\
&\quad= \underline{\mathrm{cw}}\mathrm{p}\big[c_{\mathit{fish}}^{1-2}\big]\Big(\tfrac{1}{2} \cdot \underline{\mathrm{cw}}\mathrm{p}[\mathit{rem} := f_1]\big([\mathit{rem} = \mathsf{pir}] \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\\
&\qquad\qquad\qquad + \tfrac{1}{2} \cdot \underline{\mathrm{cw}}\mathrm{p}[\mathit{rem} := f_2]\big([\mathit{rem} = \mathsf{pir}] \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\Big)\\
&\quad= \underline{\mathrm{cw}}\mathrm{p}\big[c_{\mathit{fish}}^{1-1}\big]\Big(\underline{\mathrm{cw}}\mathrm{p}[f_2 := \mathsf{pir}]\Big(\tfrac{1}{2} \cdot \big([f_1 = \mathsf{pir}] \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\\
&\qquad\qquad\qquad\qquad\qquad + \tfrac{1}{2} \cdot \big([f_2 = \mathsf{pir}] \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\Big)\Big)\\
&\quad= \underline{\mathrm{cw}}\mathrm{p}\big[c_{\mathit{fish}}^{1-1}\big]\Big(\tfrac{1}{2} \cdot \underbrace{\big([f_1 = \mathsf{pir}] \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)}_{=\ ([f_1 = \mathsf{pir}],\, [f_1 = \mathsf{pir}])} + \tfrac{1}{2} \cdot \big(\underbrace{[\mathsf{pir} = \mathsf{pir}]}_{=\ \mathbf{1}} \cdot ([f_1 = \mathsf{pir}],\ \mathbf{1})\big)\Big)
\end{aligned}
$$

[5] When defining $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\ \{c'\}](f,\ g)$ as the least fixed point $\mathit{lfp}_{\leq,\geq}(\mathcal{G})$, we reverse the order in the second component of the expectation pairs. This is because, informally, on the first component we require a least fixed point while on the second component we require a greatest fixed point, which we simulate by taking the least fixed point $\mathit{lfp}_{\leq,\geq}(\mathcal{G})$ w.r.t. the "crossed" order $\leq$, $\geq$. The definition of $\underline{\mathrm{cwl}}\mathrm{p}[\mathtt{while}\,(G)\ \{c'\}](f,\ g)$ is more straightforward since in this case we require greatest fixed points on both components of the expectation pairs.

$$
\begin{aligned}
&= \underline{\mathrm{cw}}\mathrm{p}\big[f_1 \coloneqq \mathsf{gold}\ [1/2]\ \mathsf{pir}\big]\Big([f_1 = \mathsf{pir}],\ \tfrac{1}{2}\cdot[f_1 = \mathsf{pir}] + \tfrac{1}{2}\Big)\\
&= \tfrac{1}{2}\cdot\underline{\mathrm{cw}}\mathrm{p}\big[f_1 \coloneqq \mathsf{gold}\big]\Big([f_1 = \mathsf{pir}],\ \tfrac{1}{2}\cdot[f_1 = \mathsf{pir}] + \tfrac{1}{2}\Big)\\
&+ \tfrac{1}{2}\cdot\underline{\mathrm{cw}}\mathrm{p}\big[f_1 \coloneqq \mathsf{pir}\big]\Big([f_1 = \mathsf{pir}],\ \tfrac{1}{2}\cdot[f_1 = \mathsf{pir}] + \tfrac{1}{2}\Big)\\
&= \tfrac{1}{2}\cdot\Big(\underbrace{[\mathsf{gold} = \mathsf{pir}]}_{=\,\mathbf{0}},\ \tfrac{1}{2}\cdot\underbrace{[\mathsf{gold} = \mathsf{pir}]}_{=\,\mathbf{0}} + \tfrac{1}{2}\Big) + \tfrac{1}{2}\cdot\Big(\underbrace{[\mathsf{pir} = \mathsf{pir}]}_{=\,\mathbf{1}},\ \tfrac{1}{2}\cdot\underbrace{[\mathsf{pir} = \mathsf{pir}]}_{=\,\mathbf{1}} + \tfrac{1}{2}\Big)\\
&= \tfrac{1}{2}\cdot\big(\mathbf{0},\ \tfrac{1}{2}\big) + \tfrac{1}{2}\cdot\big(\mathbf{1},\ \mathbf{1}\big) \ = \ \big(\tfrac{1}{2},\ \tfrac{3}{4}\big)\ .
\end{aligned}
$$

From these calculations we conclude that $\mathrm{cwp}[c_{\mathit{fish}}]([f_1 = \mathsf{pir}]) = \frac{1/2}{3/4} = \frac{2}{3}$. In words, the probability that $f_1 = \mathsf{pir}$ after running program c_{fish} (from any initial state) is $2/3$. △

4.3 Conditional Expectation of Loops

As demonstrated in the example above, reasoning about the outcome of loop–free programs consists mostly of syntactic reasoning. Reasoning about the outcome of loops involves, in constrast, fixed points. To circumvent this, we now study a proof rule based on invariants. As a first step to state the proof rule, we need to introduce the *characteristic functional* of a loop, which intuitively captures the effect of $\underline{\mathrm{cw}}\mathrm{p}$ on one iteration.

Definition 4.3. Given program c, guard G and expectations $(f,g) \in \mathbb{E}\times\mathbb{E}_{\le 1}$, let

$$
\begin{array}{lcl}
\mathcal{G}_{f,g}^{\langle G,c\rangle} & : & \mathbb{E}\times\mathbb{E}_{\le 1} \longrightarrow \mathbb{E}\times\mathbb{E}_{\le 1}\\
(\hat{f},\hat{g}) & \mapsto & [\neg G]\cdot(f,g) \ + \ [G]\cdot\underline{\mathrm{cw}}\mathrm{p}[c](\hat{f},\hat{g})
\end{array}
$$

be the *characteristic functional* of loop $\mathtt{while}\,(G)\,\{c\}$ with respect to post–expectations (f,g). For expectations $(f,g) \in \mathbb{E}_{\le 1}\times\mathbb{E}_{\le 1}$, we define the *characteristic liberal functional* $\mathcal{G}\ell_{f,g}^{\langle G,c\rangle} : \mathbb{E}_{\le 1}\times\mathbb{E}_{\le 1} \longrightarrow \mathbb{E}_{\le 1}\times\mathbb{E}_{\le 1}$ analogously, in terms of $\underline{\mathrm{cwl}}\mathrm{p}$. △

Observe that under this definition, the action of transformers $\underline{\mathrm{cw}}(\underline{\mathrm{l}})\mathrm{p}$ on loops can be recast as

$$
\begin{aligned}
\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g) &= \mathit{lfp}_{\le,\ge}\Big(\mathcal{G}_{f,g}^{\langle G,c\rangle}\Big)\\
\underline{\mathrm{cwl}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g) &= \mathit{gfp}_{\le,\le}\Big(\mathcal{G}\ell_{f,g}^{\langle G,c\rangle}\Big)\ .
\end{aligned}
$$

Now we can present our proof rule to determine $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g)$. The rule rests on the presence of an invariant in $\mathbb{E}\times\mathbb{E}_{\le 1}$, parametrised by the set of natural numbers. That is, let $\mathcal{I}_n \in \mathbb{E}\times\mathbb{E}_{\le 1}$ for all $n \ge 0$ and let $\mathcal{G}$ be the characteristic functional of $\mathtt{while}\,(G)\,\{c\}$ with respect to post–expectations $(f,g) \in \mathbb{E}\times\mathbb{E}_{\le 1}$. The rule then reads:

$$
\frac{\mathcal{G}(\mathbf{0},\mathbf{1}) \ = \ \mathcal{I}_0 \qquad \mathcal{G}(\mathcal{I}_n) \ = \ \mathcal{I}_{n+1}}{\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g) \ = \ \lim_{n\to\infty}\mathcal{I}_n}\ [\omega\text{–}\underline{\mathtt{cw}}\mathtt{p\text{-}while}]
$$

If $\mathcal{I}_n$ satisfies the rule premise, we say that it is an *ω–invariant* of the loop with respect to post–expectations (f,g). Intuitively, an ω–invariant $\mathcal{I}_n$ can be interpreted as a sequence of approximations to $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g)$; the larger the n, the more accurate the approximation becomes. In particular, for each n, $\mathcal{I}_n$ coincides with the exact semantics $\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g)$ of the loop in all initial states for which the loop terminates after at most n iterations.

In general, the first component of an ω–invariant is increasing with respect to n, while the second component is decreasing (see the proof of Theorem 4.5). By the Monotone Sequence Theorem[6], their limits always exist which guarantees that term $\lim_{n\to\infty} \mathcal{I}_n$ in the conclusion of the rule is well–defined.

Example 4.4. To illustrate the use of our proof rule consider the following problem: Assume Alice repeatedly flips three fair coins until all three turn tails (symbolised $\textcircled{T}$). What is the probability that she finishes after exactly N trials if in all unsuccessful trials she observed at least one tails?

The problem can be modelled by the program where $\bigvee_{i=1}^{3} b_i$ stands for $b_1 \vee b_2 \vee b_3$.

```
c_tails:  $m := 0;\ b_1, b_2, b_3 := \textcircled{H};$
          while ($\bigvee_{i=1}^{3} b_i = \textcircled{H}$) {
              $b_1, b_2, b_3 := \textcircled{H}\ [1/2]\ \textcircled{T};$
              observe ($\bigvee_{i=1}^{3} b_i = \textcircled{T}$);
              $m := m + 1$
          }
```

The pre–expectation $\mathrm{cwp}[c_{tails}]([m{=}N])$ readily gives the desired probability. The crux for determining this pre–expectation is showing that

$$\mathcal{I}_n \;=\; \left([\neg G] \cdot [m{=}N] + [G] \cdot \sum\nolimits_{i=1}^{n} \tfrac{1}{6} \cdot \left(\tfrac{3}{4}\right)^i \cdot [m{+}i{=}N],\; [\neg G] + [G] \cdot \left(\tfrac{1}{2} + \tfrac{3}{8} \cdot \left(\tfrac{3}{4}\right)^n \right) \right)$$

is an ω–invariant of the loop with respect to post–expectation $([m{=}N], \mathbf{1})$, G being the loop guard. Applying rule [ω–$\underline{\mathrm{c}}\mathrm{wp}$–$\mathtt{while}$], we obtain that for $N \geq 1$,

$$\begin{aligned}
&\underline{\mathrm{c}}\mathrm{wp}[c_{tails}]([m{=}N], \mathbf{1}) \\
&\quad = \underline{\mathrm{c}}\mathrm{wp}[m := 0;\ b_1, b_2, b_3 := \textcircled{H}]\big(\underline{\mathrm{c}}\mathrm{wp}[\mathtt{while}\,(\ldots)\,\{\ldots\}]([m{=}N], \mathbf{1})\big) \\
&\quad = \underline{\mathrm{c}}\mathrm{wp}[m := 0;\ b_1, b_2, b_3 := \textcircled{H}]\big(\lim\nolimits_{n\to\infty} \mathcal{I}_n\big) \\
&\quad = \big(\lim\nolimits_{n\to\infty} \mathcal{I}_n\big)[m, b_1, b_2, b_3 / 0, \textcircled{H}, \textcircled{H}, \textcircled{H}] \\
&\quad = \left(\mathbf{0} \cdot [0{=}N] + \mathbf{1} \cdot \sum\nolimits_{i=1}^{\infty} \tfrac{1}{6} \cdot \left(\tfrac{3}{4}\right)^i \cdot [0{+}i{=}N],\; \mathbf{0} + \mathbf{1} \cdot \left(\tfrac{1}{2} + \tfrac{3}{8} \cdot \lim\nolimits_{n\to\infty} \left(\tfrac{3}{4}\right)^n \right) \right) \\
&\quad = \left(\tfrac{1}{6} \cdot \left(\tfrac{3}{4}\right)^N,\ \tfrac{1}{2} \right) ,
\end{aligned}$$

and we conclude that Alice observes three tails after (exactly) N trials with probability

$$\mathrm{cwp}[c_{tails}]([m{=}N]) \;=\; \frac{1/6 \cdot (3/4)^N}{1/2} \;=\; \frac{\mathbf{1}}{\mathbf{3}} \cdot \left(\frac{3}{4}\right)^N , \quad \forall N \geq 1 \,.$$

As a sanity check, we can use the geometric series to verify that $\sum_{N\geq 1} \frac{1}{3} \left(\frac{3}{4}\right)^N$ sums up to 1. To complete our analysis, we are left to show that $\mathcal{I}_n$ is, indeed, an ω–invariant. To this end, we begin calculating the characteristic functional $\mathcal{G}$ of the loop with respect to post–expectation $([m{=}N], \mathbf{1})$. Throughout the calculations we write *body* for the loop body and G' for the observation condition. We then have

$$\mathcal{G}(\hat{f}, \hat{g}) \;=\; [G] \cdot \underline{\mathrm{c}}\mathrm{wp}[body](\hat{f}, \hat{g}) + [\neg G] \cdot ([m{=}N], \mathbf{1}) \,,$$

[6] If $\langle a_n \rangle_{n\in\mathbb{N}}$ is an increasing (resp. decreasing) sequence in $[0, \infty]$, then $\lim_{n\to\infty} a_n$ exists (possibly being ∞), and coincides with $\sup_n a_n$ (resp. $\inf_n a_n$).

where

$$\begin{aligned}
\underline{\mathrm{cw}}\mathrm{p}[\mathit{body}](\hat{f},\hat{g}) &= \underline{\mathrm{cw}}\mathrm{p}[b_1,b_2,b_3 \coloneqq \textcircled{H}\ [1/2]\ \textcircled{T}]\big(\underline{\mathrm{cw}}\mathrm{p}[\mathtt{observe}\,(G')](\underline{\mathrm{cw}}\mathrm{p}[m \coloneqq m{+}1](\hat{f},\hat{g}))\big) \\
&= \underline{\mathrm{cw}}\mathrm{p}[b_1,b_2,b_3 \coloneqq \textcircled{H}\ [1/2]\ \textcircled{T}]\big(\underline{\mathrm{cw}}\mathrm{p}[\mathtt{observe}\,(G')]((\hat{f},\hat{g})[m/m{+}1])\big) \\
&= \underline{\mathrm{cw}}\mathrm{p}[b_1,b_2,b_3 \coloneqq \textcircled{H}\ [1/2]\ \textcircled{T}]\big([G'] \cdot (\hat{f},\hat{g})[m/m{+}1]\big) \\
&= \sum_{(r_1,r_2,r_3)\neq(\textcircled{H},\textcircled{H},\textcircled{H})} \tfrac{1}{8} \cdot (\hat{f},\hat{g})[m,b_1,b_2,b_3/m{+}1,r_1,r_2,r_3]\ .
\end{aligned}$$

Intuitively, we can justify the last equality above because the only outcome of the coin flips that violates the observation is when the three coins turn heads, and each (non–violating) outcome occurs with probability 1/8. Formally, this step requires a repeated unfolding of $\underline{\mathrm{cw}}\mathrm{p}$ and some straightforward simplifications.

The two requirements $\mathcal{G}(\mathbf{0},\mathbf{1}) = \mathcal{I}_0$ and $\mathcal{G}(\mathcal{I}_n) = \mathcal{I}_{n+1}$ on ω–invariant $\mathcal{I}_n$ are discharged by the following calculations:

$$\begin{aligned}
\mathcal{G}(\mathbf{0},\mathbf{1}) &= [\neg G]\cdot([m{=}N],\mathbf{1}) \ + \ [G] \ \cdot \sum_{(r_1,r_2,r_3)\neq(\textcircled{H},\textcircled{H},\textcircled{H})} \tfrac{1}{8}\cdot(\mathbf{0},\mathbf{1})[m,b_1,b_2,b_3/m{+}1,r_1,r_2,r_3] \\
&= [\neg G]\cdot([m{=}N],\mathbf{1}) \ + \ [G]\cdot\tfrac{1}{8}\cdot(\mathbf{0},\mathbf{7}) \\
&= \big([\neg G]\cdot[m{=}N] + [G]\cdot\mathbf{0},\ [\neg G]\cdot\mathbf{1} + [G]\cdot\tfrac{\mathbf{7}}{\mathbf{8}}\big) \\
&= \mathcal{I}_0
\end{aligned}$$

$$\begin{aligned}
\mathcal{G}(\mathcal{I}_n) &= [\neg G]\cdot([m{=}N],\mathbf{1}) \ + \ [G] \ \cdot \sum_{(r_1,r_2,r_3)\neq(\textcircled{H},\textcircled{H},\textcircled{H})} \tfrac{1}{8}\cdot\mathcal{I}_n[m,b_1,b_2,b_3/m{+}1,r_1,r_2,r_3] \\
&= [\neg G]\cdot([m{=}N],\mathbf{1}) \ + \\
&\qquad [G]\cdot\tfrac{1}{8}\cdot\Big(1\cdot[m{+}1{=}N] + 6\cdot\textstyle\sum_{i=1}^{n}\tfrac{1}{6}\cdot\big(\tfrac{3}{4}\big)^i\cdot[m{+}1{+}i{=}N],\ 1 + 6\cdot\Big(\tfrac{1}{2}+\tfrac{3}{8}\cdot\big(\tfrac{3}{4}\big)^n\Big)\Big) \\
&= \Big([\neg G]\cdot[m{=}N] + [G]\cdot\Big(\underbrace{\tfrac{1}{8}}_{=\,1/6\cdot(3/4)^1}\cdot[m{+}1{=}N] + \textstyle\sum_{i=1}^{n}\underbrace{\tfrac{1}{6}\cdot\tfrac{6}{8}\cdot\big(\tfrac{3}{4}\big)^i}_{=\,1/6\cdot(3/4)^{i+1}}\cdot[m{+}(i{+}1){=}N]\Big), \\
&\qquad [\neg G]\cdot\mathbf{1} + [G]\cdot\Big(\underbrace{\tfrac{1}{8}\cdot\big(1{+}\tfrac{6}{2}\big)}_{=\,1/2} + \underbrace{\tfrac{1}{8}\cdot 6\cdot\tfrac{3}{8}\cdot\big(\tfrac{3}{4}\big)^n}_{=\,3/8\cdot(3/4)^{n+1}}\Big)\Big) \\
&= \Big([\neg G]\cdot[m{=}N] + [G]\cdot\textstyle\sum_{i=1}^{n+1}\tfrac{1}{6}\cdot\big(\tfrac{3}{4}\big)^i\cdot[m{+}i{=}N],\ [\neg G] + [G]\cdot\Big(\tfrac{1}{2}+\tfrac{3}{8}\cdot\big(\tfrac{3}{4}\big)^{n+1}\Big)\Big) \\
&= \mathcal{I}_{n+1}
\end{aligned}$$

In the derivation of $\mathcal{G}(\mathcal{I}_n) = \mathcal{I}_{n+1}$, the second equality holds because out of the seven outcomes of the coin flips different from $(\textcircled{H},\textcircled{H},\textcircled{H})$, one satisfies $\neg G$ and the remaining six satisfy G. △

Rule [ω–$\underline{\mathrm{cw}}\mathrm{p}$-$\mathtt{while}$] can be modified to provide an *approximation*—rather than an exact characterisation—of the behaviour of loops. The new rule relies on the presence of a single—not parametrized—invariant $\mathcal{I} \in \mathbb{E}\times\mathbb{E}_{\leq 1}$ and is stated using the order relation over pairs of expectations "$\leq\times\geq$", which compels an increasing order on the first component of pairs and a decreasing order on the second component, i.e. $(f,g) \leq\times\geq (f',g')$ iff $f \leq f'$ and $g \geq g'$. The rule reads

$$\frac{\mathcal{G}(\mathcal{I}) \leq\times\geq \mathcal{I}}{\underline{\mathrm{cw}}\mathrm{p}[\mathtt{while}\,(G)\,\{c\}](f,g) \leq\times\geq \mathcal{I}}\ [\underline{\mathrm{cw}}\mathrm{p}\text{-}\mathtt{while}]\ ,$$

where $\mathcal{G}$ is the characteristic functional of $\mathtt{while}\,(G)\,\{c\}$ with respect to post–expectations $(f, g) \in \mathbb{E} \times \mathbb{E}_{\leq 1}$. The rule is particularly useful because it allows bounding from above the conditional pre–expectation of programs with loops; in particular, by taking $g = \mathbf{1}$ it allows bounding from above the conditional pre–expectation $\mathsf{cwp}[\mathtt{while}\,(G)\,\{c\}](f)$.

We now establish the formal validity of the introduced rules. Besides being sound, both proof rules [ω–$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$] and [$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$] are complete, in the sense that there always exists an invariant that allows providing the exact semantics of the loop at hand by means of the rules.

Theorem 4.5. *Rules* [ω–$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$] *and* [$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$] *are sound and complete with respect to the* $\underline{\mathsf{cw}}\mathsf{p}$ *semantics of programs in Figure 3.*

Proof. Recall that $\underline{\mathsf{cw}}\mathsf{p}[\mathtt{while}\,(G)\,\{c\}](f,g) = \mathit{lfp}_{\leq,\geq}(\mathcal{G})$ and let us start with rule [ω–$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$]. To establish the soundness of the rule, we exploit first the continuity of $\mathcal{G}$ (which follows from the continuity of $\underline{\mathsf{cw}}\mathsf{p}$ established in Lemma A.2) to conclude that $\mathit{lfp}_{\leq,\geq}(\mathcal{G})$ can be obtained by fixed point iteration from $(\mathbf{0}, \mathbf{1})$. That is, $\mathit{lfp}_{\leq,\geq}(\mathcal{G}) = \sup_n \mathcal{G}^n(\mathbf{0}, \mathbf{1})$, where $\mathcal{G}^n$ denotes the composition of $\mathcal{G}$ with itself n times. By a standard result on ω–cpos, $\mathcal{G}^n(\mathbf{0}, \mathbf{1})$ is monotonic[7] with respect to n and hence $\sup_n \mathcal{G}^n(\mathbf{0}, \mathbf{1}) = \lim_{n\to\infty} \mathcal{G}^n(\mathbf{0}, \mathbf{1})$ by the Monotone Sequence Theorem[6]. To conclude the soundness proof, it is only left to show that $\mathcal{G}^{n+1}(\mathbf{0}, \mathbf{1}) = \mathcal{I}_n$, which can be established from the rule premise, by induction on n. The completeness of the rule readily follows from taking $\mathcal{I}_n = \mathcal{G}^{n+1}(\mathbf{0}, \mathbf{1})$.

Consider now rule [$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$]. The soundness of the rule follows from a straightforward application of Park's Lemma[8], which says that if $\mathcal{G}(\mathcal{I}) \;\preceq\times\succeq\; \mathcal{I}$ then $\mathit{lfp}_{\leq,\geq}(\mathcal{G}) \;\preceq\times\succeq\; \mathcal{I}$. The completeness of the rule follows by taking $\mathcal{I} = \mathit{lfp}_{\leq,\geq}(\mathcal{G})$. □

To conclude our study of the proof rules for loops, we highlight that rule [ω–$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$] can be readily adapted to reason about partial program correctness. It suffices to adjust the initial condition for the iteration of the characteristic functional and consider, instead, its liberal version, i.e.

$$\frac{\mathcal{G}\ell(\mathbf{1},\mathbf{1}) \;=\; \mathcal{I}_0 \qquad \mathcal{G}\ell(\mathcal{I}_n) \;=\; \mathcal{I}_{n+1}}{\underline{\mathsf{cwl}}\mathsf{p}[\mathtt{while}\,(G)\,\{c\}](f,g) \;=\; \lim\limits_{n\to\infty} \mathcal{I}_n}\ [\omega\text{–}\underline{\mathsf{cwl}}\mathsf{p}\text{–}\mathtt{while}]$$

The argument for ensuring the existence of $\lim_{n\to\infty} \mathcal{I}_n$ is analogous to that in rule [ω–$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$], the only difference being that an ω–invariant $\mathcal{I}_n$ satisfying the premises of rule [ω–$\underline{\mathsf{cwl}}\mathsf{p}$–$\mathtt{while}$] is decreasing in *both* its components, instead of increasing in the first and decreasing in the second.

The liberal version of rule [$\underline{\mathsf{cw}}\mathsf{p}$–$\mathtt{while}$] also remains valid, i.e.

$$\frac{\mathcal{I} \;\preceq\times\preceq\; \mathcal{G}\ell(\mathcal{I})}{\mathcal{I} \;\preceq\times\preceq\; \underline{\mathsf{cwl}}\mathsf{p}[\mathtt{while}\,(G)\,\{c\}](f,g)}\ ,$$

but it turns out to be useless as it does not enable bounding the conditional liberal pre–expectations of programs with loops. (Lower–bounds on both the numerator and denominator of a fraction yield no possible bound for the fraction.)

4.4 Basic Properties of Conditional Expectation Transformers

We next investigate some fundamental properties of the expectation transformer semantics of cpGCL. We begin presenting a decomposition result about $\underline{\mathsf{cw}}(\underline{\mathsf{l}})\mathsf{p}$. Concretely, we show that the two components of transformer $\underline{\mathsf{cw}}\mathsf{p}$ (resp. $\underline{\mathsf{cwl}}\mathsf{p}$) are independent. The transformer $\underline{\mathsf{cw}}\mathsf{p}$ can, indeed, be

[7] As underlying ω–cpo we take $\mathbb{E} \times \mathbb{E}_{\leq 1}$ with order $(\preceq, \succeq)$. Therefore, $\mathcal{G}^n(\mathbf{0}, \mathbf{1})$ is increasing in its first component and decreasing in its second component.

[8] If $H\colon \mathcal{D} \to \mathcal{D}$ is a continuous function over an ω–cpo $(\mathcal{D}, \sqsubseteq)$ with bottom element, then $H(d) \sqsubseteq d$ implies $\mathit{lfp}\, H \sqsubseteq d$ for every $d \in \mathcal{D}$ [53].

decoupled as the product wp × wlp (resp. wlp × wlp). To make this claim precise, we need first to extend transformer w(l)p to cpGCL; we define

$$\mathsf{wp}[\mathtt{observe}\,(G)](f) \;\triangleq\; [G]\cdot f \qquad \text{and} \qquad \mathsf{wlp}[\mathtt{observe}\,(G)](f) \;\triangleq\; [G]\cdot f\;.$$

The decomposition result of $\underline{\mathsf{cw}}(\underline{\mathsf{l}})\mathsf{p}$ is then formalised as follows:

Lemma 4.6 (Decoupling of $\underline{\mathsf{cw}}(\underline{\mathsf{l}})\mathsf{p}$). *For $c \in$ cpGCL, $f \in \mathbb{E}$, and $g, g' \in \mathbb{E}_{\leq 1}$,*

$$\underline{\mathsf{cw}}\mathsf{p}[c](f,g) \;=\; \big(\mathsf{wp}[c](f),\, \mathsf{wlp}[c](g)\big) \qquad \textit{and} \qquad \underline{\mathsf{cwl}}\mathsf{p}[c](g,g') \;=\; \big(\mathsf{wlp}[c](g),\, \mathsf{wlp}[c](g')\big)\;.$$

Proof. By induction on the structure of c. See Appendix A.1 for details. □

This decomposition result readily gives an alternative characterisation of transformers cw(l)p, namely

$$\mathsf{cwp}[c](f) \;=\; \frac{\mathsf{wp}[c](f)}{\mathsf{wlp}[c](\mathbf{1})} \qquad \text{and} \qquad \mathsf{cwlp}[c](g) \;=\; \frac{\mathsf{wlp}[c](g)}{\mathsf{wlp}[c](\mathbf{1})}\,, \tag{3}$$

and supports the argument that we employed to extend the expectation transformer semantics to conditioned programs: As an immediate corollary, one can prove that the cwp–semantics is a *conservative extension* of the wp–semantics (to conditioned programs). The same result applies to the liberal version of the semantics.

Theorem 4.7 (Compatibility with the w(l)p–semantics). *For an* unconditioned *program $c \in$ cpGCL, $f \in \mathbb{E}$, and $g \in \mathbb{E}_{\leq 1}$,*

$$\mathsf{cwp}[c](f) \;=\; \mathsf{wp}[c](f) \qquad \textit{and} \qquad \mathsf{cwlp}[c](g) \;=\; \mathsf{wlp}[c](g)\;.$$

Proof. From the alternative characterisation of transformers cw(l)p (Equation (3)) and the fact that for an unconditioned program c, $\mathsf{wlp}[c](\mathbf{1}) = \mathbf{1}$ [38]. □

This means that when applying the cwp–semantics to a probabilistic program without `observe` statements, the first component of the semantics equals the wp–semantics of McIver and Morgan. This holds for all programs, including the possibly diverging ones. In the same vein, the semantics of R2 [42] can be shown to be a conservative extension for certainly terminating probabilistic programs.

Transformer w(l)p enjoys appealing algebraic properties such as monotonicity and (sub-)linearity [38]. These properties remain valid for transformer cw(l)p.

Lemma 4.8 (Basic properties of cw(l)p). *For every $c \in$ cpGCL with at least one feasible execution (from every initial state), post–expectations $f, f' \in \mathbb{E}$, $g, g' \in \mathbb{E}_{\leq 1}$ and non–negative real constants α, α',*

Monotonicity: $f \preceq f' \implies \mathsf{cwp}[c](f) \preceq \mathsf{cwp}[c](f')$

$g \preceq g' \implies \mathsf{cwlp}[c](g) \preceq \mathsf{cwlp}[c](g')$

(Sub–)Linearity: $\mathsf{cwp}[c](\alpha\cdot f + \alpha'\cdot f') \;=\; \alpha\cdot\mathsf{cwp}[c](f) \;+\; \alpha'\cdot\mathsf{cwp}[c](f')$

$\mathsf{cwlp}[c](\alpha\cdot g + \alpha'\cdot g') \;\succeq\; \alpha\cdot\mathsf{cwlp}[c](g) \;+\; \alpha'\cdot\mathsf{cwp}[c](g')$ *for* $\alpha{+}\alpha' \leq 1$

Duality: $\mathsf{cwlp}[c](g) \;=\; \mathbf{1} - \mathsf{cwp}[c](\mathbf{1}{-}g)$

Preserv. of $\mathbf{0}$, $\mathbf{1}$: $\mathsf{cwp}[c](\mathbf{0}) = \mathbf{0}$ *and* $\mathsf{cwlp}[c](\mathbf{1}) = \mathbf{1}$.

Proof. In view of Equation (3), monotonicity and (sub–)linearity are inherited from transformer w(l)p, duality follows from the more general property $\mathsf{wlp}[c](g) + \mathsf{wp}[c](g') = \mathsf{wlp}[c](g + g')$ (see Appendix A.3), by taking $g' = \mathbf{1}{-}g$, preservation of $\mathbf{0}$ is also inherited from wp and preservation of $\mathbf{1}$ is immediate. □

Let us briefly discuss these properties. Monotonicity is an inherent property of the transformers; it guarantees e.g. that the probability that a program establishes some property Q is at most the probability that it establishes property, say, Q' whenever Q implies Q'. Linearity is relevant because it allows for modular reasoning about the specification of programs. Duality says that we can reason about partial program correctness using transformer cwp. It also simplifies our proof effort since most properties about cwlp can be established by a direct dualisation argument; for instance the preservation of **1** (by cwlp) can be derived by dualisation from the preservation of **0** (by cwp). Preservation of **0** says that the probability that a program establishes false is zero; it is the probabilistic counterpart of the so–called *law of excluded miracle* [19]. Finally, preservation of **1** says that almost–surely a program either terminates (i.e. establishes true) or diverges.

These properties are shared by transformers w(l)p and cw(l)p. There are, however, two properties that do not carry over from w(l)p to cw(l)p, namely continuity and the ability to establish a contextual equivalence. *Continuity* is an important semantic feature because, loosely speaking, it guarantees that the behaviour of a loop coincides with the limit behaviour of its finite approximations. Formally, we define the *n-unrolling* $\texttt{while}_n\,(G)\,\{c\}$ of a loop by

$$\begin{aligned}\texttt{while}_0\,(G)\,\{c\} &\triangleq \texttt{abort}\\ \texttt{while}_{n+1}\,(G)\,\{c\} &\triangleq \texttt{ite}\,(G)\,\{c;\, \texttt{while}_n\,(G)\,\{c\}\}\,\{\texttt{skip}\}\end{aligned}$$

and continuity of e.g. cwp would ensure that

$$\text{cwp}[\texttt{while}\,(G)\,\{c\}](f) \;=\; \sup_n\; \text{cwp}[\texttt{while}_n\,(G)\,\{c\}](f)\;.$$

For an infeasible loop, however, this equality breaks down. To illustrate this phenomenon, consider the loop

$$\texttt{while}\,(x{=}1)\,\{x \coloneqq 1\;[1/2]\;0;\, \texttt{observe}\,(x{=}1)\}$$

and let *body* denote its body. After some calculations we obtain

$$\begin{aligned}\underline{\text{cw}}\text{p}[\texttt{while}\,(x{=}1)\,\{\mathit{body}\}](f,\mathbf{1}) &= ([x{\neq}1]\cdot f,\; [x{\neq}1])\\ \underline{\text{cw}}\text{p}[\texttt{while}_n\,(x{=}1)\,\{\mathit{body}\}](f,\mathbf{1}) &= \left([x{\neq}1]\cdot f,\; \tfrac{1}{2^n}\cdot[x{=}1] + [x{\neq}1]\right) \quad \text{for } n \geq 1\;.\end{aligned}$$

For any initial state s where $x = 1$, $\text{cwp}[\texttt{while}\,(x{=}1)\,\{\mathit{body}\}](f)(s)$ is not well-defined because the loop is infeasible from s, while $\sup_n \text{cwp}[\texttt{while}_n\,(x{=}1)\,\{\mathit{body}\}](f)(s)$ is well-defined and gives 0.

The second property that does not carry over from w(l)p to cw(l)p is the ability to establish a *contextual equivalence*. For unconditioned programs, the notion of semantic equivalence induced by wp allows for a safe interchangeability of equivalent programs. Formally, if $\text{wp}[c_1](f) = \text{wp}[c_2](f)$ for every post–expectation f, then $\text{wp}[\mathfrak{C}[c_1]](f) = \text{wp}[\mathfrak{C}[c_2]](f)$ for every (unconditioned) program context $\mathfrak{C}[\cdot]$. Intuitively, this holds because the action of wp on a compound program is completely determined by its action on the sub–programs. In the general case, this compositionality breaks down for transformer cwp, though. As a consequence, the transformer does not induce a contextual equivalence for conditioned programs. To see this, consider the programs

$$\begin{aligned}c_1:&\quad x \coloneqq 1\\ c_2:&\quad x \coloneqq 1\;[1/2]\;0;\; \texttt{observe}\,(x{=}1)\end{aligned}$$

Both programs are cwp–equivalent since $\text{cwp}[c_1](f) = \text{cwp}[c_2](f) = f[x/1]$. However, if we put them into context $\mathfrak{C}[c] = \{c\}\;[1/2]\;\{x \coloneqq 2\}$, both programs are easily distinguished e.g. by post–condition $x = 1$, since $\text{cwp}[\mathfrak{C}[c_1]]([x{=}1]) = 1/2$, while $\text{cwp}[\mathfrak{C}[c_2]]([x{=}1]) = 1/3$. Despite this limitation, we believe that cwp–equivalence remains a useful notion as it guarantees that cwp–equivalent programs cannot be distinguished by events: Two cwp–equivalent programs assign the exact same probability to any event (or Boolean post–condition).

A better behaved transformer is $\underline{\text{cw}}\text{p}$. Its definition is completely compositional—the first principle of the denotational semantics—and it induces a contextual equivalence between conditioned programs. It is able to distinguish, for instance, the two programs above since $\underline{\text{cw}}\text{p}[c_1](f, \mathbf{1}) = (f[x/1], \mathbf{1})$, while $\underline{\text{cw}}\text{p}[c_2](f, \mathbf{1}) = ({}^1\!/_2 \cdot f[x/1], {}^1\!/_2)$.

We conclude this section discussing some alternative approaches for providing an expectation transformer semantics to conditioned programs. As mentioned before, a notable consequence of Lemma 4.6 is that we can rewrite our transformers cw(l)p as in Equation (3). There, both $\text{cwp}[c](f)$ and $\text{cwlp}[c](g)$ are normalised with respect to $\text{wlp}[c](\mathbf{1})$, the probability that c either diverges or passes all observations. An alternative approach is to normalise using wp instead of wlp, yielding the pair of transformers

$$f \mapsto \frac{\text{wp}[c](f)}{\text{wp}[c](\mathbf{1})} \qquad \text{and} \qquad g \mapsto \frac{\text{wlp}[c](g)}{\text{wp}[c](\mathbf{1})} .$$

For the transformer on the right, the denominator $\text{wp}[c](\mathbf{1})(s)$ may be smaller than the numerator $\text{wlp}[c](g)(s)$ for some state $s \in \mathbb{S}$. This leads to probabilities exceeding one. The transformer on the left normalises with respect to the terminating executions (that pass all observations). This is a fully reasonable choice for certainly terminating programs, i.e., programs that have no diverging runs, or almost-surely terminating programs, i.e., programs whose divergent behaviours have probability mass zero. This is the approach taken in the formal semantics of the probabilistic programming language R2 [28, 42] which aims at applications like image computations that typically are certainly terminating programs[9] A noteworthy consequence of adopting this transformer is that $\texttt{observe}\,(G)$ is equivalent to $\texttt{while}\,(\neg G)\,\{\texttt{skip}\}$ [28]. This is not the case when normalizing w.r.t. all (including the possibly diverging) program behaviours as is discussed in detail in Section 7.

Example 4.9. The pair of transformers discussed above together with cwp and cwlp yield, overall, four different semantic approaches for conditioned programs. Let us briefly compare these alternatives by means of a concrete program

$$c\colon \quad \{\texttt{abort}\}\ [{}^1\!/_2]\ \{x, y \coloneqq \textcircled{H}\ [{}^1\!/_2]\ \textcircled{T};\ \texttt{observe}\,(x = \textcircled{H} \vee y = \textcircled{H})\} .$$

Program c tosses a fair coin and according to the outcome either diverges or tosses a fair coin twice, and observes at least once heads. If we measure the probability that the outcome of the last coin toss was heads according to each of the four transformers, we obtain:

$$\frac{\text{wp}[c]([y = \textcircled{H}])}{\text{wlp}[c](\mathbf{1})} = \frac{2}{7} \qquad \frac{\text{wlp}[c]([y = \textcircled{H}])}{\text{wlp}[c](\mathbf{1})} = \frac{6}{7} \qquad \frac{\text{wp}[c]([y = \textcircled{H}])}{\text{wp}[c](\mathbf{1})} = \frac{2}{3} \qquad \frac{\text{wlp}[c]([y = \textcircled{H}])}{\text{wp}[c](\mathbf{1})} = 2 .$$

As mentioned before, the last transformer is not meaningful as it results in a value—in this case: a "probability"—exceeding one. Our cwp transformer (the leftmost above) yields that the probability that $y = \textcircled{H}$ after executing c while passing statement $\texttt{observe}\,(x = \textcircled{H} \vee y = \textcircled{H})$ is ${}^2\!/_7$. Intuitively, this can be seen as follows. The right (non–diverging) branch admits four runs, three of which are valid. For the sake of argument, assume the left branch has four runs as well, all diverging. Their total probability mass is ${}^1\!/_2$, the probability to abort. Seven out of the total eight runs are valid. As in total two runs establish the condition $[y = \textcircled{H}]$, we obtain $\frac{2}{7}$. As shown before, this is a conservative and simple extension of the wp–semantics to conditioned programs. For the R2–semantics (the third transformer above), this desired result does not hold. Intuitively, the R2-approach ignores the diverging branch. Then there are three (out of four) feasible runs, two of which establish $[y = \textcircled{H}]$. This yields $\frac{2}{3}$. Note that for almost–surely terminating programs, the R2 approach and our semantics coincide. △

[9] For instance, written as a double–nested for-loop iterating over both dimensions of the image.

5 OPERATIONAL SEMANTICS

As a next step, we investigate the relationship between the expectation transformer semantics of Section 4 and an operational interpretation of cpGCL programs. Inspired by [25], a small–step operational semantics for cpGCL is defined where programs are interpreted as Markov chains. We then prove that conditional weakest pre–expectations correspond to conditional expected rewards in these Markov chains. We first present the intuition in an informal manner and then define the necessary notion (such as paths and expected rewards) on Markov chains. This is followed by the detailed operational semantics and the correspondence result.

5.1 Informal Account

To each program and initial state we associate a Markov chain whose evolution fully characterises the possible program executions. Intuitively, a Markov chain is a transition system where the successor of a state is chosen according to a probability distribution, and this distribution depends only on the current state (memoryless property). In our case, the states of the Markov chain represent different points of the program execution; they are of the form $\langle c,\, s \rangle$, where c represents the program fragment left to execute and s the program state at that point. The Markov chain contains, additionally, two distinguished states $\langle \lightning \rangle$ and $\langle \mathit{sink} \rangle$. The state $\langle \lightning \rangle$ models the violation of an observation and $\langle \mathit{sink} \rangle$ models the program termination, either successful or due to a violated observation. Each successful (i.e. unblocked) terminating run of the program corresponds to a path (along states) of the Markov chain, and the probability of the run corresponds to the probability of the path in the Markov chain.

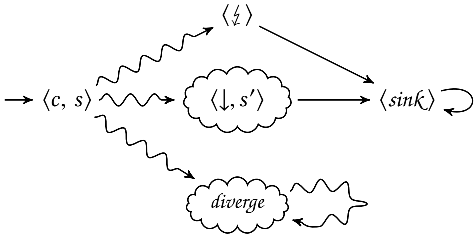


Fig. 4. Schema of the operational Markov chain of a cpGCL program.

For a program $c \in$ cpGCL and an initial state $s \in \mathbb{S}$, the general structure of the Markov chain is depicted in Figure 4. A program run either

a) terminates successfully in a terminal state of the form $\langle \downarrow,\, s' \rangle$ for some $s' \in \mathbb{S}$ (symbol "$\downarrow$" indicates that there is nothing left to execute, and s' is the final state of the run), or
b) terminates due to a false observation, transitioning to state $\langle \lightning \rangle$, or
c) diverges while passing all observations; modelled by an infinite path never reaching $\langle \mathit{sink} \rangle$.

In Figure 4, squiggly arrows indicate reaching certain states via possibly multiple paths and intermediate states; clouds indicate sets of states. Note that the sets of paths that eventually reach $\langle \lightning \rangle$, eventually reach a terminal state $\langle \downarrow, \cdot \rangle$, or diverge, are pairwise disjoint.

To be able to relate this operational program model to our expectation transformer semantics, we must incorporate post–expectations in the model. We do so by adding (real–valued) *rewards* to the states of the Markov chain. All states will have reward zero, except for the (terminal) states of the form $\langle \downarrow,\, s' \rangle$, whose reward will be the value of the post–expectation in s'. The program outcome (with respect to a given post–expectation or reward over final states) then corresponds to the so–called *conditional expected reward* to reach state $\langle \mathit{sink} \rangle$, conditioned on the fact that $\langle \lightning \rangle$ is avoided. Our correspondence result will state that this agrees with the semantics as defined by the expectation transformer cwp.

5.2 Preliminaries on Markov Chains

We next recall some preliminaries about Markov chains necessary to formalize the outlined operational model. Let $\mathcal{D}(A)$ denote the set of probability distributions $\mu\colon A \longrightarrow [0,1]$ over a countable set A, where $\sum_{a \in A} \mu(a) = 1$.

Definition 5.1 (Markov chain). A *Markov chain* is a tuple $\mathfrak{M} = (\Sigma, \sigma_I, \mathcal{P})$ with a countable set of states Σ, an initial state $\sigma_I \in \Sigma$, and a transition probability function $\mathcal{P}\colon \Sigma \longrightarrow \mathcal{D}(\Sigma)$. $\triangle$

A *path* of the Markov chain $\mathfrak{M}$ is an infinite sequence of states $\pi = \sigma_0\sigma_1\sigma_2 \ldots$ where $\sigma_i \in \Sigma$, $\sigma_0 = \sigma_I$ and $\mathcal{P}(\sigma_i)(\sigma_{i+1}) > 0$. The transition probability function $\mathcal{P}$ induces a probability measure $\mathsf{Pr}^{\mathfrak{M}}$ over the set of paths of $\mathfrak{M}$, denoted by $\mathsf{Paths}^{\mathfrak{M}}$. The formal definition of $\mathsf{Pr}^{\mathfrak{M}}$ rests on the σ–algebra induced by the cylinder sets spanned by finite prefix paths [5, Ch. 10.1]. If the Markov chain $\mathfrak{M}$ is clear from the context we write Pr for $\mathsf{Pr}^{\mathfrak{M}}$.

In our setting, we are interested in reachability properties: Given a set of *target* states $T \subseteq \Sigma$, let

$$\Diamond\, T \;\triangleq\; \left\{\pi = \sigma_0\sigma_1 \ldots \in \mathsf{Paths}^{\mathfrak{M}} \mid \exists n.\sigma_n \in T \text{ and } \sigma_i \notin T \text{ for all } 0 \le i < n\right\}$$

be the set of all paths that visit a target state in T. It follows by simple arguments that $\Diamond\, T$ is measurable. Let $\mathsf{Pr}^{\mathfrak{M}}(\Diamond\, T)$ denote the probability of eventually reaching a state in T from the initial state σ_I in Markov chain $\mathfrak{M}$. Analogously, for the set of *undesired* states $U \subseteq \Sigma$, let

$$\neg\Diamond\, U \;\triangleq\; \left\{\pi = \sigma_0\sigma_1 \ldots \in \mathsf{Paths}^{\mathfrak{M}} \;\middle|\; \sigma_i \notin U \text{ for all } i \ge 0\right\} \;=\; \mathsf{Paths}^{\mathfrak{M}} \setminus \Diamond\, U$$

be the set of paths that never visit a state in U; $\mathsf{Pr}^{\mathfrak{M}}(\neg\Diamond\, U)$ is the probability of never visiting a state in U. In our operational program interpretation, $\langle sink \rangle$ plays the role of the (single) target state, while $\langle \lightning \rangle$ represents the (single) undesired state. For the sake of succinctness, we write $\Diamond\, sink$ and $\neg\,\Diamond \lightning$ for $\Diamond\, \{\langle sink \rangle\}$ and $\neg\, \Diamond\, \{\langle \lightning \rangle\}$, respectively.

In order to be able to reason about expectations in states—after all, we are interested in capturing weakest pre-expectations—we equip Markov chains with a *reward function* $r\colon \Sigma \longrightarrow \mathbb{R}_{\ge 0}$ that associates non–negative rewards to the Markov chain states. Note that a Markov chain together with a reward function is also referred to as *Markov reward chain.* For a finite prefix $\hat{\pi} = s_0 \ldots s_n$ of a path, let $r(\hat{\pi}) \triangleq \sum_{i=0}^{n-1} r(s_i)$ be the *cumulative reward* of $\hat{\pi}$. Here, it is assumed that a reward is "earned" upon leaving a state. The reward of the last state s_n of $\hat{\pi}$ thus is not taken into account. Let $rv(\Diamond\, T)$ be the random variable that assigns to each path π in $\mathfrak{M}$ the reward $r(\hat{\pi})$ of the shortest prefix $\hat{\pi}$ of π such that the last state in $\hat{\pi}$ belongs to T. We have $rv(\Diamond\, T)(\pi) = \infty$ whenever $\pi \notin \Diamond\, T$. Let $\mathsf{ER}^{\mathfrak{M}}\,(\Diamond\, T)$ be the expectation of the random variable $rv(\Diamond\, T)$ for the Markov chain $\mathfrak{M}$ when starting in its initial state. If $\mathsf{Pr}(\Diamond\, T) < 1$, then this expectation is zero. $\mathsf{ER}^{\mathfrak{M}}\,(\Diamond\, T) \in \mathbb{R}_{\ge 0}^{\infty}$ thus represents the *expected reward* upon reaching (a target state in) T in $\mathfrak{M}$ from its starting state. From the proof of measurability of the set $\Diamond\, T$ [5, Ch. 10.1], we have

$$\mathsf{ER}^{\mathfrak{M}}\,(\Diamond\, T) \;=\; \sum_{\pi \in \Diamond T} \mathsf{Pr}^{\mathfrak{M}}(\hat{\pi}) \cdot r(\hat{\pi})$$

where $\hat{\pi} = \sigma_0 \ldots \sigma_n$ is the shortest prefix of π such that $\sigma_n \in T$ and $\mathsf{Pr}^{\mathfrak{M}}(\hat{\pi})$ is the probability of the finite path $\hat{\pi}$ defined as $\mathcal{P}(\sigma_0)(\sigma_1) \cdot \ldots \cdot \mathcal{P}(\sigma_{n-1})(\sigma_n)$. In a similar way, let $rv(\Diamond\, T \cap \neg\Diamond\, U)$ be the random variable that is defined as $rv(\Diamond\, T)$ with the additional constraint that on the shortest prefix until reaching T no state in U is visited. Then, $\mathsf{ER}^{\mathfrak{M}}\,(\Diamond\, T \cap \neg\Diamond\, U)$ is the expected value of this random variable.

To understand the role of rewards in the operational semantics, consider an unconditioned program. Let us discuss the expected reward $\mathsf{ER}\,(\Diamond\, sink)$ upon reaching $\langle sink \rangle$. Assume that the program almost–surely terminates, that is, $\mathsf{Pr}(\Diamond\, sink)$ equals one. All terminating runs of the program are represented by paths reaching $\langle sink \rangle$ (see Figure 4). The cumulative reward of such paths is the value of the post–expectation in the final state of the runs (recall that terminal states are the only ones to collect positive reward, conveyed—precisely—by the post–expectation). Then, $\mathsf{ER}\,(\Diamond\, \mathit{sink})$ gives the average of the post–expectation over the set of final states, weighted according to the probability of reaching each of these final states. As shown in [25], this is exactly the effect of transformer wp on unconditioned programs.

To extend this result to programs with `observe` statements we consider *conditional* expected rewards. Let $\mathsf{CER}^{\mathfrak{M}}\,(\Diamond\, T \mid \neg\Diamond\, U)$ be the expectation of random variable $rv(\Diamond\, T)$ with respect to the conditional probability measure

$$\mathsf{Pr}^{\mathfrak{M}}(\Diamond\, T \mid \neg\Diamond\, U) = \frac{\mathsf{Pr}^{\mathfrak{M}}(\Diamond\, T \cap \neg\Diamond\, U)}{\mathsf{Pr}^{\mathfrak{M}}(\neg\Diamond\, U)}\ .$$

Intuitively speaking, $\mathsf{CER}^{\mathfrak{M}}\,(\Diamond\, T \mid \neg\Diamond\, U)$ is the conditional expected reward to reach T while avoiding U.

Definition 5.2 (Conditional expected reward). Given a Markov chain $\mathfrak{M} = (\Sigma, \sigma_I, \mathcal{P})$, a reward function $r\colon \Sigma \longrightarrow \mathbb{R}_{\geq 0}$, and sets of states $T, U \subseteq \Sigma$, the *conditional expected reward* to reach T while avoiding U is defined as

$$\mathsf{CER}^{\mathfrak{M}}\,(\Diamond\, T \mid \neg\Diamond\, U) \;\triangleq\; \frac{\mathsf{ER}^{\mathfrak{M}}\,(\Diamond\, T \cap \neg\Diamond\, U)}{\mathsf{Pr}^{\mathfrak{M}}(\neg\Diamond\, U)}\ . \qquad \triangle$$

Both ordinary and conditional expected rewards admit a liberal version to account for the cases where the set of target states are not reached with probability one. For a reward function $r\colon \mathbb{S} \longrightarrow [0,\ 1]$, they are defined as

$$\mathsf{LER}^{\mathfrak{M}}\,(\Diamond\, T) \;\triangleq\; \mathsf{ER}^{\mathfrak{M}}\,(\Diamond\, T) + \mathsf{Pr}^{\mathfrak{M}}(\neg\Diamond\, T)\,,$$

$$\mathsf{CLER}^{\mathfrak{M}}\,(\Diamond\, T \mid \neg\Diamond\, U) \;\triangleq\; \frac{\mathsf{LER}^{\mathcal{R}}\,(\Diamond\, T \cap \neg\Diamond\, U)}{\mathsf{Pr}^{\mathfrak{M}}(\neg\Diamond\, U)}\ .$$

These liberal variants will be useful for reasoning about programs that do not terminate with probability one.

5.3 Operational Markov Reward Chain of Programs

We have all the necessary ingredients to introduce our operational model of programs in detail. Formally, this operational model is given in terms of what we call *operational Markov reward chains* (OMRC), as sketched in Figure 4.

Definition 5.3 (Operational Markov reward chain). The *operational Markov reward chain* $\mathfrak{R}_s^f[\![c]\!]$ of program $c \in \mathsf{cpGCL}$ in state $s \in \mathbb{S}$ with respect to post–expectation $f \in \mathbb{E}$ is defined as follows:

- the set of states of $\mathfrak{R}_s^f[\![c]\!]$ contains distinguished states $\langle \lightning \rangle$ (violation of observation) and $\langle \mathit{sink} \rangle$ (program termination), intermediate computation states of the form $\langle c',\ s' \rangle$ where c' is a subprogram of c and $s' \in \mathbb{S}$, and terminal states of the form $\langle \downarrow,\ s' \rangle$ for $s' \in \mathbb{S}$;
- the initial state of $\mathfrak{R}_s^f[\![c]\!]$ is $\langle c, s \rangle$;
- the transition probability function of $\mathfrak{R}_s^f[\![c]\!]$ is defined by the rules in Figure 5, and
- the reward function r of of $\mathfrak{R}_s^f[\![c]\!]$ is defined as $r(\sigma) \triangleq f(s')$ if $\sigma = \langle \downarrow,\ s' \rangle$ for some $s' \in \mathbb{S}$ and $r(\sigma) \triangleq 0$ otherwise.

The *conditional* (*liberal*) *expected outcome* of program c with respect to post–expectation $f \in \mathbb{E}$ is given by the conditional (liberal) expected reward

$$\mathsf{C(L)ER}^{\mathfrak{R}_s^f[\![c]\!]}\,(\Diamond\, \mathit{sink} \mid \neg\, \Diamond \lightning)$$

$$\textbf{[skip]}\ \frac{}{\langle \texttt{skip},\ s\rangle \longrightarrow \langle {\downarrow},\ s\rangle} \qquad \textbf{[assign]}\ \frac{}{\langle x := E,\ s\rangle \longrightarrow \langle {\downarrow},\ s[x/s(E)]\rangle} \qquad \textbf{[abort]}\ \frac{}{\langle \texttt{abort},\ s\rangle \longrightarrow \langle \texttt{abort},\ s\rangle}$$

$$\textbf{[observe-t]}\ \frac{s \models G}{\langle \texttt{observe}\,(G),\ s\rangle \longrightarrow \langle {\downarrow},\ s\rangle} \qquad \textbf{[observe-f]}\ \frac{s \not\models G}{\langle \texttt{observe}\,(G),\ s\rangle \longrightarrow \langle \lightning \rangle}$$

$$\textbf{[terminal]}\ \frac{}{\langle {\downarrow},\ s\rangle \longrightarrow \langle \mathit{sink} \rangle} \qquad \textbf{[undesired]}\ \frac{}{\langle \lightning \rangle \longrightarrow \langle \mathit{sink} \rangle} \qquad \textbf{[sink]}\ \frac{}{\langle \mathit{sink} \rangle \longrightarrow \langle \mathit{sink} \rangle}$$

$$\textbf{[if-t]}\ \frac{s \models G}{\langle \texttt{ite}\,(G)\,\{c_1\}\,\{c_2\},\ s\rangle \longrightarrow \langle c_1,\ s\rangle} \qquad \textbf{[if-f]}\ \frac{s \not\models G}{\langle \texttt{ite}\,(G)\,\{c_1\}\,\{c_2\},\ s\rangle \longrightarrow \langle c_2,\ s\rangle}$$

$$\textbf{[prob]}\ \frac{}{\langle \{c_1\}\ [p]\ \{c_2\},\ s\rangle \longrightarrow p\,\langle c_1,\ s\rangle + (1{-}p)\,\langle c_2,\ s\rangle}$$

$$\textbf{[while-t]}\ \frac{s \models G}{\langle \texttt{while}\,(G)\,\{c\},\ s\rangle \longrightarrow \langle c; \texttt{while}\,(G)\,\{c\},\ s\rangle} \qquad \textbf{[while-f]}\ \frac{s \not\models G}{\langle \texttt{while}\,(G)\,\{c\},\ s\rangle \longrightarrow \langle {\downarrow},\ s\rangle}$$

$$\textbf{[seq-}{\downarrow}\textbf{]}\ \frac{\langle c_1,\ s\rangle \longrightarrow \langle {\downarrow}, s'\rangle}{\langle c_1; c_2,\ s\rangle \longrightarrow \langle c_2,\ s'\rangle} \qquad \textbf{[seq-}\lightning\textbf{]}\ \frac{\langle c_1,\ s\rangle \longrightarrow \langle \lightning \rangle}{\langle c_1; c_2,\ s\rangle \longrightarrow \langle \lightning \rangle} \qquad \textbf{[seq]}\ \frac{\langle c_1,\ s\rangle \longrightarrow \sum_i p_i\,\langle c_1^i,\ s_i\rangle}{\langle c_1; c_2,\ s\rangle \longrightarrow \sum_i p_i\,\langle c_1^i; c_2,\ s_i\rangle}$$

Fig. 5. Rules for constructing the OMRC of programs. $\sigma \longrightarrow \mu$ denotes that the OMRC evolves from state σ to a distribution μ over states. $\sum_i p_i\, \sigma_i$ denotes the distribution that assigns probability p_i to state σ_i. In particular, σ is a shorthand for the Dirac distribution $1\,\sigma$.

of reaching $\langle \mathit{sink} \rangle$ from initial state $\langle c, s\rangle$, conditioned on not visiting $\langle \lightning \rangle$. For the liberal version, we assume that $f \in \mathbb{E}_{\leq 1}$. △

Except for the transition function, all elements of the OMRCs were already sketched in Section 5.1. The set of rules defining the transition function are rather straightforward. Let us briefly discuss them. `skip` terminates successfully (recall that "$\downarrow$" indicates a terminating state). $x := E$ updates the program state and terminates successfully. `abort` self–loops, i.e. diverges. `observe` (G) either terminates successfully or evolves into $\langle \lightning \rangle$, depending on the valuation of the guard. Terminal states and $\langle \lightning \rangle$ evolve into $\langle \mathit{sink} \rangle$, which once reached, is never left. `ite` $(G)\,\{c_1\}\,\{c_2\}$ transitions to a state containing either of its branches, according to the valuation of the guard. $\{c_1\}\ [p]\ \{c_2\}$ transitions with probability p to a state executing c_1 and with probability $1{-}p$ to a state executing c_2. `while` $(G)\,\{c\}$ either terminates successfully or unfold its body once, depending on the valuation of the guard. Finally, for the sequential composition $c_1; c_2$ we (recursively) apply one transition step in c_1 and "append" the reachable states with c_2. If c_1 terminates successfully in one step, we continue the execution with c_2 and if c_1 transitions to $\langle \lightning \rangle$ in one step (i.e. c_1 is an observation that is violated in the state at hand), we remain in state $\langle \lightning \rangle$.

Example 5.4. To illustrate the application of these rules, we now sketch the full-fledged OMRC of program c_{fish} from Section 2 (see Figure 1). The Markov chain is depicted in Figure 6. Observe that in constrast to the simplified model given in Section 2 (Figure 2), this OMRC

(1) tags states with the program fragment left to execute instead of with the current program line. This is consistent with standard small–step semantics of imperative programs and arises because it is convenient that states contain all the necessary information to determine their immediate successors (memoryless property of Markov chains);
(2) collects the undesired state $\langle \lightning \rangle$ and all terminal states $\langle {\downarrow}, \cdot\rangle$ into the absorbing state $\langle \mathit{sink} \rangle$. This is just for convenience so that when defining the program outcome, our set of target states is just the singleton $\{\langle \mathit{sink} \rangle\}$;

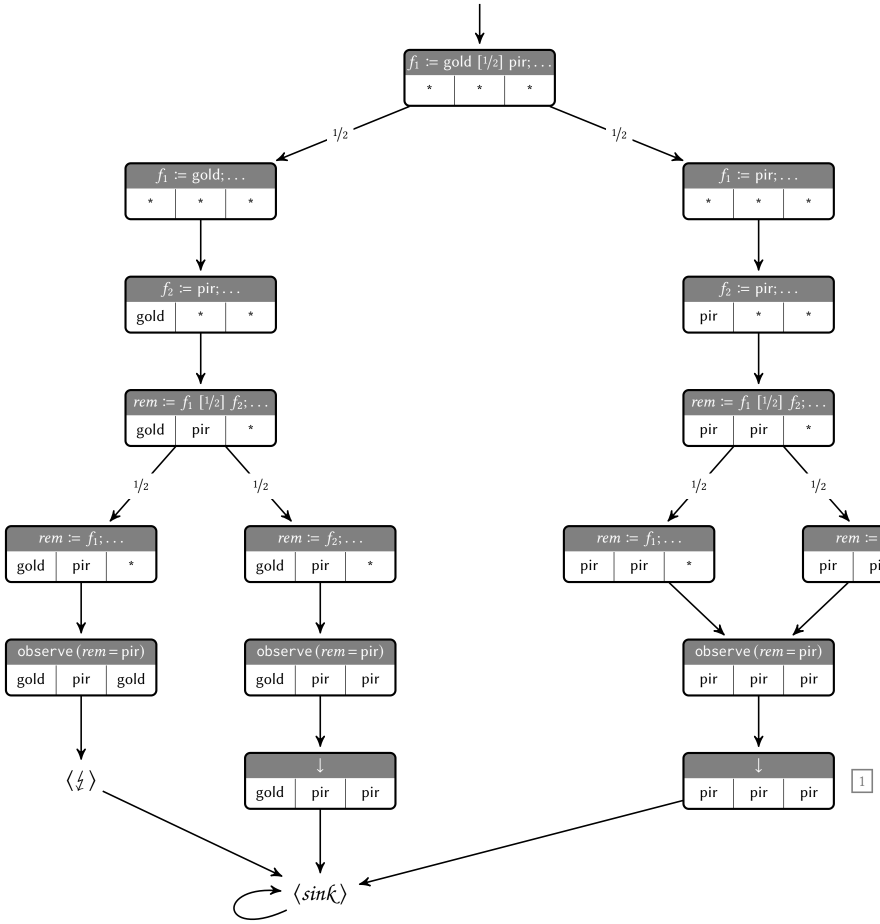


Fig. 6. Operational Markov reward chain $\mathfrak{R}_s^{[f_1=\text{pir}]}[\![c_{fish}]\!]$ associated to program c_{fish}, initial state s and post–expectation $[f_1 = \text{pir}]$. Intermediate computation states are represented by boxes, whose top most row contains the program fragment left to execute (we display only its initial instruction) and the bottom most row contains the program state at that point (from left to right, the value of variables f_1, f_2 and rem). When a transition occurs with probability one, we omit the probability in the respective edge. Only one state of the Markov chain has positive reward (of one), which is depicted on one side of the state, using a gray box.

(3) contains more "intermediate" states where only the program fragment left to execute is updated (the program state remains untouched), e.g. upon probabilistic choices. This is basically a design decision related to the granularity that we have chosen for our computational steps.

The OMRC $\mathfrak{R}_s^{[f_1=\text{pir}]}[\![c_{fish}]\!]$ depicted in Figure 6 is associated to post–expectation $[f_1 = \text{pir}]$. Terminal state $\langle{\downarrow}, s'\rangle$ with $s' = \langle f_1 \mapsto \text{pir}, f_2 \mapsto \text{pir}, rem \mapsto \text{pir}\rangle$ is the only one that establishes the post–expectation and has thus reward 1 (signaled alongside within a gray box); all the remaining states of the Markov chain have reward 0. The conditional expected reward

$$\mathsf{CER}^{\mathfrak{M}}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right) \;=\; \frac{\sum_{\hat{\pi} \in \Diamond \mathit{sink} \cap \neg \Diamond \lightning} \mathsf{Pr}^{\mathfrak{M}}(\hat{\pi}) \cdot r(\hat{\pi})}{\mathsf{Pr}^{\mathfrak{M}}(\neg \Diamond \lightning)}$$

over this Markov chain, abbreviated $\mathfrak{M}$, yields then the probability that program c_{fish} establishes $f_1 = \mathsf{pir}$ from state s. Let us determine concrete values for the numerator and denominator above. As for the numerator, the set $\Diamond \mathit{sink} \cap \neg \Diamond \lightning$ contains three paths, but only two of them—the ones traversing $\langle \downarrow, s' \rangle$—have positive cumulated reward, of 1; these two paths have each probability $1/4$. As for the denominator, set $\neg \Diamond \lightning$ contains exactly the same three paths as $\Diamond \mathit{sink} \cap \neg \Diamond \lightning$, since the program has no diverging run. Their overall probability is $1/2 \cdot 1/2 \;+\; 1/2 \cdot (1/2 + 1/2)$. This yields

$$\mathsf{CER}^{\mathfrak{M}}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right) \;=\; \frac{1/4 \cdot 1 + 1/4 \cdot 1}{1/2 \cdot 1/2 \;+\; 1/2 \cdot (1/2 + 1/2)} \;=\; \frac{2}{3}\,,$$

and puts on formal basis the informal calculations in Section 2.1 to determine the probability that program c_{fish} establishes $f_1 = \mathsf{pir}$. △

In the above example the obtained OMRC is finite. In general, this is not necessarily the case. Consider, for instance, the program

$$\begin{array}{l} b \coloneqq \mathsf{true};\; n \coloneqq 0; \\ \mathtt{while}\,(b)\,\{b \coloneqq \mathsf{true}\;[p]\;\mathsf{false};\; n \coloneqq n+1\} \end{array}$$

that simulates a geometric distribution. One can show that the program terminates with probability 1. However, its associated OMRC is countably infinite since n can take arbitrarily large values.

A simple observation on the structure of the OMRCs allows simplifying the definition of program outcomes. By definition, the conditional (liberal) expected outcome $\mathsf{C(L)ER}^{\mathfrak{R}^f_s[\![c]\!]}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right)$ of program c is the expected reward

$$\mathsf{(L)ER}^{\mathfrak{R}^f_s[\![c]\!]}\left(\Diamond\, \mathit{sink} \cap \neg \Diamond \lightning\right)$$

normalized by $\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \lightning)$. But $\Diamond\, \mathit{sink} \cap \neg \Diamond \lightning$ gives the subset of $\Diamond\, \mathit{sink}$ with paths representing unblocked (terminating) runs, which are, in effect, the only ones with positive cumulated reward. Therefore we can safely replace $\Diamond\, \mathit{sink} \cap \neg \Diamond \lightning$ with $\Diamond\, \mathit{sink}$ in the reward above. This yields the alternative characterization for the conditional outcome of programs

$$\mathsf{C(L)ER}^{\mathfrak{R}^f_s[\![c]\!]}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right) \;=\; \frac{\mathsf{(L)ER}^{\mathfrak{R}^f_s[\![c]\!]}\left(\Diamond\, \mathit{sink}\right)}{\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \lightning)}\,, \tag{4}$$

which we will shortly use to establish a correspondence theorem between our two semantic models.

5.4 Correspondence Theorem

We now investigate the connection between the operational semantics of conditioned probabilistic programs with the expectation transformer semantics of Section 4. We start with some auxiliary results. The first result establishes a relation between (liberal) expected rewards upon reaching $\langle \mathit{sink} \rangle$ and weakest (liberal) pre–expectations.

Lemma 5.5. *For program $c \in \mathsf{cpGCL}$, state $s \in \mathbb{S}$ and expectations $f \in \mathbb{E}$, $g \in \mathbb{E}_{\leq 1}$,*

$$\begin{aligned} \mathsf{ER}^{\mathfrak{R}^f_s[\![c]\!]}\left(\Diamond\, \mathit{sink}\right) &\;=\; \mathsf{wp}[c](f)(s)\,, \;\; \textit{and} \\ \mathsf{LER}^{\mathfrak{R}^g_s[\![c]\!]}\left(\Diamond\, \mathit{sink}\right) &\;=\; \mathsf{wlp}[c](g)(s)\,. \end{aligned}$$

Proof. By induction on the structure of c; see Appendix A.4 for details. □

The next result states that the probability of never visiting $\langle \lightning \rangle$ coincides with the weakest liberal pre–expectation of post–expectation $\mathbf{1}$.

Lemma 5.6. *For program $c \in \mathsf{cpGCL}$, state $s \in \mathbb{S}$ and expectation $f \in \mathbb{E}$,*

$$\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \lightning) \;=\; \mathsf{wlp}[c](\mathbf{1})(s)\;.$$

Proof. A direct inspection of Figure 4 reveals that paths in $\neg \Diamond \lightning$ avoiding state $\langle \lightning \rangle$ can be classified into two (disjoint) categories. Either (a) they represent a successful program run and visit a terminal state $\langle \downarrow, s' \rangle$ for some $s' \in \mathbb{S}$, or (b) they represent a diverging run. The set of "(a)–paths" is just $\Diamond T$ for $T = \{\langle \downarrow, s' \rangle \mid s' \in \mathbb{S}\}$, while the set of "$(b)$–paths" is $\neg \Diamond \mathit{sink}$, since paths reaching $\langle \mathit{sink} \rangle$ are exactly those that represent terminating runs. Thus,

$$\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \lightning) \;=\; \mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\Diamond T) \;+\; \mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \mathit{sink})\;.$$

Observe now that every terminal state (in T) evolves with probability one into $\langle \mathit{sink} \rangle$ and the remaining paths reaching $\langle \mathit{sink} \rangle$ have cumulated reward zero (because they reach $\langle \mathit{sink} \rangle$ via $\langle \lightning \rangle$). Then by assigning reward one to terminal states and reward zero to all other states, we can recast the probability of reaching a terminal state as an expected reward, i.e.

$$\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\Diamond T) \;=\; \mathsf{ER}^{\mathfrak{R}^{\mathbf{1}}_s[\![c]\!]}\,(\Diamond \mathit{sink})\;.$$

Overall, this yields $\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \lightning) = \mathsf{LER}^{\mathfrak{R}^{\mathbf{1}}_s[\![c]\!]}\,(\Diamond \mathit{sink})$ and a direct application of Lemma 5.5 concludes the proof. □

We now have all prerequisites to present the main result of this section, namely the correspondence between the operational and expectation transformer semantics of cpGCL programs. It turns out that the conditional weakest pre–expectation $\mathsf{cwp}[c](f)(s)$ coincides with the conditional expected reward in the OMRC $\mathfrak{R}^f_s[\![c]\!]$ of terminating (i.e. reaching $\langle \mathit{sink} \rangle$) while never violating an observation (i.e. avoiding $\langle \lightning \rangle$).

Theorem 5.7 (Correspondence theorem). *For program $c \in \mathsf{cpGCL}$, state $s \in \mathbb{S}$ and expectations $f \in \mathbb{E}$, $g \in \mathbb{E}_{\leq 1}$,*

$$\begin{aligned}\mathsf{CER}^{\mathfrak{R}^f_s[\![c]\!]}\,(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning) \;&=\; \mathsf{cwp}[c](f)(s)\;, \textit{ and}\\ \mathsf{CLER}^{\mathfrak{R}^g_s[\![c]\!]}\,(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning) \;&=\; \mathsf{cwlp}[c](g)(s)\;.\end{aligned}$$

Proof. Consider the first equation. As shown below, we can readily transform the left–hand side into the right–hand side by applying first Equation (4), then Lemmas 5.5 and 5.6, and finally Equation (3).

$$\mathsf{CER}^{\mathfrak{R}^f_s[\![c]\!]}\,(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning) \;=\; \frac{\mathsf{ER}^{\mathfrak{R}^f_s[\![c]\!]}\,(\Diamond\, \mathit{sink})}{\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]}(\neg \Diamond \lightning)} \;=\; \frac{\mathsf{wp}[c](f)(s)}{\mathsf{wlp}[c](\mathbf{1})(s)} \;=\; \mathsf{cwp}[c](f)(s)\;.$$

The proof of the second equation is similar. □

Theorem 5.7 extends a result by [25], who established a connection between an operational and expectation transformer semantics for *unconditioned* probabilistic programs. In contrast to our programming model, theirs also includes non–determinism. We thoroughly treat the interaction between non–determinism and conditioning in the next section.

6 NON–DETERMINISM

In this section, we investigate the extension of the programming language and its previously described semantics with (bounded) non–determinism. One of the primary goals of this paper is to extend the wp–semantics by McIver et al. [40, 41] with conditioning. Demonic non–determinism plays a key role in their work, and we therefore are interested in studying the interplay between conditioning and this form of non–determinism. Another motivation is that abstraction of program variables in probabilistic programs typically gives rise to (demonic) non–determinism. Resulting abstract programs in our setting thus exhibit conditioning as well as non-determinism, and the question is at stake how to treat this from a semantic point of view. Along the lines of McIver et al. and Dijkstra, this paper considers demonic non–determinism. We will show that *Markov decision processes* [46], a generalisation of Markov chains featuring non–determinism, provide a *natural interpretation* for conditioned non–deterministic programs. Expected rewards and the like, are defined subject to a given resolution of the non–determinism in the MDP, and the demonic nature gives naturally rise to taking the infimum over all possible resolutions. As a second result, we show that the expectation transformer semantics, on the contrary, is problematic in presence of both conditioning and non–determinism: our impossibility result asserts that there is *no possible (inductive) extension* of our conditional expectation transformer semantics that accounts also for non–deterministic programs.

6.1 Non–Deterministic Programs

To model non–deterministic programs we extend the cpGCL language with a binary *non–deterministic choice* construct, i.e.

$$C \;::=\; \ldots \mid \{C\} \,\square\, \{C\}\,,$$

leading to the so–called *non–deterministic* cpGCL language, abbreviated $\text{cpGCL}^{\square}$. Given programs c_1 and c_2, statement $\{c_1\} \,\square\, \{c_2\}$ represents a non–deterministic choice between c_1 and c_2. For the interpretation of a non–deterministic program, we follow [40] and assume a *demonic* model: for each individual post–expectation (and initial program state), the non–deterministic choices along the program execution are resolved by an adversary trying to *minimise* the resulting conditional weakest pre–expectation or conditional expected reward. To clarify this, consider, for instance, the program below that first sets variable x to either 0 or 1, with probability $1/2$ in each case, and then, non–deterministically, either keeps this value for x or resets it to 1.

$$\{x \coloneqq 0\} \; [1/2] \; \{x \coloneqq 1\};\; \{\texttt{skip}\} \,\square\, \{x \coloneqq 1\}$$

If we now want to determine the probability that $x = 0$ after the program execution, the demonic interpretation of non–determinism yields that this probability is zero, as the adversary will always prefer to reset the value of x to 1 because the other option would result in a greater probability, i.e. $1/2$.

As we have just illustrated, the demonic model of non–determinism provides the tightest lower bound that one can guarantee for a program's pre–expectation. The decision of adopting this model is not arbitrary: It constitutes the probabilistic counterpart of Dijkstra's original interpretation

$$\mathsf{wp}[\{c_1\} \,\square\, \{c_2\}](Q) \;=\; \mathsf{wp}[c_1](Q) \,\wedge\, \mathsf{wp}[c_2](Q)$$

of non–determinism for ordinary sequential programs [19].

6.2 Operational Semantics

Non–deterministic programs in $\text{cpGCL}^{\square}$ are interpreted as *Markov decision processes*. Markov decision processes can be seen as a generalisation of Markov chains, where to evolve from a given state σ, we first make a non–deterministic choice among the so–called *actions* enabled in σ, and then, given σ and the selected action, we proceed with a probabilistic choice of the successor state. Formally, we define a function *Act* that maps a state σ to a set $Act(\sigma)$ of enabled actions in state σ. The transition function is then a function mapping pairs (σ, α) to distributions over states, for $\alpha \in Act(\sigma)$.

Definition 6.1 (Markov decision process). A *Markov decision process* (MDP for short) is a tuple $\mathfrak{R} = (\Sigma,\ \sigma_I,\ Act, \mathcal{P})$, where Σ is a countable set of states, $\sigma_I \in \Sigma$ is the initial state, *Act* is a function mapping each state $\sigma \in \Sigma$ to the set of enabled actions in σ[10], and $\mathcal{P}\colon dom(\mathcal{P}) \rightarrow \mathcal{D}(\Sigma)$ is the transition function with $dom(\mathcal{P}) = \{(\sigma, \alpha) \mid \sigma \in \Sigma \wedge \alpha \in Act(\sigma)\}$. △

To clarify the role of actions in MDPs, consider our operational interpretation of cpGCL$^{\Box}$ programs. It will contain three possible actions:

- *left* and *right*, which are the ones enabled in states representing a non–deterministic choice (i.e. states of the form $\langle \{c_1\} \Box \{c_2\},\ s\rangle$). *left* represents taking the left branch of the non–deterministic choice, i.e. executing c_1, whereas *right* represents taking the right branch, i.e. executing c_2; and
- *default*, which is the default action enabled for all other states.

In general, the evolution of an MDP is dictated by a so–called *adversary* (aka: *scheduler*) that resolves the non–deterministic choices. The decision of adversaries may depend on the sequence of states visited so far (i.e. on the history); they are thus partial functions $\mathfrak{S}$ mapping finite state sequences onto actions such that $\mathfrak{S}(\sigma_0 \ldots \sigma_n) \in Act(\sigma_n)$ for every finite path $\sigma_0 \ldots \sigma_n$ in the domain of $\mathfrak{S}$. In our operational model of cpGCL$^{\Box}$ programs, adversaries will basically decide upon every occurrence of a non–deterministic choice, whether to take the left or right branch (possibly depending on the sequence of program states visited thus far).

Given an adversary, the evolution of an MDP is completely probabilistic. In effect, every adversary induces a Markov chain. This allows readily extending the notion of expected rewards from Markov chains to MDPs: One basically defines the expected reward of an MDP as the infimum over the expected reward of all possible induced Markov chains. Taking the infimum corresponds to demonic non–determinism as this amounts to minimising the expected reward. In the case of conditional rewards, this gives:

$$\mathsf{C(L)ER}^{\mathfrak{R}}\,(\Diamond\, T \mid \neg\Diamond\, U) \;\triangleq\; \inf_{\mathfrak{S} \in Adv(\mathfrak{R})} \mathsf{C(L)ER}^{\mathfrak{R}\|\mathfrak{S}}\,(\Diamond\, T \mid \neg\Diamond\, U)\ ,$$

where $Adv(\mathfrak{R})$ is the set of all adversaries of MDP $\mathfrak{R}$ and $\mathfrak{R}\|\mathfrak{S}$ the Markov chain induced by adversary $\mathfrak{S}$ in $\mathfrak{R}$. This corresponds to the conditional reward that the MDP can certainly guarantee, regardless of which choices are made by the adversary to resolve the non–determinism. This corresponds to demonic non–determinism.

We now have all the prerequisites to define the operational semantics of non–deterministic programs. The operational semantics of non–deterministic programs in cpGCL$^{\Box}$ follows in a similar manner to that of purely probabilistic programs in cpGCL (see Definition 5.3), the only difference being that now the model for programs is MDPs rather than MCs. The set of actions of the operational MDP of programs is as previously described (see paragraph below Definition 6.1). The transition function is defined by the set of rules in Figure 5, plus the following pair of rules to handle non–deterministic choices:

$$\textbf{[non-det-l]}\ \frac{}{\langle \{c_1\} \Box \{c_2\},\ s\rangle \xrightarrow{left} \langle c_1,\ s\rangle} \qquad \textbf{[non-det-r]}\ \frac{}{\langle \{c_1\} \Box \{c_2\},\ s\rangle \xrightarrow{right} \langle c_2,\ s\rangle}$$

[10] For technical reasons, we require that $Act(\sigma) \neq \emptyset$ for every state $\sigma \in \Sigma$.

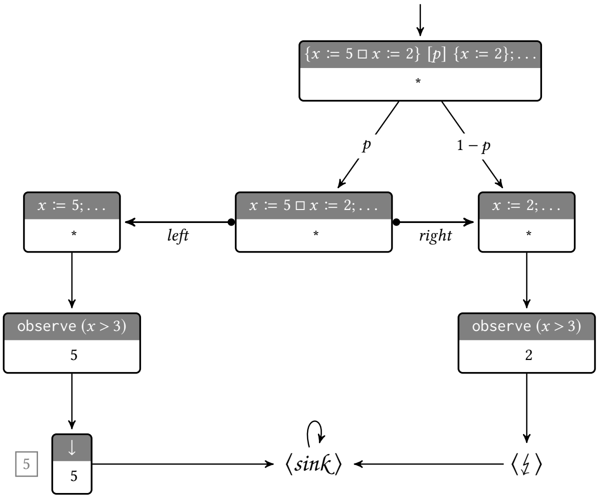


Fig. 7. Operational MDP $\mathfrak{R}^x_s[\![c_{nondet}]\!]$ associated to program c_{nondet}, initial state s and post–expectation x. Intermediate tates are represented by boxes, whose top most row contains the program fragment left to execute (we display only its initial instruction) and the bottom most row contains the program state at that point (value of variable x). Non–deterministic transitions are represented by arrows with a bold circle, labelled with the corresponding action (e.g. *left*); transitions from states that have only one enabled action (i.e. *default*) are omitted. Probabilistic transitions are labelled with the corresponding probabilities and the label is omitted if the probability is one. Only one (terminating) state of the MDP has positive reward (of 5), which is depicted on one side of the state, using a gray box.

Rule e.g. **[non-det-l]** should be read as follows: Being in state $\langle \{c_1\} \,\square\, \{c_2\},\ s\rangle$ and upon the (non–deterministic) election of action *left*, evolve into state $\langle c_1,\ s\rangle$ with probability one. The conditional (liberal) expected outcome of a non–deterministic program $c \in \mathrm{cpGCL}^{\square}$ with respect to post–expectation $f \in \mathbb{E}$ and initial state $s \in \mathbb{S}$ is given by the conditional (liberal) expected reward

$$\mathsf{C(L)ER}^{\mathfrak{R}^f_s[\![c]\!]}\,(\Diamond\ sink \mid \neg\Diamond \lightning) \;=\; \inf_{\mathfrak{S}\in Adv(\mathfrak{R}^f_s[\![c]\!])} \frac{\mathsf{(L)ER}^{\mathfrak{R}^f_s[\![c]\!]\,\|\,\mathfrak{S}}\,(\Diamond\ sink)}{\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]\,\|\,\mathfrak{S}}(\neg\Diamond \lightning)}\,,$$

of reaching $\langle sink\rangle$ from initial state $\langle c, s\rangle$, conditioned on not visiting $\langle \lightning\rangle$. This equation can be seen as the generalisation of Equation (4) to the case of non–deterministic programs. Accordingly, we call a non–deterministic program infeasible if the denominator $\mathsf{Pr}^{\mathfrak{R}^f_s[\![c]\!]\,\|\,\mathfrak{S}}(\neg\Diamond \lightning)$ becomes zero for some adversary $\mathfrak{S}^* \in Adv(\mathfrak{R}^f_s[\![c]\!])$. In this case, the conditional expected reward is undefined, denoted $\perp$ in the sequel.

Example 6.2. Consider the program

$$c_{nondet}\colon\quad \{x \coloneqq 5 \,\square\, x \coloneqq 2\}\ [p]\ \{x \coloneqq 2\};\ \mathtt{observe}\ (x > 3)$$

where with probability p either 5 or 2 is assigned non–deterministically to x, and with probability $1{-}p$, exactly 2 is assigned; after that we observe that $x > 3$. The operational model of the program is depicted in Figure 7. We are interested in computing the expected value of x and we consider thus the MDP $\mathfrak{R}^x_s[\![c_{nondet}]\!]$. The MDP admits two adversaries; in state

$$\sigma \;=\; \big\langle \{x \coloneqq 5\} \,\square\, \{x \coloneqq 2\};\ \mathtt{observe}\ (x > 3),\ s\big\rangle$$

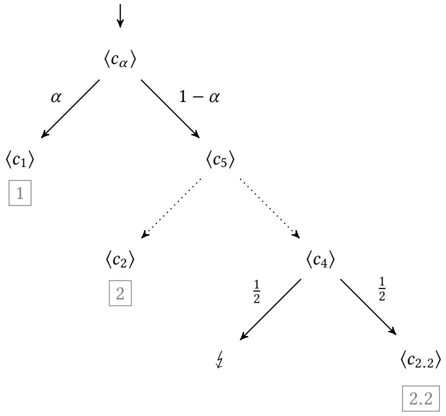


Fig. 8. Schematic depiction of the operational MDP $\mathfrak{R}_s^x[\![c_\alpha]\!]$.

one adversary selects action *left* and the other, action *right*. Consider the former adversary. In the induced MC the only path accumulating positive reward is the path π going from the initial state to the sink state through σ, and there taking action *left*. For this path we have $r(\pi) = 5$ and $\Pr(\pi) = p$; this gives an expected reward of $5 \cdot p$. The overall probability of not reaching $\langle \lightning \rangle$ is also p. The conditional expected reward of eventually reaching $\langle sink \rangle$ given that $\langle \lightning \rangle$ is not reached is hence ${}^{5 \cdot p}/_{p} = 5$. Consider now the latter adversary selecting action *right* in state σ. In this case, there is no path having positive cumulated reward in the induced MC, yielding an expected reward of 0. The probability of not reaching $\langle \lightning \rangle$ is also 0. The program is therefore infeasible and its outcome is not well–defined. △

6.3 Expectation Transformer Semantics

We now investigate the problems that occur when trying to provide an expectation transformer semantics for non–deterministic programs with conditioning. First, we show that we cannot simply extend the table in Figure 3 for non–deterministic programs. Thereafter, we provide a more general impossibility result.

6.3.1 Impossibility of an Inductive Extension of $\underline{\text{cw}}\text{p}$ *to Non–Deterministic Programs.* We argue that it is not possible to extend the rules for $\underline{\text{cw}}\text{p}$ given in Figure 3 such that the correspondence result Theorem 5.7 remains valid. The argument goes by contraposition. Consider the parametric program $c_\alpha = \{c_1\}\ [\alpha]\ \{c_5\}$, with

$$\begin{aligned} c_1 &= x := 1 \\ c_5 &= \{c_2\} \,\square\, \{c_4\} \\ c_2 &= x := 2 \\ c_4 &= \{\mathtt{observe}\ (\mathsf{false})\}\ [{}^1/_2]\ \{c_{2.2}\} \\ c_{2.2} &= x := 2.2\ , \end{aligned}$$

and $0 \leq \alpha \leq 1$; a schematic depiction of its operational MDP $\mathfrak{R}_s^x[\![c_\alpha]\!]$ is given in Figure 8. Assume now (for the purpose of a contraposition) that we can extend the rules in Figure 3, such that we have a rule for non–deterministic programs for which Theorem 5.7 remains valid. Then there exists some $(f,\, g)$, such that

$$\underline{\mathsf{cw}}\mathsf{p}[c_5](x,\, \mathbf{1}) \;=\; (f,\, g)\,,$$

and since Theorem 5.7 is supposed to remain valid, for any state s,

$$\mathsf{cwp}[c_5](x)(s) \;=\; \frac{f(s)}{g(s)} \;=\; 2 \;=\mathsf{CER}^{\mathfrak{R}^f_s[\![c_5]\!]}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right)\;.$$

This can be seen from the operational MDP as follows. In c_5, the minimal expected reward (of two) is obtained by selecting the transition to c_2. Selecting c_4 instead results in a reward of $\frac{1}{2} \cdot 2.2$ normalised by $\frac{1}{2}$, which equals 2.2. From the above it follows $g = f/2$, which in turn yields

$$\underline{\mathsf{cw}}\mathsf{p}[c_5](x,\, \mathbf{1}) \;=\; \left(f,\, \frac{f}{2}\right)\;. \tag{5}$$

Notice that in the non–deterministic choice of c_5 the *left* branch was preferred in order to minimise the conditional expected reward of x after execution of c_5.

Let $\alpha = 1/2$. We can now compute $\underline{\mathsf{cw}}\mathsf{p}$ for the entire program c_α:

$$\begin{aligned}
\underline{\mathsf{cw}}\mathsf{p}[c_\alpha](x,\, \mathbf{1}) \;&=\; \frac{1}{2}\cdot \underline{\mathsf{cw}}\mathsf{p}[x \coloneqq 1](x,\, \mathbf{1}) + \frac{1}{2}\cdot \underline{\mathsf{cw}}\mathsf{p}[c_5](x,\, \mathbf{1})\\
&=\; \frac{1}{2}\cdot(\mathbf{1},\, \mathbf{1}) + \frac{1}{2}\cdot\left(f,\, \frac{f}{2}\right)\\
&=\; \left(\frac{\mathbf{1}}{\mathbf{2}} + \frac{f}{2},\, \frac{\mathbf{1}}{\mathbf{2}} + \frac{f}{4}\right)
\end{aligned}$$

By Theorem 5.7, we obtain

$$\mathsf{cwp}[c_\alpha](x) \;=\; \frac{\frac{1}{2}+\frac{f}{2}}{\frac{1}{2}+\frac{f}{4}} \;=\; \frac{7}{5} \;=\; \mathsf{CER}^{\mathfrak{R}^f_s[\![c_\alpha]\!]}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right) \qquad\Longrightarrow\qquad f \;=\; \frac{\mathbf{4}}{\mathbf{3}}\,.$$

Using $f = 4/3$ and by recalling Equation (5), we establish

$$\underline{\mathsf{cw}}\mathsf{p}[c_5](x,\, \mathbf{1}) \;=\; \left(\frac{\mathbf{4}}{\mathbf{3}},\, \frac{\mathbf{2}}{\mathbf{3}}\right)\;.$$

Observe that $\underline{\mathsf{cw}}\mathsf{p}[c_5](x,\, \mathbf{1})$ is (as it should be) independent of α and that in the non–deterministic choice at c_5 the *right* branch was preferred so as to minimise the conditional expected reward of x after execution of c_α.

Now let $\alpha = 3/4$. Again, we derive the $\underline{\mathsf{cw}}\mathsf{p}$ of the entire program c_α by

$$\begin{aligned}
\underline{\mathsf{cw}}\mathsf{p}[c_\alpha](x,\, \mathbf{1}) \;&=\; \frac{3}{4}\cdot \underline{\mathsf{cw}}\mathsf{p}[x \coloneqq 1](x,\, \mathbf{1}) + \frac{1}{4}\cdot \underline{\mathsf{cw}}\mathsf{p}[c_5](x,\, \mathbf{1})\\
&=\; \frac{3}{4}\cdot(\mathbf{1},\, \mathbf{1}) + \frac{1}{4}\cdot\left(\frac{\mathbf{4}}{\mathbf{3}},\, \frac{\mathbf{2}}{\mathbf{3}}\right)\\
&=\; \left(\frac{\mathbf{13}}{\mathbf{12}},\, \frac{\mathbf{11}}{\mathbf{12}}\right)
\end{aligned}$$

But we have

$$\mathsf{cwp}[c_\alpha](x) \;=\; \frac{\frac{13}{12}}{\frac{11}{12}} \;=\; \frac{\mathbf{13}}{\mathbf{11}} \;<\; \frac{\mathbf{5}}{\mathbf{4}} \;=\; \mathsf{CER}^{\mathfrak{R}^f_s[\![c_\alpha]\!]}\left(\Diamond\, \mathit{sink} \mid \neg \Diamond \lightning\right)\;.$$

This contradicts the assumption that Theorem 5.7 holds. Thus, the assumption that we can assign a unique pair $(f,\, g)$ to $\underline{\mathrm{cw}}\mathrm{p}[c_5](x,\, \mathbf{1})$, independent of the context that the program c_5 is put into, was wrong and thus we cannot extend the rules for $\underline{\mathrm{cw}}\mathrm{p}$ to non–deterministic programs.

6.3.2 Non–Existence of Inductive Conditional Weakest Pre–Expectation Transformers. We now argue why (under mild assumptions) it is not possible at all to come up with a denotational semantics in the style of conditional pre–expectation transformers (CPETs for short) for full cpGCL. To show this, it suffices to consider a simple fragment of cpGCL containing only assignments, observations, probabilistic and non–deterministic choices. Let x be the only program variable that can be written or read in this fragment. We denote this fragment by cpGCL$^-$. Assume D is some appropriate domain for *representing* conditional expectations of the program variable x and let $\llbracket\,\cdot\,\rrbracket \colon D \to \mathbb{R} \cup \{\bot\}$ be an interpretation function such that for any $d \in D$ we have that $\llbracket d \rrbracket$ *is equal to* the (possibly undefined) conditional expected value of x with respect to some *fixed* initial state s_0 .

Definition 6.3 (Inductive CPETs). A *CPET* is a function $\underline{\mathrm{cw}}\mathrm{p}^* \colon \mathrm{cpGCL}^- \to D$ such that for any $c \in \mathrm{cpGCL}^-$, $\llbracket \underline{\mathrm{cw}}\mathrm{p}^*[c] \rrbracket = \mathrm{CER}^{\mathfrak{R}^{x}_{s_0}\llbracket c \rrbracket}(\Diamond\, \mathit{sink} \mid \neg\Diamond\, ↯)$. $\underline{\mathrm{cw}}\mathrm{p}^*$ is called *inductive*, if there exists some function $\mathcal{K} \colon D \times [0,\, 1] \times D \to D$ such that for any $c_1, c_2 \in \mathrm{cpGCL}^-$,

$$\underline{\mathrm{cw}}\mathrm{p}^*[\{c_1\}\ [p]\ \{c_2\}] \;=\; \mathcal{K}(\underline{\mathrm{cw}}\mathrm{p}^*[c_1],\, p,\, \underline{\mathrm{cw}}\mathrm{p}^*[c_2])\,,$$

and some function $\mathcal{N} \colon D \times D \to D$ with

$$\underline{\mathrm{cw}}\mathrm{p}^*[\{c_1\} \Box \{c_2\}] \;=\; \mathcal{N}(\underline{\mathrm{cw}}\mathrm{p}^*[c_1],\, \underline{\mathrm{cw}}\mathrm{p}^*[c_2])\,,$$

where $\forall d_1, d_2 \in D \colon \mathcal{N}(d_1,\, d_2) \in \{d_1,\, d_2\}$. △

This definition requires that the conditional pre–expectation of $\{c_1\}\ [p]\ \{c_2\}$ is determined only by the conditional pre–expectation of c_1, the conditional pre–expectation of c_2, and the probability p. Furthermore, the above definition requires that the conditional pre–expectation of $\{c_1\} \Box \{c_2\}$ is determined by the conditional pre–expectation of c_1 and the conditional pre–expectation of c_2 only. Consequently, the non–deterministic choice can be resolved by replacing it either by c_1 or c_2, which is the traditional assumption in the field of program refinement [4]. Notice that these assumptions are crucial to our impossibility result.

As we assume a fixed initial state and a fixed post–expectation, the non–deterministic choice turns out to be deterministic once the pre–expectations of c_1 and c_2 are known. Under the above assumptions (which do apply to the wp and wlp transformers) we claim:

Theorem 6.4. *There exists no inductive CPET.*

Proof. The proof goes by contraposition and basically shows that non–deterministic choices cannot be resolved without taking the *context* of a program into account. In particular we show that the non–deterministic choice in subprogram c_5 of program c_α from Section 6.3.1 has to be resolved in different ways depending on whether c_5 stands alone or is put into context $\{c_1\}\ [\alpha]\ \{c_5\}$.

For the proof, reconsider therefore the program c_α from Section 6.3.1 and choose $\alpha = 1/2$. Assume there exists an inductive CPET $\underline{\mathrm{cw}}\mathrm{p}^*$ over some appropriate domain D. Then,

$$\begin{aligned}
\underline{\mathrm{cw}}\mathrm{p}^*[c_1] &= d_1, \text{ with } \llbracket d_1 \rrbracket = 1 \\
\underline{\mathrm{cw}}\mathrm{p}^*[c_2] &= d_2, \text{ with } \llbracket d_2 \rrbracket = 2 \\
\underline{\mathrm{cw}}\mathrm{p}^*[c_{2.2}] &= d_{2.2}, \text{ with } \llbracket d_{2.2} \rrbracket = 2.2 \\
\underline{\mathrm{cw}}\mathrm{p}^*[\texttt{observe false}] &= \mathfrak{uf}, \text{ with } \llbracket \mathfrak{uf} \rrbracket = \bot
\end{aligned}$$

for some appropriate d_1, d_2, $d_{2.2}$, $\mathfrak{of} \in D$. By Definition 6.3, $\underline{\text{cwp}}^*$ being inductive requires the existence of a function $\mathcal{K}$, such that

$$\begin{aligned}\underline{\text{cwp}}^*[c_4] &= \mathcal{K}\left(\underline{\text{cwp}}^*[\texttt{observe false}],\ {}^1\!/\!{}_2,\ \underline{\text{cwp}}^*[c_{2.2}]\right)\\ &= \mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2.2})\ .\end{aligned}$$

In addition, there must be an $\mathcal{N}$ with:

$$\begin{aligned}\underline{\text{cwp}}^*[c_5] &= \mathcal{N}\left(\underline{\text{cwp}}^*[c_2],\ \underline{\text{cwp}}^*[c_4]\right)\\ &= \mathcal{N}(d_2,\ \mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2.2}))\ .\end{aligned}$$

Since c_4 is a probabilistic choice between an infeasible branch and $c_{2.2}$, the expected value for x has to be rescaled to the feasible branch. Hence $[\![\underline{\text{cwp}}^*[c_4]]\!] = 2.2$, whereas $[\![\underline{\text{cwp}}^*[c_2]]\!] = 2$. Thus:

$$[\![\ \underbrace{d_2}_{\underline{\text{cwp}}^*[c_2]}\ ]\!] \ \lnapprox\ [\![\underbrace{\mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2.2})}_{\underline{\text{cwp}}^*[c_4]}]\!] \tag{6}$$

As non–deterministic choice is demonic, we have

$$\underline{\text{cwp}}^*[c_5] \ = \ \mathcal{N}(d_2,\ \mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2.2})) \ = \ d_2\ , \tag{7}$$

since by Definition 6.3, $\mathcal{N}$ can only select either d_2 or $\mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2.2})$ and it has to select the minimum of the two options. As $\mathcal{N}\left(\underline{\text{cwp}}^*[c_2],\ \underline{\text{cwp}}^*[c_4]\right) \in \{\underline{\text{cwp}}^*[c_2],\ \underline{\text{cwp}}^*[c_4]\}$ (again by Definition 6.3) we can resolve non–determinism in c_α by either rewriting c_α to $\{c_1\}\ [{}^1\!/\!{}_2]\ \{c_2\}$ which gives

$$[\![\underline{\text{cwp}}^*\{c_1\}\ [{}^1\!/\!{}_2]\ \{c_2\}]\!] \ = \frac{3}{2} \ = \ 1.5\ ,$$

or we rewrite c_α to $\{c_1\}\ [{}^1\!/\!{}_2]\ \{c_4\}$, which gives

$$[\![\underline{\text{cwp}}^*\{c_1\}\ [{}^1\!/\!{}_2]\ \{c_4\}]\!] \ = \frac{7}{5} \ = \ 1.4\ .$$

Since $1.4 < 1.5$, the second option should be preferred by a demonic adversary. This, however, requires that:

$$\begin{aligned}\underline{\text{cwp}}^*[c_5] &= \mathcal{N}(d_2,\ \mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2+\varepsilon}))\\ &= \mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2+\varepsilon})\end{aligned}$$

Together with Equality (7) we get $d_2 = \mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2+\varepsilon})$, which implies $[\![d_2]\!] = [\![\mathcal{K}(\mathfrak{of},\ {}^1\!/\!{}_2,\ d_{2+\varepsilon})]\!]$. This contradicts the inequality (6). □

This result is related to the fact that for minimising conditional (reachability) probabilities in RMDPs positional, i.e. history–independent, adversaries are insufficient [2]. Intuitively speaking, if a *history–dependent* adversary is required, this necessitates the inductive definition of $\underline{\text{cwp}}^*$ to take the context of a statement (if any) into account. This conflicts with the principle of an inductive definition.

7 APPLICATIONS

In this section we study some applications that make use of our semantics to analyse conditioned probabilistic programs. First, we present a program transformation that hoists `observe` statements all the way up of programs delivering an `observe`–free program equivalent to the original. Second, we present a technique based on rejection sampling that simulates the `observe` statements of a program by enclosing (a slightly modified version of) the program in a global loop. These two transformations show that `observe` statements are, to some degree, syntactic sugar. Lastly, we show that loops with no information flow across iterations can be substituted by their mere body, followed by a conditioning on the loop guard.

7.1 Hoisting Observations

We introduce a semantics–preserving transformation for removing observations from conditioned probabilistic programs and establish its correctness using the expectation transformer semantics from Section 4. Intuitively, the program transformation “hoists” the `observe` statements and along the way updates the probabilities of the probabilistic choices. Given program c, the transformation delivers a semantically equivalent `observe`–free program $\hat{c}$ and—as a side product—an expectation $\hat{h} \in \mathbb{E}_{\leq 1}$ that captures the probability of the original program c to establish all observations. To illustrate this, reconsider program c_{fish} modeling our “goldfish–piranha” problem (see Section 2). The transformation yields program $\hat{c}_{fish}$ on the right, where

$$p = \frac{\frac{1}{2}\cdot[f_1 = \mathsf{pir}]}{\frac{1}{2}\cdot[f_1 = \mathsf{pir}]+\frac{1}{2}\cdot[f_2 = \mathsf{pir}]} ,$$

together with the expectation $\hat{h} = {}^3\!/\!_4$. The probability that $f_1 = \mathsf{pir}$ in $\hat{c}_{fish}$ is ${}^2\!/\!_3$, which agrees with the probability in the original program, see Example 4.2.

```
$f_1$ := gold [$1/3$] pir;
$f_2$ := pir;
$rem$ := $f_1$ [$p$] $f_2$
```

Notice that the programs yielded by this transformation belong to a slightly more general class of probabilistic programs, namely those in which the probabilities in the probabilistic choices may depend on the current program state (recall the remark from Page 9). These mappings from program states to probabilities may in some cases even be noncomputable. This is due to the fact that the rule for while loops involves a greatest fixed point construct, that may then enter the probability of a probabilistic choice by the according rule.

To apply the transformation to a program c we need to determine $\mathcal{T}(c, \mathbf{1})$, which gives the semantically equivalent program $\hat{c}$ and the expectation $\hat{h}$. The transformation $\mathcal{T}$ is defined in Figure 9 and works by inductively computing the weakest pre–expectation that guarantees the establishment of all `observe` statements and updating the probability parameter of probabilistic choices so that the pre–expectations of their branches are established in accordance with the original probability parameter. The computation of these pre–expectations is performed following the same rules as the wlp operator. The correctness of the transformation is established by the following theorem, which states that a program and its transformed version share the same terminating and non–terminating behavior.

Theorem 7.1 (Correctness of observation hoisting). *Let $c \in$ cpGCL admit at least one feasible run for every initial state*[11] *and $\mathcal{T}(c, \mathbf{1}) = \big(\hat{c},\ \hat{h}\big)$. Then for all $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leq 1}$,*

$$\mathsf{wp}[\hat{c}](f) \;=\; \mathsf{cwp}[c](f) \qquad \textit{and} \qquad \mathsf{wlp}[\hat{c}](g) \;=\; \mathsf{cwlp}[c](g)\,.$$

Proof. By the alternative characterisation of transformers cw(l)p (Equation (3)), the statement follows from the equations

$$\hat{h} \;=\; \mathsf{wlp}[c](\mathbf{1})\,, \qquad \hat{h}\cdot\mathsf{wp}[\hat{c}](f) \;=\; \mathsf{wp}[c](f) \quad \text{and} \quad \hat{h}\cdot\mathsf{wlp}[\hat{c}](f) \;=\; \mathsf{wlp}[c](f)\,,$$

which are established by Lemma A.4 in Appendix A.5, taking $h = \mathbf{1}$. □

A similar program transformation has been given for the programming language R2 in [42]. Let us point out some differences. R2 uses random assignments rather than probabilistic choices. Consequently, `observe` statements can only be hoisted until the occurrence of a random assignment. In our setting, `observe` statements are hoisted through probabilistic choices. This enables completely

[11] We require that c admits a feasible run from every initial state to ensure the well–definedness of $\mathsf{cwp}[c](f)$ and $\mathsf{cwlp}[c](g)$.

$$
\begin{array}{lll}
\mathcal{T}(\texttt{skip},\ f) & = & (\texttt{skip},\ f) \\
\mathcal{T}(\texttt{abort},\ f) & = & (\texttt{abort},\ \mathbf{1}) \\
\mathcal{T}(x \coloneqq E,\ f) & = & (x \coloneqq E,\ f[x/E]) \\
\mathcal{T}(\texttt{observe}\,(G),\ f) & = & (\texttt{skip},\ [G]\cdot f) \\
\mathcal{T}(c_1; c_2,\ f) & = & (c_1'; c_2',\ f'') \text{ where} \\
 & & (c_2', f') = \mathcal{T}(c_2, f),\ (c_1', f'') = \mathcal{T}(c_1, f') \\
\mathcal{T}(\texttt{ite}\,(G)\,\{c_1\}\,\{c_2\},\ f) & = & (\texttt{ite}\,(G)\,\{c_1'\}\,\{c_2'\},\ [G]\cdot f_1 + [\neg G]\cdot f_2) \text{ where} \\
 & & (c_1', f_1) = \mathcal{T}(c_1, f),\ (c_2', f_2) = \mathcal{T}(c_2, f) \\
\mathcal{T}(\{c_1\}\ [p]\ \{c_2\},\ f) & = & (\{c_1'\}\ [p']\ \{c_2'\},\ p\cdot f_1 + (\mathbf{1}{-}p)\cdot f_2) \text{ where} \\
 & & (c_1', f_1) = \mathcal{T}(c_1, f),\ (c_2', f_2) = \mathcal{T}(c_2, f),\ p' = \frac{p\cdot f_1}{p\cdot f_1 + (\mathbf{1}-p)\cdot f_2} \\
\mathcal{T}(\texttt{while}\,(G)\,\{c\},\ f) & = & (\texttt{while}\,(G)\,\{c'\},\ f') \text{ where} \\
 & & f' = gfp(\mathcal{H}),\ \mathcal{H}(h) = [G]\cdot(\pi_2 \circ \mathcal{T})(c, h) + [\neg G]\cdot f,\ (c', \_) = \mathcal{T}(c, f')
\end{array}
$$

Fig. 9. Program transformation for eliminating observations from cpGCL programs.

removing observe statements from programs. Another difference is, as discussed in more depth at the end of Section 4, the treatment of diverging programs. As R2 focuses on certainly terminating programs, the hoisting program transformation in [42] is correct for such programs. Our semantics treats possibly diverging programs, too. The presented hoisting program transformation is correct for such programs as well. This of relevance in a setting where it is not clear upfront whether a probabilistic program may diverge or not. Deciding whether a probabilistic program has a positive probability to diverge or not is as hard as the universal halting problem [32]; it is thus beneficial that program transformations are generally applicable.

7.2 Replacing Observations by Loops

We now study an alternative approach for removing observations from programs, while preserving their semantics. The approach can be seen as an instance of the rejection sampling method (RSM) applied to a conditional distribution [10, 47]. To understand the intuition behind this method, consider first this simpler problem: *Assume Alice wants to simulate a six-sided die but to this end she has only (fair) coins. Can she still do it?* The answer to the problem is "Yes, she can!" and the program on the right illustrates the solution. The body of the loop simulates a uniform distribution over the interval $[1, 8]$, which is repeatedly sampled (in variable i) until its outcome lies in the interval $[1, 6]$. The effect of the repeated sampling is precisely to condition the distribution of i to $1 \leq i \leq 6$. As a result, $\Pr[i = N] = 1/6$ for all $N = 1, \ldots, 6$ [49, Th 9.2].

```
$c_{die}$:   repeat
                 $a_0, a_1, a_2 \coloneqq 0\ [1/2]\ 1;$
                 $i \coloneqq 4a_0 + 2a_1 + a_2 + 1$
             until $(1 \leq i \leq 6)$
```

To apply this method to our original problem of removing program observations, we follow a similar idea: We repeatedly sample executions from the program until seeing an execution that passes all the observations. To implement this, we take the following steps: First, we introduce a flag *unblocked* that signals whether all observations along a program execution have been satisfied. We let variable *unblocked* be initially true and replace every statement observe (G) from the original program by the assignment $\mathit{unblocked} \coloneqq \mathit{unblocked} \wedge G$. In this way, variable *unblocked* remains

$$
\begin{aligned}
\mathcal{B}(\texttt{skip}) &= \texttt{skip}\\
\mathcal{B}(\texttt{abort}) &= \texttt{ite}\,(\mathit{unblocked})\,\{\texttt{abort}\}\,\{\texttt{skip}\}\\
\mathcal{B}(x := E) &= x := E\\
\mathcal{B}(\texttt{observe}\,(G)) &= \mathit{unblocked} := \mathit{unblocked} \wedge G\\
\mathcal{B}(c_1; c_2) &= \mathcal{B}(c_1); \mathcal{B}(c_2)\\
\mathcal{B}(\texttt{ite}\,(G)\,\{c_1\}\,\{c_2\}) &= \texttt{ite}\,(G)\,\{\mathcal{B}(c_1)\}\,\{\mathcal{B}(c_2)\}\\
\mathcal{B}(\{c_1\}\;[p]\;\{c_2\}) &= \{\mathcal{B}(c_1)\}\;[p]\;\{\mathcal{B}(c_2)\}\\
\mathcal{B}(\texttt{while}\,(G)\,\{c\}) &= \texttt{while}\,(G \wedge \mathit{unblocked})\,\{\mathcal{B}(c)\}
\end{aligned}
$$

(a) Tranformation that removes observations from programs and, instead, signals (un)blocked execution in variable *unblocked*. Moreover, it prevents the program divergence when *unblocked* turns to false.

```
1: $s_1, \ldots, s_n := x_1, \ldots, x_n$;
2: repeat
3:     $\mathit{unblocked} := \mathsf{true}$;
4:     $x_1, \ldots, x_n := s_1, \ldots, s_n$;
5:     $\mathcal{B}(c)$
6: until $(\mathit{unblocked})$
```

(b) observe-free program rsm[c] that simulates (conditioned) program c by repeatedly sampling executions from $\mathcal{B}(c)$.

Fig. 10. Simulation of conditioned programs based on rejection sampling.

true until an observation is violated. Secondly, since program executions are no longer blocked on violating an observation, we need to modify the program to avoid any possible subsequent divergence. This is achieved by guarding `abort` statements and loops with variable *unblocked*. These adaptations are captured in detail by program transformation $\mathcal{B}$ in Figure 10a. Finally, we need to keep a permanent copy of the initial program state since every time we sample an execution, the program must start from its original initial state. All in all, this gives the unconditioned program rsm[c] depicted in Figure 10b, which simulates the behaviour of the original program c.[12] There, $x_1, \ldots, x_n$ denote the set of variables that occur in the original program c and $s_1, \ldots, s_n$ are auxiliary variables used to store the initial program state; note that if the original program is closed (i.e. independent of its input), Lines 1 and 4 can be omitted. Line 5 includes the modified version $\mathcal{B}(c)$ of the original program c, which accounts for the replacement of observations by flag updates and guarding of `abort` statements and loops. For convenience, we use a repeat–until loop to describe program rsm[c]. Even though this type of loops is not formally contained in cpGCL, this deviation does no harm since repeat–until loops are syntactic sugar: $\texttt{repeat}\,\{c\}\,(G) \;\equiv\; c; \texttt{while}\,(\neg G)\,\{c\}$.

To illustrate the application of this method, reconsider the program c_{fish} from Section 2. The equivalent program rsm[c_{fish}] is given on the right. In the general case, to prove that (the unconditioned program) rsm[c] correctly simulates (the conditioned program) c we resort to the operational semantics from Section 6. However, we state the correctness of the simulation using the expectation transformer semantics so that we keep the presentation of all our results consistent.

```
rsm[$c_{fish}$]:  repeat
                    $\mathit{unblocked} := \mathsf{true}$;
                    $f_1 := \mathsf{gold}\ [1/2]\ \mathsf{pir}$;
                    $f_2 := \mathsf{pir}$;
                    $\mathit{rem} := f_1\ [1/2]\ f_2$;
                    $\mathit{unblocked} := \mathit{unblocked} \wedge (\mathit{rem} = \mathsf{pir})$
                  until $(\mathit{unblocked})$
```

[12] An implicit assumption here is that all expressions over program variables in rsm[c] are well-defined. This hinders the application of the method for programs such as $c = \texttt{observe}\,(x > 0); \texttt{ite}\,(1/x \le 0.5)\,\{\ldots\}\,\{\ldots\}$ because executions with $x = 0$ are no longer blocked in rsm[c]. We can get rid of this limitation by in transformation $\mathcal{B}$, guarding *all* program instructions like we do with `abort` statements.

Theorem 7.2 (Correctness of simulation by RSM). *Let* $c \in \mathsf{cpGCL}$ *be a feasible program from initial state* $s \in \mathbb{S}$. *Then for all* $f \in \mathbb{E}$,

$$\mathsf{cwp}[c](f) \;=\; \mathsf{wp}[\mathsf{rsm}[c]](f)\;.$$

Proof. In Appendix A.6. □

The underlying idea behind our program transformation $c \rightsquigarrow \mathsf{rsm}[c]$ has also been exploited by [6] to reason about conditional probabilities over system models: Given a Markov chain $\mathfrak{M}$ and a condition ψ, they show how to construct a Markov chain $\mathfrak{M}_\psi$ such that the conditional probabilities in $\mathfrak{M}$ agree with the (unconditional) probabilities in $\mathfrak{M}_\psi$.

Taken together, Theorems 7.1 and 7.2 provide two different approaches to simulate observations using the remaining cpGCL constructs, under mild conditions of program feasibility. They show that `observe` statements are, to some degree, syntactic sugar.

7.3 Replacing Loops by Observations

In some circumstances it is possible to apply a dual program transformation that replaces loops with observations. This is applicable when the set of states reached at the end of the different loop iterations are independent and identically distributed (i.i.d., for short). This is the case e.g., for the earlier program c_{die} that simulates a six-sided die. One can show that the program is semantically equivalent to the program on the right, where the effect of the loop is simulated by an observation. This kind of transformation is particularly useful because it reduces the program verification effort: it is usually easier to analyse a loop–free program with observations than a program with loops, whose analysis relies on loop invariants. In the sequel, let $\mathtt{repeat}\,\{c\}\,(G)$ be a shorthand for $\mathtt{repeat}\,c\,\mathtt{until}\,(G)$.

```
$a_0, a_1, a_2 := 0\ [1/2]\ 1;$
$i := 4a_0 + 2a_1 + a_0 + 1;$
observe $(1 \leq i \leq 6)$
```

The transformation allows replacing a loop with its body, followed by an observation conditioning to the loop guard, i.e. $\mathtt{repeat}\,\{c\}\,(G)$ with c; $\mathtt{observe}\,(G)$. To formally define the class of "i.i.d." loops to which the transformation applies, we require the notion of *n-unrolling* of a loop, given by the following clauses:

$$\begin{aligned}\mathtt{repeat}_0\,\{c\}\,(G) &\;\triangleq\; \mathtt{abort}\\ \mathtt{repeat}_{n+1}\,\{c\}\,(G) &\;\triangleq\; c;\, \mathtt{ite}\,(\neg G)\,\{\mathtt{repeat}_n\,\{c\}\,(G)\}\,\{\mathtt{skip}\}\;.\end{aligned}$$

Applying transformer wp to both sides of the last equation yields

$$\mathsf{wp}[\mathtt{repeat}_{n+1}\,\{c\}\,(G)](f) \;=\; \mathsf{wp}[c]\big([\neg G]\cdot \mathsf{wp}[\mathtt{repeat}_n\,\{c\}\,(G)](f)\big) \;+\; \mathsf{wp}[c]([G]\cdot f)\;.$$

For our intended notion of "i.i.d." loop, the left summand above can be replaced with

$$\mathsf{wp}[c]([\neg G])\cdot \mathsf{wp}[\mathtt{repeat}_n\,\{c\}\,(G)](f)\;,$$

because when executing $\mathtt{repeat}_{n+1}\,\{c\}\,(G)$, if G is not established after the first iteration, we can continue the execution from the initial state instead of the state reached after the (failed) iteration. This observation leads to our definition of i.i.d. loops below.

Definition 7.3 (i.i.d. loop). Given program $c \in \mathsf{cpGCL}$ and guard G, we say that loop $\mathtt{repeat}\,\{c\}\,(G)$ is *i.i.d.* if for all $n \in \mathbb{N}$, $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leq 1}$,

$$\begin{aligned}\mathsf{wp}[\mathtt{repeat}_{n+1}\,\{c\}\,(G)](f) &\;=\; \mathsf{wp}[c]([\neg G])\cdot \mathsf{wp}[\mathtt{repeat}_n\,\{c\}\,(G)](f) \;+\; \mathsf{wp}[c]([G]\cdot f)\,, \quad \text{and}\\ \mathsf{wlp}[\mathtt{repeat}_{n+1}\,\{c\}\,(G)](g) &\;=\; \mathsf{wp}[c]([\neg G])\cdot \mathsf{wlp}[\mathtt{repeat}_n\,\{c\}\,(G)](g) \;+\; \mathsf{wlp}[c]([G]\cdot g)\,.\end{aligned}$$ [13] △

As so defined, proving a loop i.i.d. might seem somewhat involved. However, we can do this by means of a simple data flow analysis. It is not hard to see that a loop is i.i.d. whenever there is no data flow across its iterations. In program c_{die}, this is a requirement one can readily check: we see that whenever a variable is set, its value is never read in a subsequent iteration.

The benefit of Definition 7.3 based on the finite approximations of a loop is that it immediately yields the following characterisation of the semantics for the entire loop, which will lie at the heart of the correctness proof of the proposed transformation.

Lemma 7.4 (w(l)p of i.i.d. loops). *Let $c \in \mathsf{cpGCL}$ and let $\mathsf{repeat}\,\{c\}\,(G)$ be an i.i.d. loop with $\mathsf{wp}[c]([\neg G])(s) < 1$ for every $s \in \mathbb{S}$. Then for all $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leq 1}$,*

$$\mathsf{wp}[\mathsf{repeat}\,\{c\}\,(G)](f) \;=\; \frac{\mathsf{wp}[c]([G]\cdot f)}{\mathbf{1}-\mathsf{wp}[c]([\neg G])}\,, \quad \textit{and}$$
$$\mathsf{wlp}[\mathsf{repeat}\,\{c\}\,(G)](g) \;=\; \frac{\mathsf{wlp}[c]([G]\cdot g)}{\mathbf{1}-\mathsf{wp}[c]([\neg G])}\,.$$

Proof. We prove only the first equation, the second equation follows by a similar reasoning. Using a standard (continuity) argument we can show that

$$\mathsf{wp}[\mathsf{repeat}\,\{c\}\,(G)](f) \;=\; \sup_n\; \mathsf{wp}[\mathsf{repeat}_n\,\{c\}\,(G)](f)\,,$$

and a simple induction over n gives

$$\mathsf{wp}[\mathsf{repeat}_n\,\{c\}\,(G)](f) \;=\; \mathsf{wp}[c]([G]\cdot f)\cdot \sum_{i=0}^{n-1} \mathsf{wp}[c]([\neg G])^i\,.$$

To conclude we rely on the closed form $\frac{1}{1-p}$ of the geometric series $\sum_{i=0}^{\infty} p^i$ for $|p|<1$. □

Using this result, we can readily prove the proposed transformation correct.

Theorem 7.5 (Correctness of observations for i.i.d. loops). *Let $c \in \mathsf{cpGCL}$ and let $\mathsf{repeat}\,\{c\}\,(G)$ be an i.i.d. loop with $\mathsf{wp}[c]([\neg G])(s) < 1$ for every $s \in \mathbb{S}$. Then for all $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leq 1}$,*

$$\mathsf{cwp}[\mathsf{repeat}\,\{c\}\,(G)](f) \;=\; \mathsf{cwp}[c;\,\mathsf{observe}\,(G)](f)\,, \quad \textit{and}$$
$$\mathsf{cwlp}[\mathsf{repeat}\,\{c\}\,(G)](g) \;=\; \mathsf{cwlp}[c;\,\mathsf{observe}\,(G)](g)\,. \qquad \triangle$$

Proof. Again, we consider only the first equation. By the alternative characterisation of transformers cw(l)p (Equation (3)) and Lemma 7.4, we have

$$\mathsf{cwp}[\mathsf{repeat}\,\{c\}\,(G)](f) \;=\; \frac{\mathsf{wp}[\mathsf{repeat}\,\{c\}\,(G)](f)}{\mathsf{wlp}[\mathsf{repeat}\,\{c\}\,(G)](\mathbf{1})} \;=\; \frac{\frac{\mathsf{wp}[c]([G]\cdot f)}{1-\mathsf{wp}[c]([\neg G])}}{\frac{\mathsf{wlp}[c]([G])}{1-\mathsf{wp}[c]([\neg G])}}$$
$$=\; \frac{\mathsf{wp}[c]([G]\cdot f)}{\mathsf{wlp}[c]([G])} \;=\; \frac{\mathsf{wp}[c;\,\mathsf{observe}\,(G)](f)}{\mathsf{wlp}[c;\,\mathsf{observe}\,(G)](\mathbf{1})} \;=\; \mathsf{cwp}[c;\,\mathsf{observe}\,(G)](f)\,. \quad □$$

[13] Observe that in the second equation we keep $\mathsf{wp}[c]([\neg G])$ instead of using the liberal version $\mathsf{wlp}[c]([\neg G])$. This is because $\mathsf{wp}[c](f_1+f_2) = \mathsf{wp}[c](f_1)+\mathsf{wp}[c](f_2)$, while $\mathsf{wlp}[c](g_1+g_2) = \mathsf{wp}[c](g_1)+\mathsf{wlp}[c](g_2)$ (see Lemma A.3 in Appendix A.3).

8 RELATED WORK

Weakest–precondition semantics of probabilistic programs. The foundations of semantics of probabilistic programming languages goes back to the seminal work [37]. Kozen provided semantics for probabilistic programs and developed the probabilistic propositional dynamic logic [38] to reason about such programs. Whereas his work focused on fully probabilistic programs, [40] extended this with (demonic) non–determinism. Their transformers wp[·] and wlp[·] are respectively denoted by $\langle\cdot\rangle$ and $[\cdot]$ in Kozen's work, and represent (dual) modalities of probabilistic propositional dynamic logic. Probabilistic weakest pre–condition semantics has a corresponding backward abstraction in the setting of abstract interpretation [17]. These notions are backward compatible with Dijkstra's notions of weakest (liberal) pre–conditions; that is to say, for deterministic programs Kozen's/McIver and Morgan's semantics coincide with that of Dijkstra. This paper can be seen as an extension of these lines of work with the notion of conditioning. In particular, Theorem 4.7 shows that our conditional weakest pre–condition semantics conservatively extends [38, 40]. Mechanisations of weakest pre–condition semantics using theorem provers have been conducted in HOL [29], Isabelle [16], and Coq [3]. Extensions of the wp–approach with conditioning have, to our knowledge, not yet been reported. However, a semi–automation based on bounded model checking of the operational semantics developed in this paper is presented in [30].

Relating different semantics. Relating several semantics of probabilistic programs is not new. [37] provided an interpretation in terms of functions on measurable spaces and as operators on a Banach space of measures and showed their correspondence. The correspondence between the weakest–precondition semantics of [40] and an intuitive operational semantics in terms of Markov decision processes has been reported by [25]. Similar work for Dijkstra's guarded command language was published in [39]. Theorem 5.7 can be considered as an extension of these latter results to probabilistic programs with conditioning.

Non–termination and non–determinism. The main difference with existing semantics of modern probabilistic programming languages such as R2 [28, 42] is the explicit treatment of possible diverging programs in our setting. In fact, several recent works on probabilistic programming [9, 11, 48] assume programs to be almost–surely or even always terminating. For certain applications, the restriction to terminating programs is understandable; for a semantics of a general–purpose language, we believe that possible divergence needs to be treated.

Our operational semantics deals with conditioning and non–determinism. It was shown that conditioning and non–determinism cannot both be covered by an inductive wp–semantics. This result is related to the fact that for conditional probabilities in Markov decision processes, memoryless schedulers (schedulers that on every visit to a state always take the same decision) are insufficient. Instead, history–dependent schedulers are needed, see [2, 6]. In fact, [52] already noticed the difficulties that arise when trying to integrate non–determinism and probabilities, even in the absence of conditioning. Non–determinism in probabilistic programs has been studied extensively by [40]; current practical programming languages such as R2, webPPL and so forth do not incorporate this. We believe that non–determinism is an essential feature for probabilistic programs and is not just of theoretical interest. For instance, abstraction of program variables typically gives rise to non–determinism. Capturing non–determinism, conditioning, and probabilistic choice (or sampling) in a single semantic framework enables the formal reasoning about such abstract probabilistic programs. In addition, it provides a stepping–stone towards reasoning about concurrent programs where a viable approach is to treat concurrency by interleaving (i.e., non–determinism). The paper [24] mentions the treatment of non–determinism as a challenging problem in probabilistic programming. This paper only considered demonic non–determinism. An operational semantics for a probabilistic programming language that contains both angelic and demonic non–determinism has been given in [14]. They consider stochastic two–player games as operational model, one type of player per form of non–determinism, but do not consider conditioning.

Conditioning. One of the main motivations behind modern probabilistic programming languages is the ability to condition the program runs on certain events, a feature that is at the heart of Bayesian networks. There are several syntactic ways in which this can be incorporated. [26] considered conditioning in a probabilistic constraint programming language with recursion and showed how conditional probabilities in their setting can be computed. (Computing conditional probabilities in general is undecidable as shown in [1].) In this paper, we have adopted the `observe` statements from [9] that nowadays have been adopted by various languages. The `observe` statement is related to assertions. Both `observe` (G) and `assert` (G) block all program executions violating G. However, `observe` (G) normalises the probability of the unblocked executions, while `assert` (G) does not, yielding a sub–distribution of total mass possibly less than one. `assert` statements correspond to the tests in the probabilistic propositional dynamic logic [38]. An alternative—quantitative—interpretation of `assert` statements is also studied in [48]. There, assertions are accompanied by a confidence value c and a probability value p meaning that with confidence c, the assertion holds with probability (at least) p. Assertions in probabilistic programs have also been treated in [12] where the analysis takes places using martingale theory. [9] consider conditioning in the setting of functional languages, and base their semantics on monads. Although their semantics covers conditioning on zero–probability events, unbounded loops are not considered. [42] and [9] consider `observe` statements for certainly terminating programs. Our `wp`–semantics coincides for terminating programs; we have discussed in detail at the end of Section 4 that adopting the *R2* semantics to possibly diverging programs leads somewhat counterintuitive results.

Program transformations. Most program transformations for probabilistic programs, such as slicing [28] aim to accelerate the Markov Chain Monte Carlo analysis. The transformations in this paper aim at treating conditioning. Our program transformation to “hoist” the `observe` statements through the program while updating the probabilistic choices is similar in spirit to [42]. As we use probabilistic choices and not random assignments we are able to completely remove conditioning from a program. In addition, as our semantics covers diverging programs as well, our transformation is applicable to non–terminating programs. The program transformation that replaces an `observe` statement by a loop is in fact a direct application of the principle of the rejection sampling method to conditional distributions. This has also been studied by, e.g., [49, §9] under the name of “generate and test” paradigm. As rejection sampling is the *de facto* semantics for inference on most practical probabilistic programming languages, this connection shows that our `wp`–semantics is an alternative to this. The idea to rerun a program until all observations are passed is used by [6] to automate the verification of conditioned temporal logic formulas in Markov models. Our final transformation to replace a loop by an `observe` statement has a strong resemblance with observations made in some textbooks on randomised algorithms; e.g., Theorem 7.5 that states the correctness of our transformation corresponds to [49, Theorem 9.3.(iii)].

9 CONCLUSION AND FUTURE WORK

This paper presented an in–depth study of the notion of conditioning in a simple imperative probabilistic programming language. Both a weakest–precondition and an operational semantics have been provided. Their relation has been established. The key is to consider the weakest–precondition semantics as a pair in which the probability to diverge or to violate one (or more) observations in the program is kept separately. This allows for treating possibly diverging programs, and conditioning on zero–probability events. It was shown that incorporating non–determinism in the inductive weakest–preconditioning setting is impossible. This raises the question how to deal with the combination of non–determinism and conditioning in a wp–style framework. The semantics have been used to prove the correctness of three program transformations, two of which remove conditioning, while one replaces a loop by an `observe`. An extension of both our semantics to recursive probabilistic programs with conditioning can be readily obtained based on the recent work [43]. Issue for future work include the treatment of continuous distributions, and the (semi–)automated synthesis of loop invariants. In particular, it would be interesting to investigate to what extent existing techniques for loop–invariant synthesis in probabilistic programs [7, 13, 35] can be lifted to the setting with conditioning. For continuous distributions, the operational semantics is no longer a Markov chain, but rather a stochastic relation. In addition, a wp–semantics for continuous distributions requires measure theory; recent work in that direction has been reported in [50]. We also plan to investigate the usage of our weakest–precondition framework to reason about entropy and secrecy where conditioning plays a crucial role [21].

APPENDIX

A.1 Proof of Lemma 4.6: Decoupling of $\underline{\mathsf{cw}}(\underline{\mathsf{l}})\mathsf{p}$

Lemma 4.6. *For $c \in \mathsf{cpGCL}$, $f \in \mathbb{E}$, and $g, g' \in \mathbb{E}_{\leq 1}$,*

$$\underline{\mathsf{cw}}\mathsf{p}[c](f, g) \;=\; \big(\mathsf{wp}[c](f),\, \mathsf{wlp}[c](g)\big) \qquad \textit{and} \qquad \underline{\mathsf{cwl}}\mathsf{p}[c](g, g') \;=\; \big(\mathsf{wlp}[c](g),\, \mathsf{wlp}[c](g')\big)\;.$$

Proof. By induction on the structure of c. Except for `while`–loops, the proof for all other program constructs is rather straightforward. For $c = \texttt{while}\,(G)\,\{c'\}$ we have

$$\begin{aligned}
&\underline{\mathsf{cw}}\mathsf{p}[\texttt{while}\,(G)\,\{c'\}](f,\, g)\\
&\quad= \mathit{lfp}_{\preceq,\succeq}(X_1, X_2)\boldsymbol{\cdot}\; [G]\cdot \underline{\mathsf{cw}}\mathsf{p}[c'](X_1,\, X_2) + [\neg G]\cdot (f, g)\\
&\quad= \mathit{lfp}_{\preceq,\succeq}(X_1, X_2)\boldsymbol{\cdot}\; [G]\cdot \big(\mathsf{wp}[c'](X_1),\, \mathsf{wlp}[c'](X_2)\big) + [\neg G]\cdot (f, g) && \text{(I.H. on } c'\text{)}\\
&\quad= \mathit{lfp}_{\preceq,\succeq}(X_1, X_2)\boldsymbol{\cdot}\; \underbrace{\big([G]\cdot \mathsf{wp}[c'](X_1) + [\neg G]\cdot f,\; [G]\cdot \mathsf{wlp}[c'](X_2) + [\neg G]\cdot g\big)}_{H(X_1, X_2)}
\end{aligned}$$

Now let H_1 (resp. H_2) be the first (resp. second) projection of H. Since the value of $H_1(X_1,\, X_2)$ (resp. $H_2(X_1,\, X_2)$) does not depend on X_2 (resp. X_1) and

$$\begin{aligned}
H_1(X_1,\, \_) &= [G]\cdot \mathsf{wp}[c'](X_1) + [\neg G]\cdot f\\
H_2(\_,\, X_2) &= [G]\cdot \mathsf{wlp}[c'](X_2) + [\neg G]\cdot g\,,
\end{aligned}$$

we can derive the continuity of both projections from the continuity of wp and wlp (Lemma A.1). Since H_1 and H_2 are continuous, Bekić's Theorem [8] says that the least fixed point of H is given by $\big(\widehat{X_1},\, \widehat{X_2}\big)$, where

$$\begin{aligned}
\widehat{X_1} &= \mathit{lfp}_{\preceq}\, X_1\boldsymbol{\cdot}\, H_1\big(X_1,\, \mathit{lfp}_{\succeq}\, X_2\boldsymbol{\cdot}\, H_2(X_1,\, X_2)\big)\\
&= \mathit{lfp}_{\preceq}\, X_1\boldsymbol{\cdot}\, H_1\big(X_1,\, \_\big)\\
&= \mathit{lfp}_{\preceq}\, X_1\boldsymbol{\cdot}\, [G]\cdot \mathsf{wp}[c'](X_1) + [\neg G]\cdot f\\
&= \mathsf{wp}[\texttt{while}\,(G)\,\{c'\}](f)
\end{aligned}$$

and

$$\begin{aligned}
\widehat{X_2} &= \mathit{lfp}_{\succeq}\, X_2\boldsymbol{\cdot}\, H_2\big(\,\mathit{lfp}_{\preceq}\, X_1\boldsymbol{\cdot}\, H_1(X_1,\, X_2),\, X_2\big)\\
&= \mathit{lfp}_{\succeq}\, X_2\boldsymbol{\cdot}\, H_2\big(\_,\, X_2\big)
\end{aligned}$$

$$
\begin{aligned}
&= \mathit{lfp}_{\geq}\, X_2\text{.}\ [G] \cdot \mathsf{wlp}[c'](X_2) + [\neg G] \cdot g \\
&= \mathit{gfp}_{\leq}\, X_2\text{.}\ [G] \cdot \mathsf{wlp}[c'](X_2) + [\neg G] \cdot g \\
&= \mathsf{wlp}[\mathtt{while}\,(G)\,\{c'\}](g)\ .
\end{aligned}
$$

This concludes the proof since, overall, we obtain

$$
\underline{\mathsf{cw}}\mathsf{p}[\mathtt{while}\,(G)\,\{c'\}](f, g) \;=\; \mathit{lfp}_{\leq,\geq}(H) \;=\; \big(\mathsf{wp}[\mathtt{while}\,(G)\,\{c'\}](f), \mathsf{wlp}[\mathtt{while}\,(G)\,\{c'\}](g)\big)\ .
$$

□

A.2 Continuity of w(l)p and $\underline{\text{cw}}(\underline{\text{l}})\text{p}$

Lemma A.1 (Continuity of w(l)p). *For every program $c \in$ cpGCL, the expectation transformers $\mathsf{wp}[c]\colon \mathbb{E} \to \mathbb{E}$ and $\mathsf{wlp}[c]\colon \mathbb{E}_{\leq 1} \to \mathbb{E}_{\leq 1}$ are continuous mappings over $(\mathbb{E}, \preceq)$ and $(\mathbb{E}_{\leq 1}, \succeq)$, respectively.*

Proof. Let $f_1 \preceq f_2 \preceq \ldots$ and $g_1 \succeq g_2 \succeq \ldots$ be two ω-chains in $\mathbb{E}$ and $\mathbb{E}_{\leq 1}$, respectively. We have to show that

$$
\sup_n \mathsf{wp}[c](f_n) \;=\; \mathsf{wp}[c](\sup_n f_n) \qquad \text{and} \qquad \inf_n \mathsf{wlp}[c](g_n) \;=\; \mathsf{wlp}[c](\inf_n g_n)\ .
$$

We proceed by induction on the structure of c. For $c = \mathtt{observe}\ (G)$, the statement is immediate since

$$
\sup_n \mathsf{wp}[\mathtt{observe}\ (G)](f_n) \;=\; \sup_n [G] \cdot f_n \;=\; [G] \cdot \sup_n \cdot f_n \;=\; \mathsf{wp}[\mathtt{observe}\ (G)](\sup_n f_n)\ ,
$$

and likewise for $\inf_n \mathsf{wlp}[\mathtt{observe}\ (G)](g_n) = \mathsf{wlp}[\mathtt{observe}\ (G)](\inf_n g_n)$. The remaining program constructs are covered in [25]. □

Lemma A.2 (Continuity of $\underline{\text{cw}}(\underline{\text{l}})\text{p}$). *For every program $c \in$ cpGCL, the expectation transformers $\underline{\mathsf{cw}}\mathsf{p}[c]\colon \mathbb{E} \times \mathbb{E}_{\leq 1} \to \mathbb{E} \times \mathbb{E}_{\leq 1}$ and $\underline{\mathsf{cwl}}\mathsf{p}[c]\colon \mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1} \to \mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1}$ are continuous mappings over $(\mathbb{E} \times \mathbb{E}_{\leq 1}, \preceq \times \succeq)$ and $(\mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1}, \succeq \times \succeq)$, respectively.*

Proof. Immediate from Lemmas A.1 and 4.6, since continuous functions are closed under products. □

A.3 Duality between cwp and cwlp

Lemma A.3. *For every program $c \in$ cpGCL and expectations $g, g' \in \mathbb{E}_{\leq 1}$ such that $g + g' \preceq \mathbf{1}$,*

$$
\mathsf{wlp}[c](g) + \mathsf{wp}[c](g') \;=\; \mathsf{wlp}[c](g + g')\ .
$$

Proof. By induction on the structure of c. We sketch the cases of sequential composition and while–loops since the remaining cases are immediate from the definition of w(l)p. For $c = c_1; c_2$ we have

$$
\begin{aligned}
&\mathsf{wlp}[c_1; c_2](g) + \mathsf{wp}[c_1; c_2](g') \\
&\quad= \mathsf{wlp}[c_1]\big(\mathsf{wlp}[c_2](g)\big) + \mathsf{wp}[c_1]\big(\mathsf{wp}[c_2](g')\big) && \text{(def. wlp, wp)} \\
&\quad= \mathsf{wlp}[c_1]\big(\mathsf{wlp}[c_2](g) + \mathsf{wp}[c_2](g')\big) && \text{(I.H. on } c_1\text{)} \\
&\quad= \mathsf{wlp}[c_1]\big(\mathsf{wlp}[c_2](g{+}g')\big) && \text{(I.H. on } c_2\text{)} \\
&\quad= \mathsf{wlp}[c_1; c_2](g{+}g') && \text{(def. wlp)}
\end{aligned}
$$

For $c = \mathtt{while}\,(G)\,\{c'\}$, the statement reduces to

$$
\mathit{gfp}_{\preceq}(\mathcal{F}\ell_g) + \mathit{lfp}_{\preceq}(\mathcal{F}_{g'}) \;=\; \mathit{gfp}_{\preceq}(\mathcal{F}\ell_{g+g'})\ ,
$$

where $\mathcal{F}\ell_h(f) = [G]\cdot \mathsf{wlp}[c'](f) + [\neg G]\cdot h$ and $\mathcal{F}_h(f) = [G]\cdot \mathsf{wp}[c'](f) + [\neg G]\cdot h$. Using the same argument (and notation) as in the proof of Theorem 4.5, we can show that the above equation is equivalent to $\lim_{n\to\infty} \mathcal{F}\ell_g^n(\mathbf{1}) + \lim_{n\to\infty} \mathcal{F}_{g'}^n(\mathbf{0}) \;=\; \lim_{n\to\infty} \mathcal{F}\ell_{g+g'}^n(\mathbf{1})$, which, in turn, follows from the statement

$$\forall n\text{.}\;\; \mathcal{F}\ell_g^n(\mathbf{1}) + \mathcal{F}_{g'}^n(\mathbf{0}) \;=\; \mathcal{F}\ell_{g+g'}^n(\mathbf{1})\;.$$

We proceed by induction on n. The base case reduces to $\mathbf{1} + \mathbf{0} = \mathbf{1}$. For the inductive case we reason as follows:

$$\begin{aligned}
&\mathcal{F}\ell_g^{n+1}(\mathbf{1}) \;+\; \mathcal{F}_{g'}^{n+1}(\mathbf{0}) \\
&\quad= [G]\cdot \mathsf{wlp}[c']\big(\mathcal{F}\ell_g^n(\mathbf{1})\big) + [\neg G]\cdot g \;+\; [G]\cdot \mathsf{wp}[c']\big(\mathcal{F}_{g'}^n(\mathbf{0})\big) + [\neg G]\cdot g' && (\text{def. } \mathcal{F}\ell_g, \mathcal{F}_{g'})\\
&\quad= [G]\cdot \mathsf{wlp}[c']\big(\mathcal{F}\ell_g^n(\mathbf{1}) + \mathcal{F}_{g'}^n(\mathbf{0})\big) \;+\; [\neg G]\cdot (g{+}g') && (\text{I.H. on } c')\\
&\quad= [G]\cdot \mathsf{wlp}[c']\big(\mathcal{F}\ell_{g+g'}^n(\mathbf{1})\big) \;+\; [\neg G]\cdot (g{+}g') && (\text{I.H. on } n)\\
&\quad= \mathcal{F}\ell_{g+g'}^{n+1}(\mathbf{1}) && (\text{def. } \mathcal{F}\ell_{g+g'})
\end{aligned}$$

□

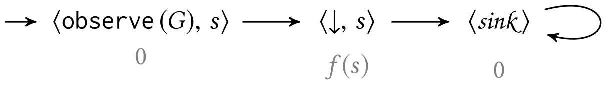
A.4 Proof of Lemma 5.5

Lemma 4.6. *For program $c \in$ cpGCL, state $s \in \mathbb{S}$ and expectations $f \in \mathbb{E}$, $g \in \mathbb{E}_{\leq 1}$,*

$$\mathsf{ER}^{\mathcal{R}_s^f[\![c]\!]}\,(\Diamond\, \mathit{sink}) \;=\; \mathsf{wp}[c](f)(s)\;,\;\; \textit{and} \tag{8}$$

$$\mathsf{LER}^{\mathcal{R}_s^g[\![c]\!]}\,(\Diamond\, \mathit{sink}) \;=\; \mathsf{wlp}[c](g)(s)\;. \tag{9}$$

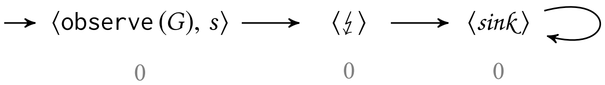
Proof. We begin with Equation (8). The proof proceeds by induction on the structure of c. Except for the case of observations, the proof argument for all other program constructs follows the same idea as employed in [25, Theorem 23]. For $c = \mathsf{observe}\,(G)$, we distinguish two cases. In Case 1 we have $s \models G$, the OMRC $\mathcal{R}_s^f[\![\mathsf{observe}\,(G)]\!]$ is[14]



and $\Diamond\, \mathit{sink} = \{\hat{\pi}_1\}$ with $\hat{\pi}_1 = \langle \mathsf{observe}\,(G),\, s\rangle \to \langle {\downarrow},\, s\rangle \to \langle \mathit{sink}\rangle$. Then,

$$\begin{aligned}
\mathsf{ER}^{\mathcal{R}_s^f[\![\mathsf{observe}\,(G)]\!]}\,(\Diamond\, \mathit{sink}) \;&=\; \sum_{\hat{\pi}\in\{\hat{\pi}_1\}} \mathsf{Pr}(\hat{\pi})\cdot r(\hat{\pi})\\
&=\; 1\cdot f(s) \;=\; [G](s)\cdot f(s) \;=\; \mathsf{wp}[\mathsf{observe}\,(G)](f)(s)\;.
\end{aligned}$$

In Case 2 we have $s \not\models G$, the OMRC $\mathcal{R}_s^f[\![\mathsf{observe}\,(G)]\!]$ is



and $\Diamond\, \mathit{sink} = \{\hat{\pi}_1\}$ with $\hat{\pi}_1 = \langle \mathsf{observe}\,(G),\, s\rangle \to \langle \lightning \rangle \to \langle \mathit{sink}\rangle$. Then,

$$\begin{aligned}
\mathsf{ER}^{\mathcal{R}_s^f[\![\mathsf{observe}\,(G)]\!]}\,(\Diamond\, \mathit{sink}) \;&=\; \sum_{\hat{\pi}\in\{\hat{\pi}_1\}} \mathsf{Pr}(\hat{\pi})\cdot r(\hat{\pi})\\
&=\; 1\cdot 0 \;=\; [G](s)\cdot f(s) \;=\; \mathsf{wp}[\mathsf{observe}\,(G)](f)(s)\;.
\end{aligned}$$

[14] If transitions have probability 1, we omit this in our figures. Moreover, all states—with the exception of ⟨sink⟩—are left out if they are not reachable from the initial state.

The case of loops is not thoroughly treated in [25] as the authors do not argue why the fact that Equation (8) holds for the finite unrollings of a loop implies that it also holds for the entire loop. For the sake of completeness, we provide herein a full proof argument. Assume $c = \texttt{while}\,(G)\,\{c'\}$. Since transformer wp is continuous, its action on a loop coincides with the limit of its action on the finite unrollings (see Section 4.4, Page 19), i.e.

$$\mathsf{wp}[\texttt{while}\,(G)\,\{c'\}](f) \;=\; \sup_n\, \mathsf{wp}[\texttt{while}_n\,(G)\,\{c'\}](f)\;.$$

Using the inductive hypothesis on c', we can also establish by induction on n that

$$\forall n\textbf{.}\;\mathsf{wp}[\texttt{while}_n\,(G)\,\{c'\}](f) \;=\; \mathsf{ER}^{\mathcal{R}_s^f[\![\texttt{while}_n\,(G)\,\{c'\}]\!]}\,(\Diamond\, sink)\;.$$

To conclude, we are only left to show that

$$\sup_n\,\mathsf{ER}^{\mathcal{R}_s^f[\![\texttt{while}_n\,(G)\,\{c'\}]\!]}\,(\Diamond\, sink) \;=\; \mathsf{ER}^{\mathcal{R}_s^f[\![\texttt{while}\,(G)\,\{c'\}]\!]}\,(\Diamond\, sink)\;.$$

Observe that every path in the OMRC $\mathcal{R}_s^f[\![\texttt{while}_n\,(G)\,\{c'\}]\!]$ either terminates properly or is prematurely aborted (yielding 0 reward) because it reaches the bound of n iterations. But the OMRC $\mathcal{R}_s^f[\![\texttt{while}\,(G)\,\{c'\}]\!]$ for the unbounded loop does not prematurely abort any execution. Therefore, the left–hand side is upper bounded by the right–hand side. To prove the reverse inequality, observe that paths from $\mathcal{R}_s^f[\![\texttt{while}\,(G)\,\{c'\}]\!]$ that collect positive reward are necessarily finite. Therefore, for each of them there must exist some $n \in \mathbb{N}$ such that $\mathcal{R}_s^f[\![\texttt{while}_n\,(G)\,\{c'\}]\!]$ includes this path. By taking the supremum of these n's, we include in the left-hand-side every path from $\mathcal{R}_s^f[\![\texttt{while}\,(G)\,\{c'\}]\!]$ that collects positive reward.

This concludes the proof of Equation (8). The proof of Equation (9) also goes by induction on the structure of c and expect for the case of observations, whose proof argument is identical as for Equation (8), all the remaining cases follow the same ideas as in [25, Theorem 23]. □

A.5 Correctness of Observation–Hoisting Transformation

Lemma A.4. *Let $c \in$ cpGCL. Then for all expectations $f \in \mathbb{E}$ and $g, h \in \mathbb{E}_{\leq 1}$,*

$$\hat{h}\cdot\mathsf{wp}[\hat{c}](f) = \mathsf{wp}[c](h\cdot f)\ (10) \qquad \hat{h}\cdot\mathsf{wlp}[\hat{c}](g) = \mathsf{wlp}[c](h\cdot g)\ (11) \qquad \hat{h} = \mathsf{wlp}[c](h)\ (12)$$

where $\left(\hat{c}, \hat{h}\right) = \mathcal{T}(c,\ h)$.

Proof. We prove only Equation (10) and (12) since Equation (11) follows a reasoning similar to that of Equation (10). The proof proceeds by induction on the structure of c. We consider only the cases of sequential composition, probabilistic choice and while–loops since the other cases follow from the definition of w(l)p and elementary algebraic steps. We refer to the inductive hypothesis about (10) (resp. (12)) as IH$_{10}$ (resp. IH$_{12}$).

- *The sequential composition* $c_1; c_2$. Let $\left(\hat{c}_2, \hat{h}_2\right) = \mathcal{T}(c_2, h)$ and $\left(\hat{c}_1, \hat{h}_1\right) = \mathcal{T}(c_1, \hat{h}_2)$. By definition we have $\mathcal{T}(c_1; c_2, h) = \left(\hat{c}_1; \hat{c}_2, \hat{h}_1\right)$. Then

$$\begin{aligned}
\hat{h}_1\cdot\mathsf{wp}[\hat{c}_1;\hat{c}_2](f) &= \hat{h}_1\cdot\mathsf{wp}[\hat{c}_1]\,(\mathsf{wp}[\hat{c}_2](f)) && \text{(def. wp)}\\
&= \mathsf{wp}[c_1](\hat{h}_2\cdot\mathsf{wp}[\hat{c}_2](f)) && (\text{IH}_{10} \text{ on } c_1)\\
&= \mathsf{wp}[c_1](\mathsf{wp}[c_2](h\cdot f)) && (\text{IH}_{10} \text{ on } c_2)\\
&= \mathsf{wp}[c_1;c_2](h\cdot f) && \text{(def. wp)}
\end{aligned}$$

$$\begin{aligned}
\hat{h}_1 &= \mathsf{wlp}[c_1](\hat{h}_2) && (\text{IH}_{12} \text{ on } c_1)\\
&= \mathsf{wlp}[c_1](\mathsf{wlp}[c_2](h)) && (\text{IH}_{12} \text{ on } c_2)
\end{aligned}$$

$$= \ \mathsf{wlp}[c_1; c_2](h). \qquad \text{(def. wlp)}$$

■ *The probabilistic choice* $\{c_1\}\ [p]\ \{c_2\}$. Let $\left(\hat{c}_1,\ \hat{h}_1\right) = \mathcal{T}(c_1,\ h)$ and $\left(\hat{c}_2,\ \hat{h}_2\right) = \mathcal{T}(c_2,\ h)$. By definition we have

$$\mathcal{T}(\{c_1\}\ [\phi]\ \{c_2\}, h) \ = \ \left(\{\hat{c}_1\}\ \left[{}^{\phi\cdot\hat{h}_1}/{}_{\hat{h}}\right]\ \{\hat{c}_2\},\ \phi\cdot\hat{h}_1 + (\mathbf{1}{-}\phi)\cdot\hat{h}_2\right)$$

with $\hat{h} = \phi\cdot\hat{h}_1 + (\mathbf{1}{-}\phi)\cdot\hat{h}_2$. To prove Equation (10)

$$\hat{h}\cdot\mathsf{wp}[\{\hat{c}_1\}\ \left[{}^{\phi\cdot\hat{h}_1}/{}_{\hat{h}}\right]\ \{\hat{c}_2\}](f) \ = \ \mathsf{wp}[\{c_1\}\ [\phi]\ \{c_2\}](h\cdot f)\,,$$

we make a case distinction between those states that are mapped by $\hat{h}$ to a positive number and those that are mapped to zero. In the first case, i.e. if $\hat{h}(s) > 0$, we reason as follows:

$$\begin{aligned}
&\hat{h}(s)\cdot\mathsf{wp}[\{\hat{c}_1\}\ \left[{}^{\phi\cdot\hat{h}_1}/{}_{\hat{h}}\right]\ \{\hat{c}_2\}](f)(s) \\
&\quad = \ \hat{h}(s)\cdot\left(\tfrac{\phi\cdot\hat{h}_1}{\hat{h}}(s)\cdot\mathsf{wp}[\hat{c}_1](f)(s) \ + \ \tfrac{(\mathbf{1}-\phi)\cdot\hat{h}_2}{\hat{h}}(s)\cdot\mathsf{wp}[\hat{c}_2](f)(s)\right) && \text{(def. wp, algebra)} \\
&\quad = \ \phi(s)\cdot\hat{h}_1(s)\cdot\mathsf{wp}[\hat{c}_1](f)(s) \ + \ (\mathbf{1}{-}\phi)(s)\cdot\hat{h}_2(s)\cdot\mathsf{wp}[\hat{c}_2](f)(s) && \text{(algebra)} \\
&\quad = \ \phi(s)\cdot\mathsf{wp}[c_1](h\cdot f)(s) \ + \ (\mathbf{1}{-}\phi)(s)\cdot\mathsf{wp}[c_2](h\cdot f)(s) && (\text{IH}_{10}\text{ on } c_1, c_2) \\
&\quad = \ \mathsf{wp}[\{c_1\}\ [\phi]\ \{c_2\}](h\cdot f)(s) && \text{(algebra)}
\end{aligned}$$

In the second case, i.e. if $\hat{h}(s) = 0$, the claim holds because we will have $\mathsf{wp}[\{c_1\}\ [\phi]\ \{c_2\}](h{\cdot}f)(s) = 0$. To see this, note that if $\hat{h}(s) = 0$ then either $\phi(s) = 0 \wedge \hat{h}_2(s) = 0$ or $\phi(s) = 1 \wedge \hat{h}_1(s) = 0$. Now assume we are in the first case (an analogous argument works for the other case); using the IH_{10} over c_2 we conclude that

$$\mathsf{wp}[\{c_1\}\ [0]\ \{c_2\}](h\cdot f)(s) \ = \ \mathsf{wp}[c_2](h\cdot f)(s) \ = \ \hat{h}_2(s)\cdot\mathsf{wp}[c_2](f)(s) \ = \ 0\,.$$

To prove Equation (12) we apply the IH_{12} on c_1 and c_2:

$$\phi\cdot\hat{h}_1 + (\mathbf{1}{-}\phi)\cdot\hat{h}_2 \ = \ \phi\cdot\mathsf{wlp}[c_1](h) + (\mathbf{1}{-}\phi)\cdot\mathsf{wlp}[c_2](h) \ = \ \mathsf{wlp}[\{c_1\}\ [\phi]\ \{c_2\}](h)\,.$$

■ *The loop* $\mathtt{while}\,(G)\,\{c\}$. Let $\hat{h} = \mathit{gfp}(\mathcal{H})$ where $\mathcal{H}(X) = [G]\cdot\mathcal{T}_c(X) + [\neg G]\cdot h$ and $\mathcal{T}_c(\cdot)$ is a short–hand for $\pi_2 \circ T(c,\cdot)$. If we let $(\hat{c}, \theta) = \mathcal{T}(c, \hat{h})$, by definition of $\mathcal{T}$ we have

$$\mathcal{T}(\mathtt{while}\,(G)\,\{c\}, h) \ = \ (\mathtt{while}\,(G)\,\{\hat{c}\}, \hat{h})\,.$$

Equation (10) says that

$$\hat{h}\cdot\mathsf{wp}[\mathtt{while}\,(G)\,\{\hat{c}\}](f) \ = \ \mathsf{wp}[\mathtt{while}\,(G)\,\{c\}](h\cdot f)\,.$$

Now if we let $H(X) = [G]\cdot\mathsf{wp}[\hat{c}](X) + [\neg G]\cdot f$ and $I(X) = [G]\cdot\mathsf{wp}[c](X) + [\neg G]\cdot h\cdot f$, the equation can be rewritten as $\hat{h}\cdot\mathit{lfp}(H) = \mathit{lfp}(I)$ and a straightforward argument using the Kleene fixed point theorem (and the continuity of wp established in Lemma A.1) shows that it is entailed by $\forall n\text{.}\ \hat{h}\cdot H^n(\mathbf{0}) = I^n(\mathbf{0})$. We prove this statement by induction on n. The case $n = 0$ is trivial. For the inductive case we reason as follows:

$$\begin{aligned}
\hat{h}\cdot H^{n+1}(\mathbf{0}) \ &= \ \mathcal{H}(\hat{h})\cdot H^{n+1}(\mathbf{0}) && (\text{def. } \hat{h}) \\
&= \ ([G]\cdot\mathcal{T}_c(\hat{h}) + [\neg G]\cdot h)\cdot H^{n+1}(\mathbf{0}) && (\text{def. } \mathcal{H}) \\
&= \ ([G]\cdot\mathcal{T}_c(\hat{h}) + [\neg G]\cdot h)\cdot([G]\cdot\mathsf{wp}[\hat{c}](H^n(\mathbf{0})) + [\neg G]\cdot f) && (\text{def. } H) \\
&= \ [G]\cdot\mathcal{T}_c(\hat{h})\cdot\mathsf{wp}[\hat{c}](H^n(\mathbf{0})) \ + \ [\neg G]\cdot h\cdot f && \text{(algebra)} \\
&= \ [G]\cdot\theta\cdot\mathsf{wp}[\hat{c}](H^n(\mathbf{0})) \ + \ [\neg G]\cdot h\cdot f && (\text{def. } \theta) \\
&= \ [G]\cdot\mathsf{wp}[c](\hat{h}\cdot H^n(\mathbf{0})) \ + \ [\neg G]\cdot h\cdot f && (\text{IH}_{10}\text{ on } c)
\end{aligned}$$

$$
\begin{aligned}
&= I(\hat{h} \cdot H^n(\mathbf{0})) && \text{(def. } I\text{)} \\
&= I^{n+1}(\mathbf{0}) && \text{(IH on } n\text{)}
\end{aligned}
$$

We now turn to proving Equation (12)

$$\hat{h} \;=\; \mathsf{wlp}[\mathtt{while}\,(G)\,\{c\}](h)\;.$$

By letting $J(X) = [G] \cdot \mathsf{wlp}[c](X) + [\neg G] \cdot h$, the claim reduces to $\mathit{gfp}(\mathcal{H}) = \mathit{gfp}(J)$, which we prove showing that $\hat{h} = \mathit{gfp}(\mathcal{H})$ is a fixed point of J and $\mathit{gfp}(J)$ is a fixed point of $\mathcal{H}$. (These assertions basically imply that $\mathit{gfp}(\mathcal{H}) \geq \mathit{gfp}(J)$ and $\mathit{gfp}(J) \geq \mathit{gfp}(\mathcal{H})$, respectively.)

$$
\begin{aligned}
J(\hat{h}) &= [G] \cdot \mathsf{wlp}[c](\hat{h}) \;+\; [\neg G] \cdot h && \text{(def. } J\text{)} \\
&= [G] \cdot \theta \;+\; [\neg G] \cdot h && (\text{IH}_{12} \text{ on } c) \\
&= [G] \cdot \mathcal{T}_c(\hat{h}) \;+\; [\neg G] \cdot h && \text{(def. } \theta\text{)} \\
&= \mathcal{H}(\hat{h}) && \text{(def. } \mathcal{H}\text{)} \\
&= \hat{h} && \text{(def. } \hat{h}\text{)}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{H}(\mathit{gfp}(J)) &= [G] \cdot \mathcal{T}_c(\mathit{gfp}(J)) \;+\; [\neg G] \cdot h && \text{(def. } \mathcal{H}\text{)} \\
&= [G] \cdot \mathsf{wlp}[c](\mathit{gfp}(J)) \;+\; [\neg G] \cdot h && (\text{IH}_{12} \text{ on } c) \\
&= J(\mathit{gfp}(J)) && \text{(def. } J\text{)} \\
&= \mathit{gfp}(J) && \text{(def. } \mathit{gfp}(J)\text{)}
\end{aligned}
$$

□

A.6 Proof of Theorem 7.2

Theorem 7.2 (Correctness of simulation by RSM). *Let $c \in$ cpGCL be a feasible program from initial state $s \in \mathbb{S}$. Then for all $f \in \mathbb{E}$,*

$$\mathsf{cwp}[c](f) \;=\; \mathsf{wp}[\mathsf{rsm}[c]](f)\;.$$

Proof. It relies on the following observations:

(1) every path $\hat{\pi}$ of $\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]$ reaching $\langle \mathit{sink} \rangle$ with $r(\hat{\pi}) > 0$ is of the form $\hat{\pi}_{in} \circ \hat{\pi}_1^{\lightning} \circ \cdots \circ \hat{\pi}_m^{\lightning} \circ \hat{\pi}^{\checkmark}$ for some $m \in \mathbb{N}$ (possibly 0, meaning that $\hat{\pi} = \hat{\pi}_{in} \circ \hat{\pi}^{\checkmark}$), where $\hat{\pi}_{in}$ is the path fragment that accounts for the initialization of variables $s_1, \ldots, s_n$ (Line 1 in Figure 10b), $\hat{\pi}_i^{\lightning}$ represents an iteration of the loop in $\mathsf{rsm}[c]$ that fails to pass the (now gone) observations of c, and $\hat{\pi}^{\checkmark}$ an iteration that does pass the observations;

(2) every path of type $\hat{\pi}^{\checkmark}$ in $\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]$ corresponds to a path $\hat{\pi}^*$ of $\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]$ in $\Diamond\, \mathit{sink} \cap \neg\Diamond\, \neg \mathit{unblocked}$ (they have equal probabilities and cumulated rewards);

(3) for every m, $\mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat{\pi}_1^{\lightning} \circ \cdots \circ \hat{\pi}_m^{\lightning}) \;=\; \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\Diamond\, \neg \mathit{unblocked})^m$ since all the loop iterations of $\mathsf{rsm}[c]$ are independent (because the original program state s is restored at the beginning of each iteration);

(4) each path of $\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]$ in $\Diamond\, \mathit{sink}$ (resp. $\neg\Diamond\, \neg \mathit{unblocked}$) corresponds to a paths of $\mathfrak{R}_s^f[\![c]\!]$ in $\Diamond\, \mathit{sink}$ (resp. $\neg\Diamond\, \lightning$) (they have equal probabilities and cumulated rewards).

Given this, we reason as follows:

$$
\begin{aligned}
&\mathsf{wp}[\mathsf{rsm}[c]](f) \\
= \;&\{\text{Lemma 5.5}\}
\end{aligned}
$$

$$\mathsf{ER}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}\,(\lozenge \mathit{sink})$$

$$= \big\{\text{Definition of expected rewards}\big\}$$

$$\sum_{\hat\pi \text{ of } \mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!] \text{ in } \lozenge \mathit{sink}} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi)\cdot r^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi)$$

$$= \big\{\text{Observation (1)}\big\}$$

$$\sum_{m=0}^{\infty}\ \sum_{\substack{\hat\pi=\hat\pi_{in}\circ\hat\pi_1^{\lightning}\circ\cdots\circ\hat\pi_m^{\lightning}\circ\hat\pi^{\checkmark} \\ \text{of } \mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!] \text{ in } \lozenge \mathit{sink}}} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi)\cdot r^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi)$$

$$= \big\{\mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi) = \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi_1^{\lightning}\circ\cdots\circ\hat\pi_m^{\lightning})\cdot \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi^{\checkmark})$$
$$\text{and } r^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi) = r^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi^{\checkmark})\big\}$$

$$\sum_{m=0}^{\infty}\ \sum_{\substack{\hat\pi_{in}\circ\hat\pi_1^{\lightning}\circ\cdots\circ\hat\pi_m^{\lightning}\circ\hat\pi^{\checkmark} \\ \text{of } \mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!] \text{ in } \lozenge \mathit{sink}}} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi_1^{\lightning}\circ\cdots\circ\hat\pi_m^{\lightning})\cdot \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi^{\checkmark})\cdot r^{\mathfrak{R}_s^f[\![\mathsf{rsm}[c]]\!]}(\hat\pi^{\checkmark})$$

$$= \big\{\text{Observations (2) and (3)}\big\}$$

$$\sum_{m=0}^{\infty}\ \sum_{\substack{\hat\pi^{*} \text{ of } \mathfrak{R}_s^f[\![\mathcal{B}(c)]\!] \text{ in} \\ \lozenge \mathit{sink}\cap\neg\lozenge\neg \mathit{unblocked}}} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\lozenge\neg \mathit{unblocked})^m\cdot \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\hat\pi^{*})\cdot r^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\hat\pi^{*})$$

$$= \big\{\text{Algebra}\big\}$$

$$\left(\sum_{m=0}^{\infty} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\lozenge\neg \mathit{unblocked})^m\right)\cdot \sum_{\substack{\hat\pi^{*} \text{ of } \mathfrak{R}_s^f[\![\mathcal{B}(c)]\!] \text{ in} \\ \lozenge \mathit{sink}\cap\neg\lozenge\neg \mathit{unblocked}}} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\hat\pi^{*})\cdot r^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\hat\pi^{*})$$

$$= \Big\{\text{Closed form } \tfrac{1}{1-p} \text{ of the geometric series } \textstyle\sum_{i=0}^{\infty} p^i \text{ for } |p|<1\Big\}$$

$$\frac{\sum_{\substack{\hat\pi^{*} \text{ of } \mathfrak{R}_s^f[\![\mathcal{B}(c)]\!] \text{ in} \\ \lozenge \mathit{sink}\cap\neg\lozenge\neg \mathit{unblocked}}} \mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\hat\pi^{*})\cdot r^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\hat\pi^{*})}{1-\mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\lozenge\neg \mathit{unblocked})}$$

$$= \big\{\text{Definition of expected rewards, } \mathsf{Pr}^{\mathfrak{M}}(\neg\lozenge A) = 1-\mathsf{Pr}^{\mathfrak{M}}(\lozenge A)\big\}$$

$$\frac{\mathsf{ER}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}\,(\lozenge \mathit{sink}\cap\neg\lozenge\neg \mathit{unblocked})}{\mathsf{Pr}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}(\neg\lozenge\neg \mathit{unblocked})}$$

$$= \big\{\text{Definition of conditional expected rewards}\big\}$$

$$\mathsf{CER}^{\mathfrak{R}_s^f[\![\mathcal{B}(c)]\!]}\,(\lozenge \mathit{sink}\mid\neg\lozenge\neg \mathit{unblocked})$$

$$= \big\{\text{Observation (4)}\big\}$$

$$\mathsf{CER}^{\mathfrak{R}_s^f[\![c]\!]}\,(\lozenge \mathit{sink}\mid\neg\lozenge\lightning)$$

$$= \big\{\text{Theorem 5.7}\big\}$$

$$\mathsf{cwp}[c](f)\,. \qquad \square$$

ACKNOWLEDGMENTS

This work was supported by the Excellence Initiative of the German federal and state government, and the CDZ project CAP (GZ 1023). Moreover, we would like to thank Pedro D'Argenio (Universidad Nacional de Córdoba, Argentina) and Tahiry Rabehaja (Macquarie University, Australia) for the valuable discussions preceding this paper.

REFERENCES

[1] Nathanael Leedom Ackerman, Cameron E. Freer, and Daniel M. Roy. 2011. Noncomputable Conditional Distributions. In *Proc. of LICS*. IEEE Computer Society, 107–116.

[2] Miguel E. Andrés and Peter van Rossum. 2008. Conditional Probabilities over Probabilistic and Nondeterministic Systems. In *TACAS (LNCS)*, Vol. 4963. Springer, 157–172.

[3] Philippe Audebaud and Christine Paulin-Mohring. 2009. Proofs of randomized algorithms in Coq. *Sci. Comput. Program.* 74, 8 (2009), 568–589.

[4] Ralph-Johan Back and Joakim von Wright. 1998. *Refinement Calculus - A Systematic Introduction.* Springer.

[5] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking.* MIT Press.

[6] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Steffen Märcker. 2014. Computing Conditional Probabilities in Markovian Models Efficiently. In *TACAS (LNCS)*, Vol. 8413. Springer, 515–530.

[7] Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016. Synthesizing Probabilistic Invariants via Doob's Decomposition. In *CAV (LNCS)*, Vol. 9779. Springer, 43–61.

[8] Hans Bekić. 1984. Definable Operation in General Algebras, and the Theory of Automata and Flowcharts. In *Programming Languages and Their Definition.* Springer, 30–55.

[9] Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. 2011. Measure Transformer Semantics for Bayesian Machine Learning. In *ESOP (LNCS)*, Vol. 6602. Springer, 77–96.

[10] Göran Broström and Leif Nilsson. 2000. Acceptance–Rejection Sampling from the Conditional Distribution of Independent Discrete Random Variables, given their Sum. *Statistics: A Journal of Theoretical and Applied Statistics* 34, 3 (2000), 247–257.

[11] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. 2013. Verifying quantitative reliability for programs that execute on unreliable hardware. In *OOPSLA*. ACM Press, 33–52.

[12] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV (LNCS)*, Vol. 8044. Springer, 511–526.

[13] Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *SAS (LNCS)*, Vol. 8723. Springer, 85–100.

[14] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *POPL*. ACM Press, 327–342.

[15] Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. 2013. Bayesian inference using data flow analysis. In *ESEC/SIGSOFT FSE*. ACM Press, 92–102.

[16] David Cock. 2014. pGCL for Isabelle. *Archive of Formal Proofs* (2014).

[17] Patrick Cousot and Michael Monerau. 2012. Probabilistic Abstract Interpretation. In *ESOP (LNCS)*, Vol. 7211. Springer, 169–193.

[18] Jerry den Hartog and Erik P. de Vink. 2002. Verifying Probabilistic Programs Using a Hoare Like Logic. *Int. J. Found. Comput. Sci.* 13, 3 (2002), 315–340.

[19] Edsger W. Dijkstra. 1976. *A Discipline of Programming.* Prentice Hall.

[20] Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. 2015. ProbLog2: Probabilistic Logic Programming. In *ECML/PKDD (3) (LNCS)*, Vol. 9286. Springer, 312–315.

[21] Barbara Espinoza and Geoffrey Smith. 2013. Min-entropy as a resource. *Inf. Comput.* 226 (2013), 57–75.

[22] Noah D. Goodman and Andreas Stuhlmüller. 2014. *The Design and Implementation of Probabilistic Programming Languages.* (electronic).

[23] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio V. Russo, Johannes Borgström, and John Guiver. 2014. Tabular: a schema-driven probabilistic programming language. In *POPL*. ACM Press, 321–334.

[24] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *FOSE*. ACM Press, 167–181.

[25] Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2014. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Perform. Eval.* 73 (2014), 110–132.

[26] Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. 1999. Stochastic Processes as Concurrent Constraint Programs. In *POPL*. ACM Press, 189–202.

[27] Eric C. R. Hehner. 2011. A Probability Perspective. *Formal Aspects of Computing* 23, 4 (2011), 391–419.

[28] Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. 2014. Slicing Probabilistic Programs. In *PLDI*. ACM Press, 133–144.

[29] Joe Hurd, Annabelle McIver, and Carroll Morgan. 2005. Probabilistic guarded commands mechanized in *HOL*. *Theor. Comput. Sci.* 346, 1 (2005), 96–112.

[30] Nils Jansen, Christian Dehnert, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Lukas Westhofen. 2016. Bounded Model Checking for Probabilistic Programs. In *ATVA (LNCS)*. Springer, 68–85.

[31] Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Federico Olmedo, Friedrich Gretz, and Annabelle McIver. 2015. Conditioning in Probabilistic Programming. 319 (2015), 199–216. https://doi.org/10.1016/j.entcs.2015.12.013

[32] Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the Hardness of Almost-Sure Termination. In *Mathematical Foundations of Computer Science (MFCS), Part I (LNCS)*, Vol. 9234. Springer, 307–318.

[33] Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2017. A Weakest Pre-Expectation Semantics for Mixed-Sign Expectations. In *Proc. of LICS*. IEEE Computer Society, 1–12.

[34] Joost-Pieter Katoen, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, and Federico Olmedo. 2015. Understanding Probabilistic Programs. In *Correct System Design (LNCS)*, Vol. 9360. Springer, 15–32.

[35] Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. 2010. Linear-Invariant Generation for Probabilistic Programs. In *SAS (LNCS)*, Vol. 6337. Springer, 390–406.

[36] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2010. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design* 36, 3 (2010), 246–280.

[37] Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.* 22, 3 (1981), 328–350.

[38] Dexter Kozen. 1985. A probabilistic PDL. *J. Comput. Syst. Sci.* 30, 2 (1985), 162–178.

[39] Johan J. Lukkien. 1994. Operational Semantics and Generalized Weakest Preconditions. *Sci. Comput. Program.* 22, 1-2 (1994), 137–155.

[40] Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement And Proof For Probabilistic Systems.* Springer.

[41] Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (1996), 325–353.

[42] Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. 2014. R2: An Efficient MCMC Sampler for Probabilistic Programs. In *AAAI*. AAAI Press, 2476–2482.

[43] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Proc. of LICS*. ACM Press, 672–682.

[44] Brooks Paige and Frank Wood. 2014. A Compilation Target for Probabilistic Programming Languages. In *ICML (Proc. of JMLR)*, Vol. 32. JMLR.org, 1935–1943.

[45] Avi Pfeffer. 2016. *Practical Probabilistic Programming.* Manning Publications Co.

[46] Martin Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley and Sons.

[47] Christian Robert and George Casella. 2013. *Monte Carlo statistical methods.* Springer Science & Business Media.

[48] Adrian Sampson, Pavel Panchekha, Todd Mytkowicz, Kathryn S. McKinley, Dan Grossman, and Luis Ceze. 2014. Expressing and verifying probabilistic assertions. In *PLDI*. ACM Press, 14.

[49] Victor Shoup. 2009. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press.

[50] Sam Staton, Hongseok Yang, Frank Wood, Chris Heunen, and Ohad Kammar. 2016. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proc. of LICS*. ACM Press, 525–534.

[51] Henk Tijms. 2007. *Understanding Probability: Chance Rules in Everyday Life.* Cambridge University Press.

[52] Daniele Varacca and Glynn Winskel. 2006. Distributing probability over non-determinism. *Mathematical Structures in Computer Science* 16, 1 (2006), 87–113.

[53] Wolfgang Wechler. 1992. *Universal Algebra for Computer Scientists.* EATCS Monographs on Theoretical Computer Science, Vol. 25. Springer.

[54] Herbert Wiklicky. 2016. On Dynamical Probabilities, or: How to Learn to Shoot Straight. In *Proc. of COORDINATION (LNCS)*, Vol. 9686. Springer, 262–277.

Received February 2007; revised March 2009; accepted June 2009