Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 319 (2015) 199–216

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Conditioning in Probabilistic Programming

Nils Jansen Benjamin[1] Lucien Kaminski[1] Joost-Pieter Katoen[1] Federico Olmedo[1]

RWTH Aachen University, Aachen, Germany

Friedrich Gretz[2] Annabelle McIver[2]

Macquarie University, Sydney, Australia

Abstract

In this paper, we investigate the semantic intricacies of conditioning in probabilistic programming, a major feature, e.g., in machine learning. We provide a quantitative weakest pre–condition semantics. In contrast to all other approaches, non–termination is taken into account by our semantics. We also present an operational semantics in terms of Markov models and show that expected rewards coincide with quantitative pre–conditions. A program transformation that entirely eliminates conditioning from programs is given; the correctness is shown using our semantics. Finally, we show that an inductive semantics for conditioning in non–deterministic probabilistic programs cannot exist.

Keywords: Probabilistic Programming, Semantics, Conditional Probabilities, Program Transformation

1 Introduction

In recent years, interest in probabilistic programming has rapidly grown [9,11]. This is due to its wide applicability, for example in machine learning for describing distribution functions; Bayesian inference is pivotal in their analysis. It is used in security for describing both cryptographic constructions such as randomized encryption and experiments defining security properties [4]. Probabilistic programs, being extensions of familiar notions, render these fields accessible to programming communities. A rich palette of probabilistic programming languages exists including `Church` [8] as well as modern approaches like probabilistic `C` [23], `Tabular` [10] and `R2` [22].

Probabilistic programs are sequential programs having two main features: (1) the ability to *draw values at random* from probability distributions, and (2) the

⋆ This work was supported by the Excellence Initiative of the German federal and state government.

[1] Emails: {nils.jansen, benjamin.kaminski, katoen, federico.olmedo}@informatik.rwth-aachen.de

[2] Emails: annabelle.mciver@mq.edu.au, friedrich.gretz@students.mq.edu.au

http://dx.doi.org/10.1016/j.entcs.2015.12.013

ability to *condition the value of variables* in a program through so–called *observations*. The semantics of languages without conditioning is well–understood: In his seminal work, Kozen [19] considered denotational semantics for probabilistic programs without non–determinism or observations. One of these semantics—the expectation transformer semantics—was adopted by McIver and Morgan [21], who added support for non–determinism; a corresponding operational semantics is given in [13]. Other relevant works include probabilistic power–domains [17], semantics of constraint probabilistic programming languages [15,14], and semantics for stochastic λ–calculi [26].

Semantic intricacies. The difficulties that arise when program variables are conditioned through observations is less well–understood. This gap is filled in this paper. Previous work on semantics for programs with `observe` statements [22,16] do neither consider the possibility of *non–termination* nor the powerful feature of *non–determinism*. In contrast, we thoroughly study a more general setting which accounts for non–termination by means of a very simple yet powerful probabilistic programming language supporting non–determinism and observations. Let us first analyze a few examples illustrating the different problems. We start with the problem of non–termination; consider the two program snippets

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

The program on the left just assigns the value 2 to the program variable x, while the program on the right tosses a fair coin—which is modeled through a *probabilistic choice*—and depending on the outcome either performs the same variable assignment or diverges due to the `abort` instruction. The semantics given in [22,16] does not distinguish these two programs and is only sensible in the context of terminating programs. A programmer writing only terminating programs is already unrealistic in the non–probabilistic setting. Our semantics does not rely on the assumption that programs always terminate and is able to distinguish these two programs.

To discuss *observations*, consider the program snippet P_{obs_1}

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

which assigns zero to the variable x with probability $1/2$ while x is assigned one with the same likelihood, after which we condition to the outcome of x being one. The `observe` statement blocks all invalid runs violating its condition and renormalizes the probabilities of the remaining valid runs. This differs, e.g., from program annotations like (probabilistic) *assertions* [25] as we will see later. The interpretation of the program is the expected outcome conditioned on the valid runs. For P_{obs_1}, this yields the outcome $1 \cdot 1$—there is one valid run that happens with probability one, with x being one.

More involved problems arise when programs are *infeasible* meaning all runs are blocked. Consider a slight variant of the program above, called P_{obs_2}:

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

The left branch of the probabilistic choice is infeasible. Is this program equivalent to the sample program P_{obs_1}? It will turn out that this is the case, meaning that setting an infeasible program into context can render it feasible.

The situation becomes more complicated when considering loopy programs that may diverge. Consider the following two programs:

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

Program P_{div} diverges as x is set to one in every iteration. This yields a null *expected outcome*. Due to the conditioning on $x{=}1$, P_{andiv} has just a single (valid)—non–terminating—run, but this run *almost surely* never happens, i.e. it happens with probability zero. The conditional expected outcome of P_{andiv} can thus not be measured. Our semantics can distinguish these programs while programs with (probabilistic) assertions must be loop–free to avoid similar problems [25]. Other approaches insist on the absence of diverging loops [5]. Neither of these assumptions are realistic.

Non–determinism is a powerful means to deal with unknown information, as well as to specify abstractions in situations where implementation details are unimportant. This feature turns out to be intricate in combination with conditioning. [3] Consider the program P_{nondet}

$$\{\{x \coloneqq 5\} \,\square\, \{x \coloneqq 2\}\}\ [1/4]\ \{x \coloneqq 2\};\ \texttt{observe}\ (x{>}3),$$

where with probability $1/4$, x is set either to 5 or to 2 non–deterministically (denoted $\{x \coloneqq 5\} \,\square\, \{x \coloneqq 2\}$), while x is set to 2 with likelihood $3/4$. Resolving the non–deterministic choice in favor of setting x to five yields a conditional expectation of 5 for x, obtained as $5 \cdot 1/4$ rescaled over the single valid run of P_{nondet}. Taking the right branch however induces two invalid runs due to the violation of the condition $x{>}3$, yielding a non–measurable conditional outcome.

Contributions. The above issues—non–termination, loops, and non–determinism—indicate that conditioning in probabilistic programs is far from trivial. This paper presents a thorough semantic treatment of conditioning in a probabilistic extension of Dijkstra's guarded command language (known as pGCL [21]), an elementary though foundational language that includes (amongst others) parametric probabilistic choice. We take several semantic viewpoints.

We first provide a conditional version of a *weakest pre–condition* (wp) semantics à la [21]. This is typically defined inductively over the structure of the program. We show that combining both non–determinism and conditioning *cannot* be treated in this manner. To treat possibly non–terminating programs, due to e.g., diverging loops or abortion, this is complemented by a weakest *liberal* pre–condition (wlp) semantics. Moreover, our w(l)p semantics is backward compatible with the original

[3] As stated in [11], "representing and inferring sets of distributions is more complicated than dealing with a single distribution, and hence there are several technical challenges in adding non–determinism to probabilistic programs".

pGCL semantics for programs without conditioning; this *does not* apply to alternative approaches such as R2 [22].

Furthermore, Markov Decision Processes (MDPs) [24] are used as the basis for an *operational* semantics. This semantics is simple and elegant while covering *all* aforementioned phenomena, including non–determinism. We show that *conditional expected rewards* in the MDP–semantics correspond to (conditional) wp in the denotational semantics, extending a similar result for pGCL [13].

Finally, we present a program transformation which entirely eliminates conditioning from any program and prove its correctness using our semantics.

Summarized, after introducing pGCL (Section 2), we give a denotational semantics for fully probabilistic programs (Section 3). We provide the first operational semantics for imperative probabilistic programming languages with conditioning and both probabilistic and non–deterministic choice (Section 4). Our semantics enables us to prove the correctness of a program transformation that eliminates `observe` statements (Section 5). Finally, we show that it is not possible to provide an inductive semantics for programs that include both conditioning and non–determinism (Section 6).

An extended version of this paper including all proofs and further program transformations for eliminating `observe` statements is available in [12].

2 The Programming Language

In this section we briefly present the probabilistic programming language used for our development. The language is an extension of the *probabilistic guarded command language* (pGCL) of McIver and Morgan [21]. The original pGCL is given by syntax

$$\begin{aligned}\mathcal{P} \;::=\; & \texttt{skip} \mid \texttt{abort} \mid x \coloneqq E \mid \mathcal{P};\mathcal{P} \mid \texttt{ite}\,(G)\,\{\mathcal{P}\}\,\{\mathcal{P}\} \\ & \mid \{\mathcal{P}\}\;[p]\;\{\mathcal{P}\} \mid \{\mathcal{P}\} \,\square\, \{\mathcal{P}\} \mid \texttt{while}\,(G)\,\{\mathcal{P}\}\end{aligned}$$

and constitutes a plain extension of Dijkstra's guarded command language (GCL) [7] with a binary probabilistic choice operator. Here, x belongs to $\mathcal{V}$, the set of program variables; E is an arithmetical expression over $\mathcal{V}$; G a Boolean expression over $\mathcal{V}$; and p a real–valued parameter with domain [0, 1]. Most of the pGCL instructions are self–explanatory; we elaborate only on the following: $\{P\}\;[p]\;\{Q\}$ is a *probabilistic choice* where program P is executed with probability p and program Q with probability $1-p$; $\{P\} \,\square\, \{Q\}$ is a *non–deterministic choice* between P and Q; finally `abort` is syntactic sugar for the diverging program `while` (`true`) {`skip`}.

To model probabilistic programs with conditioning we extend pGCL with observations, leading to the *conditional* pGCL (cpGCL). At the syntactic level, an observation is introduced with the instruction `observe` (G), G being a Boolean expression over $\mathcal{V}$. The effect of such an instruction is to block all invalid program executions violating G and rescale the probability of the remaining executions so that they sum up to one.

As an illustrative example consider the following pair of programs:

$$P_1\colon \quad \{x \coloneqq 0\}\ [p]\ \{x \coloneqq 1\};\ \{y \coloneqq 0\}\ [q]\ \{y \coloneqq -1\}$$
$$P_2\colon \quad \{x \coloneqq 0\}\ [p]\ \{x \coloneqq 1\};\ \{y \coloneqq 0\}\ [q]\ \{y \coloneqq -1\};\ \texttt{observe}\ (x{+}y{=}0)$$

Program P_1 admits all (four) runs, two of which satisfy $x{=}0$; for this program the probability of $x{=}0$ is p. Program P_2—due to the `observe` statement requiring $x{+}y{=}0$—admits only two runs, only one of them satisfying $x{=}0$; for this program the probability of $x{=}0$ is $\frac{pq}{pq+(1-p)(1-q)}$.

Note that there exists a connection between the `observe` statement used in our work and the well–known assert statement. Both statements `observe` (G) and assert (G) block all runs violating G. The crucial difference, however, is that `observe` (G) normalizes the probability of the unblocked runs while assert (G) does not, yielding then a sub–distribution of total mass possibly less than one [20,4].

3 Denotational Semantics for Conditional pGCL

In this section we recall the expectation transformer semantics of pGCL and extend it to conditional programs in the fully probabilistic fragment of cpGCL.

3.1 Expectation Transformers in pGCL

Expectation transformers are a quantitative version of predicate transformers [7] used to endow probabilistic pGCL programs a formal semantics. Loosely speaking, they capture the average or expected outcome of a program, measured w.r.t. a utility or reward function over the set of final states. To make this more precise, let $\mathbb{S}$ be the set of program states, where a *program state* is a variable valuation. Now assume that P is a *fully probabilistic* program, i.e. a program without non–deterministic choices. Intuitively, we can think of P as a mapping from an initial state $\sigma \in \mathbb{S}$ to a distribution of final states $[\![P]\!](\sigma)$; its formal semantics is captured by a transformer $\mathsf{wp}[P]$, which acts as follows: Given a random variable $f\colon \mathbb{S} \to \mathbb{R}_{\geq 0}$, $\mathsf{wp}[P](f)$ maps every initial state σ to the expected value $\mathbf{E}_{[\![P]\!](\sigma)}(f)$ of f with respect to the distribution of final states $[\![P]\!](\sigma)$. Symbolically,

$$\mathsf{wp}[P](f)(\sigma) \;=\; \mathbf{E}_{[\![P]\!](\sigma)}(f)\ .$$

In particular, if $f = \chi_A$ is the characteristic function of some event A, $\mathsf{wp}[P](f)$ retrieves the probability that the event occurred after the execution of P. (Moreover, if P is a deterministic program in GCL, $\mathbf{E}_{[\![P]\!](\sigma)}(\chi_A)$ is $\{0,1\}$–valued and we recover the ordinary notion of predicate transformers introduced by Dijkstra [7].)

For a program P including non–deterministic choices, the execution of P yields a *set* of final distributions. To account for this, we assume that $\mathsf{wp}[P](f)(\sigma)$ gives the tightest lower bound $\inf_{\mu \in [\![P]\!](\sigma)} \mathbf{E}_{\mu}(f)$ for the expected value of f. This corresponds with the notion of a *demonic* adversary resolving the non–deterministic choices.

We follow McIver and Morgan [21] and use the term *expectation* to refer to a random variable mapping program states to real values. The expectation transformer wp then transforms a post–expectation f into a pre–expectation $\mathsf{wp}[P](f)$ and can be defined by induction on the structure of P, following the rules in Figure 1. The transformer wp also admits a liberal variant wlp, which differs from wp in the way in which non–termination is treated.

Formally, the transformer wp operates on *unbounded expectations* in $\mathbb{E} = \mathbb{S} \to \mathbb{R}^{\infty}_{\geq 0}$ and wlp operates on *bounded expectations* in $\mathbb{E}_{\leq 1} = \mathbb{S} \to [0, 1]$. Here $\mathbb{R}^{\infty}_{\geq 0}$ denotes the set of non–negative real values with the adjoined ∞ value. In order to guarantee the well–definedness of wp and wlp we need to provide $\mathbb{E}$ and $\mathbb{E}_{\leq 1}$ the structure of a directed–complete partial order. Expectations are ordered pointwise, i.e. $f \sqsubseteq g$ iff $f(\sigma) \leq g(\sigma)$ for every state $\sigma \in \mathbb{S}$. The least upper bound of directed subsets is also defined pointwise.

In the remainder we make use of the following notation related to expectations. We use bold fonts for constant expectations, e.g. $\mathbf{1}$ denotes the constant expectation 1. Given an arithmetical expression E over program variables we simply write E for the expectation that in state σ returns $\sigma(E)$. Given a Boolean expression G over program variables we use χ_G to denote the $\{0, 1\}$–valued expectation that returns 1 if $\sigma \models G$ and 0 otherwise.

3.2 Conditional Expectation Transformers

We now study how to extend the notion of expectation transformers to conditioned probabilistic programs without non–determinism in cpGCL. To illustrate the intuition behind our solution, consider the following scenario: Assume we want to measure the probability that some event A occurs after the execution of a conditioned program P. Since P contains observations, its execution leads to a conditional distribution $\mu|_O$ of final states. Now the conditional probability that A occurs (given that O occurs) is given as the quotient of the probabilities $\Pr[\mu \in A \wedge O]$ and $\Pr[\mu \in O]$. Motivated by this observation, we introduce an expectation transformer $\mathsf{cwp}[\cdot] \colon \mathbb{E} \times \mathbb{E}_{\leq 1} \to \mathbb{E} \times \mathbb{E}_{\leq 1}$, whose application $\mathsf{cwp}[P](\chi_A, \mathbf{1})$ will yield the desired pair of probabilities $(\Pr[\mu \in A \wedge O], \Pr[\mu \in O])$. We are only left to define a transformer $\underline{\mathsf{cwp}}[P]$ that computes the corresponding quotient. Formally, we let

$$\underline{\mathsf{cwp}}[P](f) \;\triangleq\; \frac{\mathsf{cwp}_1[P](f, \mathbf{1})}{\mathsf{cwp}_2[P](f, \mathbf{1})} \,,$$

where $\mathsf{cwp}_1[P](f, g)$ (resp. $\mathsf{cwp}_2[P](f, g)$) denotes the first (resp. second) component of $\mathsf{cwp}[P](f, g)$. If $\mathsf{cwp}_2[P](f, \mathbf{1})(\sigma) = 0$, then $\underline{\mathsf{cwp}}[P](f)$ is not well–defined in σ (in the same way as the conditional probability $\Pr(A \mid B)$ is not well–defined [4] when $\Pr(B) = 0$) and we say that program P is *infeasible* from state σ, meaning that *all* its executions are blocked by observations.

As so defined, $\underline{\mathsf{cwp}}[P](f)$ represents the weakest *conditional* pre–expectation of

[4] In the *continuous* setting we could define a conditional density even when conditioning on events with 0 measure using the Radon–Nikodym theorem. However, our programs generate *discrete* distributions only.

P	$\mathsf{wp}[P](f)$	$\mathsf{cwp}[P](f,\, g)$
`skip`	f	$(f,\, g)$
`abort`	$\mathbf{0}$	$(\mathbf{0},\, \mathbf{1})$
$x \coloneqq E$	$f[x/E]$	$(f[x/E],\, g[x/E])$
`observe` (G)	— not defined —	$\chi_G \cdot (f,\, g)$
$P_1;\ P_2$	$(\mathsf{wp}[P_1] \circ \mathsf{wp}[P_2])(f)$	$(\mathsf{cwp}[P_1] \circ \mathsf{cwp}[P_2])(f, g)$
`ite` $(G)\,\{P_1\}\,\{P_2\}$	$\chi_G \cdot \mathsf{wp}[P_1](f) + \chi_{\neg G} \cdot \mathsf{wp}[P_2](f)$	$\chi_G \cdot \mathsf{cwp}[P_1](f,\, g) + \chi_{\neg G} \cdot \mathsf{cwp}[P_2](f,\, g)$
$\{P_1\}\ [p]\ \{P_2\}$	$p \cdot \mathsf{wp}[P_1](f) + (1{-}p) \cdot \mathsf{wp}[P_2](f)$	$p \cdot \mathsf{cwp}[P_1](f,\, g) + (1{-}p) \cdot \mathsf{cwp}[P_2](f,\, g)$
$\{P_1\} \,\square\, \{P_2\}$	$\lambda\sigma\!\bullet\ \min\{\mathsf{wp}[P_1](f)(\sigma),\, \mathsf{wp}[P_2](f)(\sigma)\}$	— not defined —
`while` $(G)\,\{P'\}$	$\boldsymbol{\mu}\, \hat{f}\!\bullet\ \Big(\chi_G \cdot \mathsf{wp}[P'](\hat{f}) + \chi_{\neg G} \cdot f\Big)$	$\boldsymbol{\mu}_{\sqsubseteq,\sqsupseteq}(\hat{f},\, \hat{g})\!\bullet\ \Big(\chi_G \cdot \mathsf{cwp}[P'](\hat{f},\, \hat{g}) + \chi_{\neg G} \cdot (f,\, g)\Big)$
P	$\mathsf{wlp}[P](f)$	$\mathsf{cwlp}[P](f,\, g)$
`abort`	$\mathbf{1}$	$(\mathbf{1},\, \mathbf{1})$
`while` $(G)\,\{P'\}$	$\boldsymbol{\nu}\, \hat{f}\!\bullet\ \Big(\chi_G \cdot \mathsf{wp}[P'](\hat{f}) + \chi_{\neg G} \cdot f\Big)$	$\boldsymbol{\nu}_{\sqsubseteq,\sqsubseteq}(\hat{f},\, \hat{g})\!\bullet\ \Big(\chi_G \cdot \mathsf{cwp}[P'](\hat{f},\, \hat{g}) + \chi_{\neg G} \cdot (f,\, g)\Big)$

Fig. 1. Definitions for the wp/wlp and cwp/cwlp operators. The wlp (cwlp) operator differs from wp (cwp) only for `abort` and the `while`–loop. Multiplication $h \cdot (f,\, g)$ is meant componentwise yielding $(h \cdot f,\, h \cdot g)$. Likewise, addition $(f,\, g) + (f',\, g')$ is meant componentwise yielding $(f + f',\, g + g')$.

P with respect to post–expectation f and $\underline{\mathsf{cwp}}[\cdot]$ generalizes the transformer $\mathsf{wp}[\cdot]$ to conditioned programs. The weakest liberal conditional pre–expectation $\underline{\mathsf{cwlp}}[P](f)$ is defined analogously, in terms of the transformer $\mathsf{cwlp}[P] \colon \mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1} \to \mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1}$.

We are only left to provide definitions for $\mathsf{cwp}[P]$ and $\mathsf{cwlp}[P]$. Both transformers are defined by induction on the structure of P, following the rules in Figure 1. Let us briefly explain these rules. $\mathsf{cwp}[\texttt{skip}]$ behaves as the identity since `skip` has no effect. $\mathsf{cwp}[\texttt{abort}]$ 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. $\mathsf{cwp}[x \coloneqq E]$ maps (f, g) to pre–expectation $(f[x/E],\, g[x/E])$, where $h[x/E](\sigma) = h(\sigma[x/E])$ and $\sigma[x/E]$ denotes the usual variable update on states. $\mathsf{cwp}[P_1; P_2]$ is obtained as the functional composition (denoted $\circ$) of $\mathsf{cwp}[P_1]$ and $\mathsf{cwp}[P_2]$. $\mathsf{cwp}[\texttt{observe}\ (G)]$ restricts post–expectations to those states that satisfy G; states that do not satisfy G are mapped to 0. $\mathsf{cwp}[\texttt{ite}\,(G)\,\{P_1\}\,\{P_2\}]$ behaves either as $\mathsf{cwp}[P_1]$ or $\mathsf{cwp}[P_2]$ according to the evaluation of G. $\mathsf{cwp}[\{P_1\}\ [p]\ \{P_2\}]$ is obtained as a convex combination of $\mathsf{cwp}[P_1]$ and $\mathsf{cwp}[P_2]$, weighted according to p. $\mathsf{cwp}[\texttt{while}\,(G)\,\{P'\}]$ is defined using standard fixed point techniques.[5] The cwlp transformer follows the same rules as cwp, except for the `abort` and `while` statements. $\mathsf{cwlp}[\texttt{abort}]$ takes any post–expectation to pre–expectation $(\mathbf{1}, \mathbf{1})$; $\mathsf{cwlp}[\texttt{while}\,(G)\,\{P\}]$ is defined in terms of a greatest rather than a least fixed point.

Observe that Figure 1 presents no rule for the non–deterministic choice operator. Therefore our conditional expectation transformers $\underline{\mathsf{cwp}}$/$\underline{\mathsf{cwlp}}$ can only handle fully probabilistic cpGCL programs. In Section 6 we elaborate on this limitation.

Example 3.1 Assume we want to compute the expected value of the expression

[5] We define $\mathsf{cwp}[\texttt{while}\,(G)\,\{P\}]$ as the least fixed point w.r.t. the order $(\sqsubseteq, \sqsupseteq)$ in $\mathbb{E} \times \mathbb{E}_{\leq 1}$. This way we encode the greatest fixed point in the second component w.r.t. the order $\sqsubseteq$ over $\mathbb{E}_{\leq 1}$ as the least fixed point w.r.t. the dual order $\sqsupseteq$.

$10{+}x$ after executing program P' given as:

1: $\{x \coloneqq 0\}\ [1/2]\ \{x \coloneqq 1\};$
2: $\texttt{ite}\,(x=1)\,\big\{\{y \coloneqq 0\}\ [1/2]\ \{y \coloneqq 2\}\big\}\big\{\{y \coloneqq 0\}\ [4/5]\ \{y \coloneqq 3\}\big\};$
3: $\texttt{observe}\ (y{=}0)$

The computation of $\mathsf{cwp}[P'](10{+}x, \mathbf{1})$ goes as follows:

$$\begin{aligned}
\mathsf{cwp}[P'](10{+}x,\mathbf{1}) &= \mathsf{cwp}[P'_{1\text{-}2}](\mathsf{cwp}[\texttt{observe}\ (y{=}0)](10{+}x,\mathbf{1}))\\
&= \mathsf{cwp}[P'_{1\text{-}2}](f,g) \text{ where } (f,g) = \chi_{y=0}\cdot(10{+}x,\mathbf{1})\\
&= \mathsf{cwp}[P'_{1\text{-}1}](\mathsf{cwp}[\texttt{ite}\,(x{=}1)\,\{\ldots\}\,\{\ldots\}](f,g))\\
&= \mathsf{cwp}[P'_{1\text{-}1}](\chi_{x=1}\cdot(h,i) + \chi_{x\neq 1}\cdot(h',i')) \text{ where}\\
&\quad (h,i) = \mathsf{cwp}[\{y{\coloneqq}0\}\ [1/2]\ \{y{\coloneqq}2\}](f,g) = \tfrac{1}{2}\cdot(10+x,\,\mathbf{1})\ , \text{ and}\\
&\quad (h',i') = \mathsf{cwp}[\{y{\coloneqq}0\}\ [4/5]\ \{y{\coloneqq}3\}](f,g) = \tfrac{4}{5}\cdot(10+x,\,\mathbf{1})\\
&= \tfrac{1}{2}\cdot\tfrac{4}{5}\cdot(\mathbf{10+0},\,\mathbf{1}) + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot(\mathbf{10+1},\,\mathbf{1}) = \big(\tfrac{\mathbf{27}}{\mathbf{4}},\tfrac{\mathbf{13}}{\mathbf{20}}\big)\ .
\end{aligned}$$

The expected value of $10{+}x$ is then given by $\underline{\mathsf{cwp}}[P'](10{+}x) = \frac{\mathbf{27}}{\mathbf{4}}/\frac{\mathbf{13}}{\mathbf{20}} = \frac{\mathbf{135}}{\mathbf{13}} \approx 10.38$.

In the rest of this section we investigate some properties of the expectation transformer semantics (of the fully probabilistic fragment) of cpGCL. As every fully probabilistic pGCL program is contained in cpGCL, we begin by studying the relation between the wp–semantics of pGCL and the cwp–semantics of cpGCL. To that end, we extend the wp operator to cpGCL by the clauses $\mathsf{wp}[\texttt{observe}\ (G)](f) = \chi_G \cdot f$ and $\mathsf{wlp}[\texttt{observe}\ (G)](f) = \chi_G \cdot f$. Our first result says that cwp (resp. cwlp) can be decoupled as the product wp × wlp (resp. wlp × wlp).

Lemma 3.2 (Decoupling of cw(l)p) *Let P be a fully probabilistic* cpGCL *program, $f \in \mathbb{E}$ and $f', g \in \mathbb{E}_{\leq 1}$. Then* $\mathsf{cwp}[P](f,\,g) = \big(\mathsf{wp}[P](f),\,\mathsf{wlp}[P](g)\big)$ *and* $\mathsf{cwlp}[P](f',\,g) = \big(\mathsf{wlp}[P](f'),\,\mathsf{wlp}[P](g)\big)$.

Our next result shows that the cwp–semantics is a conservative extension of the wp–semantics for the fully probabilistic fragment of pGCL. The same applies to the weakest liberal pre–expectation semantics.

Theorem 3.3 (Compatibility with the w(l)p–semantics) *Let P be a fully probabilistic* pGCL *program, $f \in \mathbb{E}$, and $g \in \mathbb{E}_{\leq 1}$. Then* $\mathsf{wp}[P](f) = \underline{\mathsf{cwp}}[P](f)$ *and* $\mathsf{wlp}[P](g) = \underline{\mathsf{cwlp}}[P](g)$.

Proof. By Lemma 3.2 and the fact that $\mathsf{wlp}[P](\mathbf{1}) = \mathbf{1}$ (see Lemma 3.4). □

We now show that cwp and cwlp preserve the so–called healthiness conditions of wp and wlp.

Lemma 3.4 (Healthiness conditions of cwp and cwlp) *For every fully probabilistic* cpGCL *program P with at least one feasible execution (from every initial state), every $f, g \in \mathbb{E}$ and non–negative real constants α, β:*

i) $f \sqsubseteq g$ implies $\underline{\mathsf{cwp}}[P](f) \sqsubseteq \underline{\mathsf{cwp}}[P](g)$ and likewise for $\underline{\mathsf{cwlp}}$ (monotonicity).

ii) $\underline{\mathsf{cwp}}[P](\alpha \cdot f + \beta \cdot g) = \alpha \cdot \underline{\mathsf{cwp}}[P](f) + \beta \cdot \underline{\mathsf{cwp}}[P](g)$ *(linearity).*

iii) $\underline{\mathsf{cwp}}[P](\mathbf{0}) = \mathbf{0}$ *and* $\underline{\mathsf{cwlp}}[P](\mathbf{1}) = \mathbf{1}$.

Proof. Using Lemma 3.2 one can show that the transformers $\underline{\mathsf{cwp}}$ and $\underline{\mathsf{cwlp}}$ inherit these properties from wp and wlp. For details see [12, p. 15]. □

We conclude this section by discussing alternative approaches for providing an expectation transformer semantics for $P \in \mathsf{cpGCL}$. By Lemma 3.2, the transformers $\underline{\mathsf{cwp}}[P]$ and $\underline{\mathsf{cwlp}}[P]$ can be recast as

$$f \mapsto \frac{\mathsf{wp}[P](f)}{\mathsf{wlp}[P](\mathbf{1})} \qquad \text{and} \qquad f \mapsto \frac{\mathsf{wlp}[P](f)}{\mathsf{wlp}[P](\mathbf{1})}\ ,$$

respectively. An alternative is to normalize using wp instead of wlp in the denominator, yielding the two transformers

$$\textit{i)}\quad f \mapsto \frac{\mathsf{wp}[P](f)}{\mathsf{wp}[P](\mathbf{1})} \qquad \text{and} \qquad \textit{ii)}\quad f \mapsto \frac{\mathsf{wlp}[P](f)}{\mathsf{wp}[P](\mathbf{1})}\ .$$

Transformer *ii)* is not meaningful, as the denominator $\mathsf{wp}[P](\mathbf{1})(\sigma)$ may be smaller than the numerator $\mathsf{wlp}[P](f)(\sigma)$ for some state $\sigma \in \mathbb{S}$. This might lead to probabilities exceeding one. Transformer *i)* normalizes w.r.t. the terminating executions. This interpretation corresponds to the semantics of the probabilistic programming language `R2` [22,16] and is *only meaningful if programs terminate almost surely* (i.e. with probability one). A noteworthy consequence of adopting transformer *i)* is that `observe` (G) is equivalent to `while` $(\neg G)$ $\{$`skip`$\}$ [16], see the discussion in Section 5.

Let us briefly compare the four alternatives by means of a concrete program P:

$$\big\{\texttt{abort}\big\}\ [1/2]\big\{\{x \coloneqq 0\}\ [1/2]\ \{x \coloneqq 1\}; \{y \coloneqq 0\}\ [1/2]\ \{y \coloneqq 1\};\ \texttt{observe}\ (x{=}0 \vee y{=}0)\big\}$$

P tosses a fair coin and according to the outcome either diverges or tosses a fair coin twice and observes at least once heads $(y{=}0 \vee x{=}0)$. We measure the probability that the outcome of the last coin toss was heads according to each transformer:

$$\frac{\mathsf{wp}[P](\chi_{y=0})}{\mathsf{wlp}[P](\mathbf{1})} = \frac{2}{7} \qquad \frac{\mathsf{wlp}[P](\chi_{y=0})}{\mathsf{wlp}[P](\mathbf{1})} = \frac{6}{7} \qquad \frac{\mathsf{wp}[P](\chi_{y=0})}{\mathsf{wp}[P](\mathbf{1})} = \frac{2}{3} \qquad \frac{\mathsf{wlp}[P](\chi_{y=0})}{\mathsf{wp}[P](\mathbf{1})} = 2$$

As mentioned before, the transformer *ii)* is not significant as it yields a "probability" exceeding one. Note that our cwp–semantics yields that the probability of $y{=}0$ after the execution of P while passing all `observe`–statements is $\frac{2}{7}$. As shown before, this is a conservative and natural extension of the wp–semantics. This does not apply to the `R2`–semantics, as this would require an adaptation of rules for `abort` and `while`.

4 Operational Semantics for Conditional pGCL

This section presents an operational semantics for cpGCL using Markov decision processes (MDPs) as underlying model. We begin by recalling the notion of MDPs.

For that, let $Distr(S)$ denote the set of distributions $\mu\colon S \to \mathbb{R}$ over S with $\sum_{s\in S} \mu(s) = 1$.

Definition 4.1 An *MDP* is a tuple $\mathfrak{R} = (S, s_I, Act, \mathcal{P}, L)$ with a countable set of states S, an initial state $s_I \in S$, a finite set of actions Act, a transition probability function $\mathcal{P}\colon S \times Act \to Distr(S)$ with $\sum_{s'\in S} \mathcal{P}(s, \alpha)(s') = 1$ for all $(s, \alpha) \in S \times Act$ and a labeling function $L\colon S \to 2^{AP}$ for a set of atomic propositions AP.

A function $r\colon S \to \mathbb{R}_{\geq 0}$ is used to add *rewards* to an MDP. A *path* of $\mathfrak{R}$ is a finite or infinite sequence $\pi = s_0\alpha_0 s_1\alpha_1 \ldots$ such that $s_i \in S$, $\alpha_i \in Act$, $s_0 = s_I$, and $\mathcal{P}(s_i, \alpha_i)(s_{i+1}) > 0$ for all $i \geq 0$. The i-th state s_i of π is denoted $\pi(i)$. The set of all paths of $\mathfrak{R}$ is denoted by $\mathsf{Paths}^{\mathfrak{R}}$. $\mathsf{Paths}^{\mathfrak{R}}(s)$ is the set of paths starting in s and $\mathsf{Paths}^{\mathfrak{R}}(s, s')$ is the set of all finite paths starting in s and ending in s'. This is also lifted to sets of states. We sometimes omit superscript $\mathfrak{R}$ in $\mathsf{Paths}^{\mathfrak{R}}$.

MDPs operate by a non–deterministic choice of an action $\alpha \in Act$ that is *enabled* at state s and a subsequent probabilistic determination of a successor state according to $\mathcal{P}(s, \alpha)$. For resolving the non–deterministic choices, so–called *schedulers* are used. Here, *deterministic* and *memoryless* schedulers suffice which are functions $\mathfrak{S}\colon S \to Act$. Let $Sched^{\mathfrak{R}}$ denote the class of all such schedulers for $\mathfrak{R}$.

For MDP $\mathfrak{R}$, the fully probabilistic system ${}^{\mathfrak{S}}\mathfrak{R}$ induced by a scheduler $\mathfrak{S} \in Sched^{\mathfrak{R}}$ is called the *induced Markov Chain (MC)* on which a *probability measure* over paths is defined. The measure $\Pr^{\mathcal{R}}$ for MC $\mathcal{R}$ is given by $\Pr^{\mathcal{R}}\colon \mathsf{Paths}^{\mathcal{R}} \to [0, 1] \subseteq \mathbb{R}$ with $\Pr^{\mathcal{R}}(\hat{\pi}) = \prod_{i=0}^{n-1} \mathcal{P}(s_i, s_{i+1})$, for a finite path $\hat{\pi} = s_0 \ldots s_n$. This is lifted to infinite paths using the standard cylinder set construction, see [2, Ch. 10]. The *cumulated reward* of a finite path $\hat{\pi} = s_0 \ldots s_n$ is given by $r(\hat{\pi}) = \sum_{i=0}^{n-1} r(s_i)$. Note that in our special setting the cumulated reward will not be infinite.

We consider *reachability properties* $\Diamond T$ for a set of target states $T \subseteq S$ where $\Diamond T$ also denotes all paths that reach T from the initial state s_I. Analogously, the set $\neg\Diamond T$ contains all paths that never reach a state in T.

First, consider reward objectives for MCs. The *expected reward* for a countable set of paths $\Diamond T$ is given by $\mathsf{ExpRew}^{\mathcal{R}}(\Diamond T) = \sum_{\hat{\pi}\in\Diamond T} \Pr^{\mathcal{R}}(\hat{\pi}) \cdot r(\hat{\pi})$. For a reward bounded by one, the notion of the *liberal* expected reward also takes the mere probability of *not* reaching the target states into account: $\mathsf{LExpRew}^{\mathcal{R}}(\Diamond T) = \mathsf{ExpRew}^{\mathcal{R}}(\Diamond T) + \Pr^{\mathcal{R}}(\neg\Diamond T)$. To exclude the probability of paths that reach "undesired" states, we let $U = \{s \in S \mid \lightning \in L(s)\}$ and define the *conditional expected reward* for the condition $\neg\Diamond U$ by [6]

$$\mathsf{CExpRew}^{\mathcal{R}}(\Diamond T \,|\, \neg\Diamond U) \;\triangleq\; \frac{\mathsf{ExpRew}^{\mathcal{R}}(\Diamond T \cap \neg\Diamond U)}{\Pr^{\mathcal{R}}(\neg\Diamond U)} \,.$$

Reward objectives for MDPs are now defined using a *demonic* scheduler $\mathfrak{S} \in Sched^{\mathfrak{R}}$ minimizing probabilities and expected rewards for the induced MC ${}^{\mathfrak{S}}\mathfrak{R}$. For the expected reward this yields $\mathsf{ExpRew}^{\mathfrak{R}}(\Diamond T) = \inf_{\mathfrak{S}\in Sched^{\mathfrak{R}}} \mathsf{ExpRew}^{{}^{\mathfrak{S}}\mathfrak{R}}(\Diamond T)$. For con-

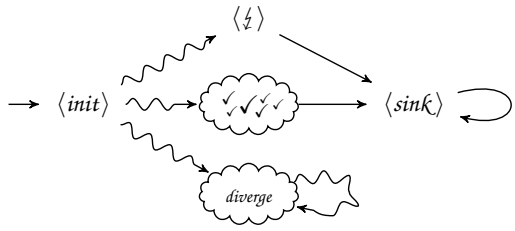
[6] Note that strictly formal one would have to define the intersection of sets of finite and possibly infinite paths by means of a cylinder set construction considering all infinite extensions of finite paths.

ditional expected reward properties, the value of the quotient is minimized:

$$\mathsf{CExpRew}^{\mathfrak{R}}\left(\Diamond\, T \,|\, \neg\Diamond\, U\right) \triangleq \inf_{\mathfrak{S}\in Sched^{\mathfrak{R}}} \frac{\mathsf{ExpRew}^{\mathfrak{S}\mathfrak{R}}\left(\Diamond\, T \cap \neg\Diamond\, U\right)}{\Pr^{\mathfrak{S}\mathfrak{R}}(\neg\Diamond\, U)} \,.$$

The liberal reward notions for MDPs are analogous. Regarding the quotient minimization we assume "$\frac{0}{0} < 0$" as we see $\frac{0}{0}$—being undefined—to be less favorable than 0. For details about conditional probabilities and expected rewards see [3].

The structure of the operational MDP of a cpGCL program is depicted on the right. Terminating runs eventually end up in the $\langle \mathit{sink} \rangle$ state; other runs are diverging (never reach $\langle \mathit{sink} \rangle$). A program terminates either successfully, i.e. a run passes a ✓–labelled state, or terminates due to a violation of an observation, i.e. a run passes $\langle \lightning \rangle$. Squiggly arrows indicate reaching certain states via possibly multiple paths and states; the clouds indicate that there might be several states of the particular kind. The ✓–labelled states are the *only ones* with positive reward. Note that the sets of paths that eventually reach $\langle \lightning \rangle$, or ✓, or diverge are pairwise disjoint.



Definition 4.2 [Operational cpGCL semantics] The *operational semantics* of $P \in \mathsf{cpGCL}$ for $\sigma \in \mathbb{S}$ and $f \in \mathbb{E}$ is the MDP $\mathfrak{R}^f_\sigma[\![P]\!] = (S, \langle P, \sigma\rangle, \mathit{Act}, \mathcal{P}, L, r)$, such that S is the smallest set of states with $\langle \lightning \rangle \in S$, $\langle \mathit{sink} \rangle \in S$, and $\langle Q, \tau\rangle, \langle \downarrow, \tau\rangle \in S$ for $Q \in \mathsf{pGCL}$ and $\tau \in \mathbb{S}$. $\langle P, \sigma\rangle \in S$ is the initial state. $\mathit{Act} = \{\mathit{left}, \mathit{right}\}$ is the set of actions. A state of the form $\langle \downarrow, \tau\rangle$ denotes a terminal state in which no program is left to be executed. $\mathcal{P}$ is formed according to SOS rules given in [12, p. 5].

For some $\tau \in \mathbb{S}$, the labelling and the reward function is given by:

$$L(s) \triangleq \begin{cases} \{\checkmark\}, & \text{if } s = \langle \downarrow, \tau\rangle \\ \{\mathit{sink}\}, & \text{if } s = \langle \mathit{sink}\rangle \\ \{\lightning\}, & \text{if } s = \langle \lightning \rangle \\ \emptyset, & \text{otherwise,} \end{cases} \qquad r(s) \triangleq \begin{cases} f(\tau), & \text{if } s = \langle \downarrow, \tau\rangle \\ 0, & \text{otherwise .} \end{cases}$$

To determine the *conditional expected outcome of program* P given that all observations are true, we need to determine the *expected reward to reach* $\langle \mathit{sink} \rangle$ *from the initial state conditioned on not reaching* $\langle \lightning \rangle$ under a demonic scheduler. For $\mathfrak{R}^f_\sigma[\![P]\!]$ this is given by $\mathsf{CExpRew}^{\mathfrak{R}^f_\sigma[\![P]\!]}\left(\Diamond \mathit{sink} \,|\, \neg\Diamond \lightning\right)$. Recall for the condition $\neg\Diamond \lightning$ that all paths not eventually reaching $\langle \lightning \rangle$ either diverge (thus collect reward 0) or pass by a ✓–labelled state and eventually reach $\langle \mathit{sink} \rangle$. This gives us:

$$\mathsf{CExpRew}^{\mathfrak{R}^f_\sigma[\![P]\!]}\left(\Diamond \mathit{sink} \,|\, \neg\Diamond \lightning\right) = \inf_{\mathfrak{S}\in Sched^{\mathfrak{R}^f_\sigma[\![P]\!]}} \frac{\mathsf{ExpRew}^{\mathfrak{S}\mathfrak{R}^f_\sigma[\![P]\!]}\left(\Diamond \mathit{sink} \cap \neg\Diamond \lightning\right)}{\Pr^{\mathfrak{S}\mathfrak{R}^f_\sigma[\![P]\!]}(\neg\Diamond \lightning)}$$

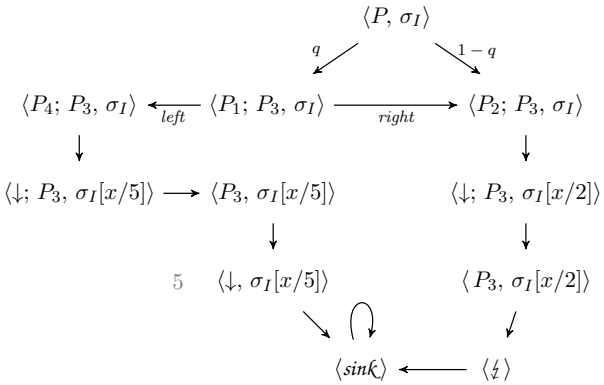
$$= \inf_{\mathfrak{S} \in Sched^{\mathfrak{R}^f_\sigma[\![P]\!]}} \frac{\mathsf{ExpRew}^{\mathfrak{S}_{\mathfrak{R}^f_\sigma[\![P]\!]}}(\Diamond sink)}{\Pr^{\mathfrak{S}_{\mathfrak{R}^f_\sigma[\![P]\!]}}(\neg\Diamond \lightning)} \ .$$

The liberal version $\mathsf{CLExpRew}^{\mathcal{R}^f_\sigma[\![P]\!]}(\Diamond sink \,|\, \neg\Diamond \lightning)$ is defined analogously.

Example 4.3 Consider the program $P \in \mathsf{cpGCL}$:

$$\{\{x \coloneqq 5\} \,\square\, \{x \coloneqq 2\}\} \ [q] \ \{x \coloneqq 2\}; \texttt{observe}\ (x{>}3)$$

where with parametrized probability q a non–deterministic choice between x being assigned 2 or 5 is executed, and with probability $1{-}q$, x is directly assigned 2. Let for readability $P_1 = \{x \coloneqq 5\} \,\square\, \{x \coloneqq 2\}$, $P_2 = x \coloneqq 2$, $P_3 = \texttt{observe}\ (x{>}3)$, and $P_4 = x \coloneqq 5$. The operational MDP $\mathfrak{R}^x_{\sigma_I}[\![P]\!]$ for an arbitrary initial variable valuation σ_I and post–expectation x is depicted below:



The only state with positive reward is $s' \coloneqq \langle\downarrow, \sigma_I[x/5]\rangle$ and its reward is indicated by number 5. Assume first a scheduler choosing action *left* in state $\langle P_1; P_3, \sigma_I\rangle$. In the induced MC the only path accumulating positive reward is the path π going from $\langle P, \sigma_I\rangle$ via s' to $\langle sink\rangle$ with $r(\pi) = 5$ and $\Pr(\pi) = q$. This gives an expected reward of $5 \cdot q$. The overall probability of not reaching $\langle \lightning \rangle$ is also q. The conditional expected reward of eventually reaching $\langle sink\rangle$ given that $\langle \lightning \rangle$ is not reached is hence $\frac{5 \cdot q}{q} = 5$. Assume now the *demonic* scheduler choosing *right* at state $\langle P_1; P_3, \sigma_I\rangle$. In this case there is no path having positive accumulated reward in the induced MC, yielding an expected reward of 0. The probability of not reaching $\langle \lightning \rangle$ is also 0. The conditional expected reward in this case is undefined ($0/0$) and thus the *right* branch is preferred over the *left* branch. In general, the operational MDP need not be finite, even if the program terminates almost–surely (i.e. with probability 1).

We now investigate the connection to the denotational semantics of Section 3, starting with some auxiliary results. First, we establish a relation between (liberal) expected rewards and weakest (liberal) pre–expectations.

Lemma 4.4 *For any fully probabilistic $P \in \mathsf{cpGCL}$, $f \in \mathbb{E}, g \in \mathbb{E}_{\leq 1}$, and $\sigma \in \mathbb{S}$:*

$$\mathsf{ExpRew}^{\mathcal{R}^f_\sigma[\![P]\!]}(\Diamond\langle sink\rangle) = \mathsf{wp}[P](f)(\sigma) \qquad \text{(i)}$$

$$\mathsf{LExpRew}^{\mathcal{R}^g_\sigma[\![P]\!]}\left(\Diamond\langle \mathit{sink}\rangle\right) = \mathsf{wlp}[P](g)(\sigma) \tag{ii}$$

Moreover, the probability of never reaching $\langle \lightning \rangle$ in the MC of program P coincides with the weakest liberal pre–expectation of P w.r.t. post–expectation $\mathbf{1}$:

Lemma 4.5 *For any fully probabilistic $P \in \mathsf{cpGCL}$, $g \in \mathbb{E}_{\leq 1}$, and $\sigma \in \mathbb{S}$ we have* $\Pr^{\mathcal{R}^g_\sigma[\![P]\!]}(\neg\Diamond\lightning) = \mathsf{wlp}[P](\mathbf{1})(\sigma)$.

We now have all prerequisites in order to present the main result of this section: the correspondence between the operational and expectation transformer semantics of fully probabilistic cpGCL programs. It turns out that the weakest (liberal) pre–expectation $\underline{\mathsf{cwlp}}[P](f)(\sigma)$ (resp. $\underline{\mathsf{cwlp}}[P](f)(\sigma)$) coincides with the conditional (liberal) expected reward in the RMC $\mathcal{R}^f_\sigma[\![P]\!]$ of terminating while never violating an `observe`–statement, i.e., avoiding the $\langle \lightning \rangle$ states.

Theorem 4.6 (Correspondence theorem) *For any fully probabilistic $P \in \mathsf{cpGCL}$, $f \in \mathbb{E}$, $g \in \mathbb{E}_{\leq 1}$ and $\sigma \in \mathbb{S}$,*

$$\begin{aligned}\mathsf{CExpRew}^{\mathcal{R}^f_\sigma[\![P]\!]}\left(\Diamond \mathit{sink} \mid \neg\Diamond\lightning\right) &= \underline{\mathsf{cwp}}[P](f)(\sigma)\\ \mathsf{CLExpRew}^{\mathcal{R}^g_\sigma[\![P]\!]}\left(\Diamond \mathit{sink} \mid \neg\Diamond\lightning\right) &= \underline{\mathsf{cwlp}}[P](g)(\sigma)\ .\end{aligned}$$

Proof. The proof makes use of Lemmas 4.4, 4.5, and Lemma 3.2 which are themselves proven by induction on the structure of P. For details see [12, p. 13-14, 16-21]. □

5 Program Transformation

In this section we present a program transformation for removing observations from fully probabilistic cpGCL programs and use the expectation transformer semantics from Section 3 to prove the transformation correct. Intuitively, the presented program transformation "hoists" the `observe` statements while updating the probabilities of the probabilistic choices. Given a fully probabilistic program $P \in \mathsf{cpGCL}$, the transformation delivers a semantically equivalent `observe`–free program $\hat{P} \in \mathsf{pGCL}$ and—as a side product—an expectation $\hat{h} \in \mathbb{E}_{\leq 1}$ that captures the probability that the original program establishes all `observe` statements. For an intuition, reconsider the program from Example 3.1. The transformation yields program

$$\{x \coloneqq 0\}\ [8/13]\ \{x \coloneqq 1\}; \texttt{ite}\ (x{=}1)\ \{\{y \coloneqq 0\}\ [1]\ \{y \coloneqq 2\}\}\{\{y \coloneqq 0\}\ [1]\ \{y \coloneqq 3\}\}$$

and expectation $\hat{h} = \mathbf{\frac{13}{20}}$. By eliminating dead code in both probabilistic choices and coalescing the branches in the conditional, we can simplify the program to

$$\{x \coloneqq 0\}\ [8/13]\ \{x \coloneqq 1\};\ y \coloneqq 0$$

As a sanity check note that the expected value of $10{+}x$ in this program is equal to $10 \cdot \frac{8}{13} + 11 \cdot \frac{5}{13} = \frac{135}{13}$, which agrees with the result obtained by analyzing the

$$
\begin{array}{ll}
\mathcal{T}(\texttt{skip}, f) & = (\texttt{skip}, f) \\
\mathcal{T}(\texttt{abort}, f) & = (\texttt{abort}, \mathbf{1}) \\
\mathcal{T}(x \coloneqq E, f) & = (x \coloneqq E, f[E/x]) \\
\mathcal{T}(\texttt{observe}\ (G), f) & = (\texttt{skip}, \chi_G \cdot f) \\
\mathcal{T}(\texttt{ite}\,(G)\,\{P\}\,\{Q\}, f) & = (\texttt{ite}\,(G)\,\{P'\}\,\{Q'\}, \chi_G \cdot f_P + \chi_{\neg G} \cdot f_Q) \ \text{where} \\
 & \quad (P', f_P) = \mathcal{T}(P, f),\ (Q', f_Q) = \mathcal{T}(Q, f) \\
\mathcal{T}(\{P\}\ [p]\ \{Q\}, f) & = (\{P'\}\ [p']\ \{Q'\}, p \cdot f_P + (\mathbf{1}{-}p) \cdot f_Q) \ \text{where} \\
 & \quad (P', f_P) = \mathcal{T}(P, f),\ (Q', f_Q) = \mathcal{T}(Q, f),\ p' = \frac{p \cdot f_P}{p \cdot f_P + (\mathbf{1}-p) \cdot f_Q} \\
\mathcal{T}(\texttt{while}\,(G)\,\{P\}, f) & = (\texttt{while}\,(G)\,\{P'\}, f') \ \text{where} \\
 & \quad f' = \boldsymbol{\nu}\, X \bullet\ (\chi_G \cdot (\pi_2 \circ \mathcal{T})(P, X) + \chi_{\neg G} \cdot f)\,,\ (P', \_) = \mathcal{T}(P, f') \\
\mathcal{T}(P; Q, f) & = (P'; Q', f'') \ \text{where}\ (Q', f') = \mathcal{T}(Q, f),\ (P', f'') = \mathcal{T}(P, f')
\end{array}
$$

Fig. 2. Program transformation for eliminating `observe` statements in fully probabilistic cpGCL programs.

original program. Formally, the program transformation is given by a function

$$\mathcal{T} : \mathsf{cpGCL} \times \mathbb{E}_{\leq 1} \to \mathsf{pGCL} \times \mathbb{E}_{\leq 1}\ .$$

To apply the transformation to a program P we need to determine $\mathcal{T}(P, \mathbf{1})$, which gives the semantically equivalent program $\hat{P}$ and the expectation $\hat{h}$.

The transformation is defined in Figure 2 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 5.1 (Program Transformation Correctness) *Let P be a fully probabilistic* cpGCL *program that admits at least one valid run for every initial state and let $\mathcal{T}(P, \mathbf{1}) = (\hat{P}, \hat{h})$. Then for any $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leq 1}$, we have $\mathsf{wp}[\hat{P}](f) = \underline{\mathsf{cwp}}[P](f)$ and $\mathsf{wlp}[\hat{P}](g) = \underline{\mathsf{cwlp}}[P](g)$.*

Proof. See [12, p. 21]. □

A similar program transformation has been given by Nori *et al.* [22]. Whereas they use random assignments to introduce randomization in their programming model, we use probabilistic choices. Consequently, they can hoist `observe`–statements only until the occurrence of a random assignment, while we are able to hoist `observe`–statements over probabilistic choices and completely remove them from programs. Another difference is that the semantics of Nori *et al.* only accounts for terminating program behaviors and thus they can guarantee the correctness of the program transformation for almost–surely terminating programs only. Our semantics is more expressive and enables establishing the correctness for non–terminating program behavior, too.

6 Denotational Semantics for Full cpGCL

In this section we argue why (under mild assumptions) it is not possible to provide a denotational semantics in the style of conditional pre–expectation transformers (CPETs for short) for full cpGCL, i.e. including non–determinism. 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 with respect to some *fixed* initial state σ_0 and let $[\![\cdot]\!] \colon D \to \mathbb{R} \cup \{\bot\}$ be an interpretation function such that for any $d \in D$ we have that $[\![d]\!]$ *is equal to* the (possibly undefined) conditional expected value of x.

Definition 6.1 [Inductive CPETs] A *CPET* is a function $\mathsf{cwp}^* \colon \mathsf{cpGCL}^- \to D$ such that for any $P \in \mathsf{cpGCL}^-$, $[\![\mathsf{cwp}[P]]\!] = \mathsf{CExpRew}^{\mathfrak{R}^x_{\sigma_0}[\![P]\!]}(\Diamond\, \mathit{sink} \mid \neg\Diamond\, \lightning)$. cwp^* is called *inductive*, if there exist two functions $\mathcal{K} \colon \mathsf{cpGCL}^- \times [0,1] \times \mathsf{cpGCL}^- \to D$ and $\mathcal{N} \colon \mathsf{cpGCL}^- \times \mathsf{cpGCL}^- \to D$, such that for any $P_1, P_2 \in \mathsf{cpGCL}^-$ we have $\mathsf{cwp}^*[\{P_1\}\ [p]\ \{P_2\}] = \mathcal{K}(\mathsf{cwp}^*[P_1], p, \mathsf{cwp}^*[P_2])$ and $\mathsf{cwp}^*[\{P_1\} \,\square\, \{P_2\}] = \mathcal{N}(\mathsf{cwp}^*[P_1], \mathsf{cwp}^*[P_2])$, where $\forall d_1, d_2 \in D \bullet \mathcal{N}(d_1, d_2) \in \{d_1, d_2\}$.

This definition suggests that the conditional pre–expectation of $\{P_1\}\ [p]\ \{P_2\}$ is determined only by the conditional pre–expectation of P_1 and P_2, and the probability p. Furthermore the above definition suggests that the conditional pre–expectation of $\{P_1\} \,\square\, \{P_2\}$ is also determined by the conditional pre–expectation of P_1 and P_2 only. Consequently, the non–deterministic choice can be resolved by replacing it either by P_1 or P_2. While this might seem like a strong limitation, the above definition is compatible with the interpretation of non–deterministic choice as demonic choice: The choice is deterministically driven towards the worst option. The requirement $N(d_1, d_2) \in \{d_1, d_2\}$ is also necessary for interpreting non–deterministic choice as an abstraction where implementation details are not important.

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 P_1 and P_2 are known. Under the above assumptions (which do apply to the wp and wlp transformers) we claim:

Theorem 6.2 *There exists no inductive CPET.*

Proof. [Sketch] (for details, see [12, p. 11]). By contradiction: Consider the program $P = \{P_1\}\ [1/2]\ \{P_5\}$ with

$$
\begin{aligned}
P_1 &: \quad x \coloneqq 1 \\
P_5 &: \quad \{P_2\} \,\square\, \{P_4\} \\
P_2 &: \quad x \coloneqq 2 \\
P_4 &: \quad \{\texttt{observe false}\}\ [1/2]\ \{P_3\} \\
P_3 &: \quad x \coloneqq 2.2
\end{aligned}
$$

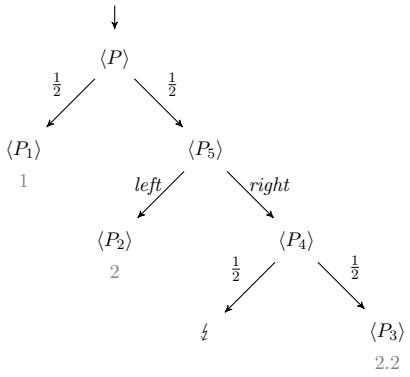


Fig. 3. Schematic depiction of the RMDP $\mathfrak{R}^{x}_{\sigma_0}[\![P]\!]$

A schematic depiction of the $\mathfrak{R}^{x}_{\sigma_0}[\![P]\!]$ is given in Figure 3. Assume there exists an inductive CPET cwp^* over some appropriate domain D. With the program given above, one can get to the contradiction $[\![\mathsf{cwp}^*[P_5]]\!] = [\![\mathsf{cwp}^*[P_4]]\!] > [\![\mathsf{cwp}^*[P_2]]\!] = [\![\mathsf{cwp}^*[P_5]]\!]$. □

As an immediate corollary of Theorem 6.2 we obtain the following result:

Corollary 6.3 *We cannot extend the* cwp *or* cwlp *rules in Figure 1 for non–deterministic programs such that Theorem 4.6 extends to full* cpGCL.

This result is related to Varacca and Winskel's work [27], who have already noticed the difficulties that arise when trying to integrate non–determinism and probabilities, even in the absence of conditioning. When conditioning is taken into account, Andrés and van Rossum [1] have also observed that positional schedulers—i.e. the kind of schedulers implicitly considered in the expectation transformer semantics—are not sufficient for minimizing probabilities. In contrast to our work, their development is done in the context of temporal logics.

7 Conclusion and Future Work

This paper presented an extensive treatment of semantic issues in probabilistic programs with conditioning. Major contributions are the treatment of non–terminating programs (both operationally and for weakest liberal pre–expectations), our results on combining non–determinism with conditioning, as well as the presented program transformation. We firmly believe that a thorough understanding of these semantic issues provides a main cornerstone for enabling automated analysis techniques such as loop invariant synthesis [5,18], program analysis [6] and model checking [3] to the class of probabilistic programs with conditioning. Future work consists of investigating conditional invariants and a further investigation of non–determinism in combination with conditioning.

Acknowledgment

We would like to thank Pedro D'Argenio and Tahiry Rabehaja for the valuable discussions preceding this paper.

References

[1] Andrés, M. E. and P. van Rossum, *Conditional probabilities over probabilistic and nondeterministic systems*, in: *Proc. of TACAS*, LNCS **4963** (2008), pp. 157–172.

[2] Baier, C. and J. Katoen, "Principles of Model Checking," MIT Press, 2008.

[3] Baier, C., J. Klein, S. Klüppelholz and S. Märcker, *Computing conditional probabilities in Markovian models efficiently*, in: *Proc. of TACAS*, LNCS **8413** (2014), pp. 515–530.

[4] Barthe, G., B. Köpf, F. Olmedo and S. Z. Béguelin, *Probabilistic relational reasoning for differential privacy*, ACM Trans. Program. Lang. Syst. **35** (2013), p. 9.

[5] Chakarov, A. and S. Sankaranarayanan, *Expectation invariants for probabilistic program loops as fixed points*, in: *Proc. of SAS*, LNCS **8723** (2014), pp. 85–100.

[6] Cousot, P. and M. Monerau, *Probabilistic abstract interpretation*, in: H. Seidl, editor, *Proc. of ESOP*, LNCS **7211** (2012), pp. 169–193.

[7] Dijkstra, E. W., "A Discipline of Programming," Prentice Hall, 1976.

[8] Goodman, N. D., V. K. Mansinghka, D. M. Roy, K. Bonawitz and J. B. Tenenbaum, *Church: a language for generative models*, in: *Proc. of UAI* (2008), pp. 220–229.

[9] Goodman, N. D. and A. Stuhlmüller, "The Design and Implementation of Probabilistic Programming Languages." (electronic), 2014, http://dippl.org.

[10] Gordon, A. D., T. Graepel, N. Rolland, C. V. Russo, J. Borgström and J. Guiver, *Tabular: a schema-driven probabilistic programming language*, in: *Proc. of POPL* (2014), pp. 321–334.

[11] Gordon, A. D., T. A. Henzinger, A. V. Nori and S. K. Rajamani, *Probabilistic programming*, in: *Proc. of FOSE* (2014), pp. 167–181.

[12] Gretz, F., N. Jansen, B. L. Kaminski, J.-P. Katoen, A. McIver and F. Olmedo, *Conditioning in probabilistic programming*, CoRR **abs/1504.00198** (2015).

[13] Gretz, F., J.-P. Katoen and A. McIver, *Operational versus weakest pre-expectation semantics for the probabilistic guarded command language*, Perform. Eval. **73** (2014), pp. 110–132.

[14] Gupta, V., R. Jagadeesan and P. Panangaden, *Stochastic processes as concurrent constraint programs*, in: *Proc. of POPL* (1999), pp. 189–202.

[15] Gupta, V., R. Jagadeesan and V. A. Saraswat, *Probabilistic concurrent constraint programming*, in: *Concurrency Theory*, LNCS **1243** (1997), pp. 243–257.

[16] Hur, C.-K., A. V. Nori, S. K. Rajamani and S. Samuel, *Slicing probabilistic programs*, in: *Proc. of PLDI* (2014), pp. 133–144.

[17] Jones, C. and G. D. Plotkin, *A probabilistic powerdomain of evaluations*, in: *Logic in Computer Science* (1989), pp. 186–195.

[18] Katoen, J.-P., A. McIver, L. Meinicke and C. C. Morgan, *Linear-invariant generation for probabilistic programs*, in: *Proc. of SAS*, LNCS **6337** (2010), pp. 390–406.

[19] Kozen, D., *Semantics of probabilistic programs*, J. Comput. Syst. Sci. **22** (1981), pp. 328–350.

[20] Kozen, D., *A probabilistic {PDL}*, Journal of Computer and System Sciences **30** (1985), pp. 162 – 178.

[21] McIver, A. and C. Morgan, "Abstraction, Refinement And Proof For Probabilistic Systems," Springer, 2004.

[22] Nori, A. V., C. Hur, S. K. Rajamani and S. Samuel, *R2: an efficient MCMC sampler for probabilistic programs*, in: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014, pp. 2476–2482.

[23] Paige, B. and F. Wood, *A compilation target for probabilistic programming languages*, in: *Proc. of ICML*, JMLR Proceedings **32** (2014), pp. 1935–1943.

[24] Puterman, M., "Markov Decision Processes: Discrete Stochastic Dynamic Programming," John Wiley and Sons, 1994.

[25] Sampson, A., P. Panchekha, T. Mytkowicz, K. S. McKinley, D. Grossman and L. Ceze, *Expressing and verifying probabilistic assertions*, in: *Proc. of PLDI* (2014), p. 14.

[26] Scott, D. S., *Stochastic λ-calculi: An extended abstract*, J. Applied Logic **12** (2014), pp. 369–376.

[27] Varacca, D. and G. Winskel, *Distributing probability over non-determinism*, Mathematical. Structures in Comp. Sci. **16** (2006), pp. 87–113.