CONDITIONING 8

Describing randomized algorithms for solving computationally difficult problems more efficiently on average is a classical use case of probabilistic programs. A more recent use case is found in machine learning: There, probabilistic programs conveniently describe complex probability distributions. The main goal is to have a mathematically rigorous description that is yet easily accessible to a working programmer [Gor+14]. A key ingredient in such descriptions is *conditioning*, which — as the name suggests — allows to describe conditional probability distributions.

As a simple example of such a conditional distribution, suppose that we would like to model that variable x is distributed according to a geometric distribution with parameter $1/2$, conditioned on the event that x is odd. In standard mathematical notation, the probability mass function

$$\mathsf{P}(x = k \mid x \text{ is odd}) = \begin{cases} \frac{3}{2^{k+1}}, & \text{if } k \text{ is odd,} \\ 0, & \text{if } k \text{ is even.} \end{cases}$$

describes the probability that x has value k, *given* that x is odd. Notice that the mathematical representation above completely hides both the *computational aspects* underlying a geometric distribution as well as the *event on which we want to condition*. As for the underlying computational aspect, an (unconditioned) geometric distribution can be described by the following *algorithm*: Keep flipping a fair coin until you throw, say, heads. *Count* in x the number of coin flips you had to perform in order to achieve that goal.

On the positive side, the above mathematical representation is *compact* and *explicit*. Questions, for instance, about probabilities of certain events or about the expected value of x can easily be answered. On the negative side, if we are merely presented with the above mathematical representation of the conditional distribution, it is rather difficult to extract the underlying algorithm that constructed that distribution. That in turn makes it rather difficult to understand the distribution and in particular difficult to adjust it: Say we wanted to adjust the parameter of the geometric distribution to $1/3$ instead of $1/2$. Then the resulting probability mass function is

$$\mathsf{P}'(x = k \mid x \text{ is odd}) = \begin{cases} \frac{2^k \cdot 5}{3^{k+2}}, & \text{if } k \text{ is odd,} \\ 0, & \text{if } k \text{ is even.} \end{cases}$$

However, it is not at all obvious how to obtain P' from P.

Another negative aspect of representing distributions by probability mass functions is that it is not so obvious *how to sample* from a distribution given only its probability mass function. Instead, general purpose sampling algorithms have to be employed. The probability of a specific sample being returned by the sampling algorithm corresponds to the probability specified by the probability mass function it gets as input, however often only up to some precision.

As an alternative, the probabilistic program

```
1: x := 0;
2: c := 1;
3: while(c = 1){
4:     x := x + 1;
5:     {c := 0} [1/2] {c := 1}
6: };
7: observe(x is odd)
```

describes the same conditional distribution P while not hiding the details of its construction: Both the repeated coin flips (Lines 3 to 6) as well as the event on which we condition (Line 7) are *explicitly typed out* in the above program. We can even explicitly see the individual coin flips (Line 5) and the counting (Line 4). We also notice that in the program representation it is completely obvious how to adjust the parameter of the geometric distribution from $1/2$ to $1/3$: We simply replace the $1/2$ by $1/3$ in the program.

As another positive aspect of the program representation, each probability distribution described by a probabilistic program comes with its own special purpose sampling algorithm: *the program itself!* By executing the program once, we effectively obtain one sample from the probability distribution that it implicitly describes.

While the program representation above is easy to understand and easy to sample from, the probability mass function it represents is not as explicitly given as with the probability mass function representation. A main task for probabilistic programs, in particular with conditioning, is thus *inference*, i.e. determining an „explicit representation of the probability distribution implicitly specified by the probabilistic program“ [Gor+14].

A workable approach to inference is determining the expected value of some function f after executing a probabilistic program [Gor+14]. We have already studied in Chapter 4 how this can be accomplished for programs without conditioning by means of the weakest preexpectation calculus. In this chapter, we will extend the weakest preexpectation calculus to reasoning about probabilistic programs *with conditioning*. We will show how previous approaches to the inference task are inferior when it comes to dealing with nontermination and present rules for reasoning about loops.

8.1 cpGCL — pGCL WITH CONDITIONING

In this section, we present syntax and operational semantics of the *probabilistic guarded command language with conditioning*. As for the syntax, we endow pGCL (cf. Section 3.2) with an additional `observe` statement, as we have already done tacitly in the introductory example.

Definition 8.1 (pGCL with Conditioning [Jan+15a; Olm+18]): *Recall* Definition 3.1 *which defines all notions related to the probabilistic guarded command language* pGCL. *The set of programs in probabilistic guarded command language with conditioning, denoted* cpGCL, *is given by the grammar*

$$
\begin{aligned}
C \longrightarrow\ & \texttt{skip} && \text{(effectless program)}\\
\mid\ & \texttt{diverge} && \text{(freeze)}\\
\mid\ & x := E && \text{(assignment)}\\
\mid\ & x :\approx \mu && \text{(random assignment)}\\
\mid\ & \texttt{observe}\,(\varphi) && \text{(conditioning)}\\
\mid\ & C \,\texttt{;}\, C && \text{(sequential composition)}\\
\mid\ & \texttt{if}\,(\varphi)\,\{C\}\,\texttt{else}\,\{C\} && \text{(conditional choice)}\\
\mid\ & \{C\}\,[p]\,\{C\} && \text{(probabilistic choice)}\\
\mid\ & \texttt{while}(\varphi)\{C\}\,, && \text{(while loop)}
\end{aligned}
$$

where $x \in \mathsf{Vars}$ is a program variable, E is an arithmetic expression over program variables, μ is a distribution expression, φ is a boolean expression over program variables guarding a choice or a loop, and p is a probability expression. Recall Definition 3.1 *for the meaning of the above technical terms.*

Note that cpGCL *programs are by definition tame, i.e. they contain no nondeterministic choices.*

Operationally, all instructions of a cpGCL program are executed exactly the same way as pGCL instructions (cf. Section 3.3). The obvious exception are `observe` statements, since these are not part of the pGCL language. The `observe` statements are executed as follows:

When an $\texttt{observe}\,(\varphi)$ instruction is encountered and the current program state is σ, it is checked whether σ satisfies the observation φ, i.e. whether $\sigma \models \varphi$. If so, the computation proceeds as if the $\texttt{observe}\,(\varphi)$ instruction was a `skip` instruction. If, however, $\sigma \not\models \varphi$, the computation terminates *unsuccessfully* in a designated observation violation state $\lightning$.

Formally, the operational behavior of a cpGCL program is given by extending the computation tree semantics of pGCL (see Definition 3.4). In particular, the SOS rules in Figure 3.1 are complemented by two rules for

the observe statement, namely:

$$\frac{\varphi(\sigma) \;=\; \text{true}}{\langle \texttt{observe}\,(\varphi), \sigma, n, \theta, \eta, q\rangle \;\vdash\; \langle \downarrow, \sigma, n+1, \theta, \eta, q\rangle} \text{ (observe1)}$$

$$\frac{\varphi(\sigma) \;=\; \text{false}}{\langle \texttt{observe}\,(\varphi), \sigma, n, \theta, \eta, q\rangle \;\vdash\; \langle \downarrow, \lightning, n+1, \theta, \eta, q\rangle} \text{ (observe2)}$$

For a given cpGCL program C and input σ, any computation path in the computation tree of executing C on input σ has *exactly one of three* forms:

A. The path terminates successfully in some final state $\tau \neq \lightning$.

B. The path terminates unsuccessfully in $\lightning$.

C. The path does not terminate but also does not violate any observation.

Executing C on σ thus gives rise to three disjoint sets of paths which we denote for illustrational purposes by A, B, and C, respectively. A diagrammatic depiction of this situation is shown in Figure 8.1.

Besides a computation tree semantics, we also described the semantics of pGCL programs as inducing a (sub)distribution over final states (see Section 3.3.2). Using the extension of the computation tree semantics described above, we obtain from a cpGCL program a (sub)distribution $[\![C]\!]_\sigma$ over $\Sigma \cup \{\lightning\}$ by applying Definition 3.8. However, for cpGCL programs, we would rather like to describe the *conditional distribution* $[\![C]\!]_\sigma\big|_{\neg\lightning}$ over terminal states, *conditioned* on the event $\neg$B, i.e. on the event that no observation violation occurred during computation [Gor+14]. Note that the event $\neg$B coincides with the event A $\cup$ C, i.e. the event that either the program terminates successfully or not at all. Th conditional distribution $[\![C]\!]_\sigma\big|_{\neg\lightning}$ can be described by

$$[\![C]\!]_\sigma\big|_{\neg\lightning}(\tau) \;=\; \begin{cases} 0, & \text{if } \tau = \lightning \text{ and } [\![C]\!]_\sigma(\lightning) < 1 \\ \frac{[\![C]\!]_\sigma(\tau)}{1-[\![C]\!]_\sigma(\lightning)}, & \text{if } \tau \neq \lightning \text{ and } [\![C]\!]_\sigma(\lightning) < 1 \\ \text{undefined}, & \text{if } [\![C]\!]_\sigma(\lightning) = 1\,. \end{cases}$$

As we can see, the distribution $[\![C]\!]_\sigma\big|_{\neg\lightning}$ is a rather unwieldy object and sometimes even undefined, namely whenever we would have to deal with a division by zero. The expectation transformer based approach to reasoning about cpGCL programs which we present in the remainder of this chapter is more satisfactory in that aspect: It always gives well–defined, meaningful, and expected results, even in the problematic division–by–zero case.

8.2 CONDITIONAL EXPECTATION TRANSFORMERS

Extending weakest preexpectation reasoning for pGCL à la Chapter 4 to cpGCL is the subject matter of this section. This technique will allow us

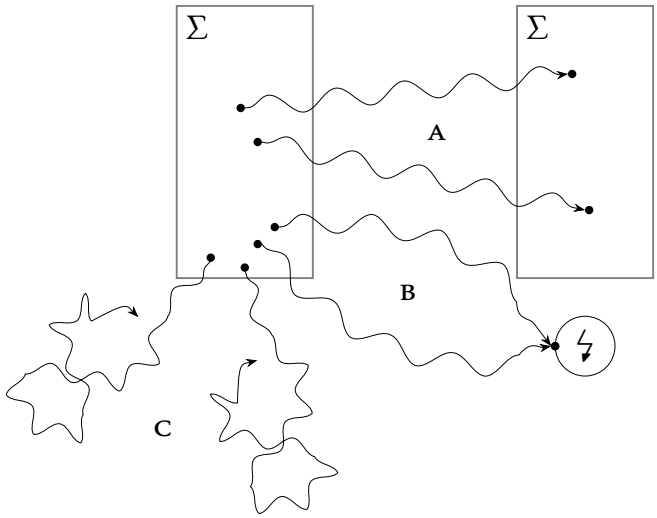

Figure 8.1: Executing a cpGCL program can lead to either one of three outcomes: (A) the program does not violate any observation along its computation and *terminates successfully* in some final state, (B) the program violates an observation along its computation and *terminates unsuccessfully* in the observation failure state $\lightning$, or (C) the program does not violate any observation along its computation but also *does not terminate*.

to reason about *conditional expected values* and *conditional probabilities*. As we did with pGCL, we will present two calculi: the *conditional weakest preexpectation calculus* for total correctness and the *conditional weakest liberal preexpectation calculus* for partial correctness.

8.2.1 *Conditional Weakest Preexpectations*

For a given postexpectation $f \in \mathbb{E}$ and program $C \in \mathsf{cpGCL}$, we are interested in the *conditional expected value of* f after successful termination of C on a given input σ, *given* that *no* observation that is encountered along the computation is violated. More precisely, we are interested in a mathematical object that maps every input σ to the respective conditional expected value. Put in terms of the three sets A, B, and C we described in Section 8.1, we are interested in the quantity

$$\frac{\mathsf{EV}\big(f \cdot [\neg \mathrm{B}]\big)}{\mathsf{Pr}(\neg \mathrm{B})} = \frac{\mathsf{EV}\big(f \cdot [\mathrm{A}] + f \cdot [\mathrm{C}]\big)}{\mathsf{Pr}(\mathrm{A} \cup \mathrm{C})} = \frac{\mathsf{EV}(f \cdot [\mathrm{A}])}{\mathsf{Pr}(\mathrm{A} \cup \mathrm{C})},$$

i.e. the expected value of f, given that no observation is violated. The last equality comes about because all paths in the set c are infinite and hence there exists no path in c that reaches a final state in which f could be evaluated with non–zero probability. The conditional expected value of f described above constitutes our notion of *conditional weakest preexpectations*. By choosing $f = [F]$, i.e. f is the indicator function of an event F, we see that reasoning about conditional weakest preexpectations subsumes reasoning about *conditional probabilities*.

Our general approach will be to calculate the numerator and denominator of the fraction above *separately*, in order not to run into trouble with problematic cases such as for instance „0/0“. Keeping a pair, we can just pair 0 and 0 without running into problems with undefinedness. We also notice that the numerator is a general expected value, whereas the denominator represents a probability. We will hence reason about pairs of expectations in $\mathbb{E} \times \mathbb{E}_{\leq 1}$. We call such pairs *conditional expectations*.

Definition 8.2 (Conditional Expectations [Jan+15a; Olm+18]):

A. *The set of conditional expectations, denoted $\mathbb{C}$, is defined as the set of pairs comprising of a general expectation in $\mathbb{E}$ and a one–bounded expectation in $\mathbb{E}_{\leq 1}$ (cf. Definition 4.1), i.e.*

$$\mathbb{C} \;=\; \mathbb{E} \times \mathbb{E}_{\leq 1} \,.$$

We denote a pair in $\mathbb{C}$ consisting of first component f and second component g by

$$\underline{f \,/\, g}$$

to indicate that it represents a fraction.

A complete lattice on $\mathbb{C}$ is induced by the partial order $\trianglelefteq$, given by

$$\underline{f \,/\, g} \;\trianglelefteq\; \underline{f' \,/\, g'} \qquad \text{iff} \qquad f \preceq f' \quad \text{and} \quad g \succeq g' \,.$$

Notice that the order on the second components is the reversed order of the first components. The least and the greatest element in the complete lattice $(\mathbb{C}, \trianglelefteq)$ is given by

$$\underline{0 \,/\, 1} \quad \text{and} \quad \underline{\infty \,/\, 0} \,,$$

respectively, where $0, 1, \infty \in \mathbb{E}$. The supremum of a subset $S \subseteq \mathbb{C}$ (with respect to the order $\trianglelefteq$) is given pointwise by

$$\sup_{\trianglelefteq} S \;=\; \underline{\sup_{\preceq} \left\{ f \;\middle|\; \underline{f \,/\, g} \in S \right\} \Big/ \inf_{\preceq} \left\{ g \;\middle|\; \underline{f \,/\, g} \in S \right\}} \,,$$

where the supremum and the infimum on the right–hand–side are understood with respect to the order $\preceq$.

B. *For* $\underline{f/g}, \underline{f'/g'} \in \mathbb{C}$ *and* $h \in \mathbb{E}_{\leq 1}$*, we define an addition* $\oplus$ *by*

$$\underline{f/g} \oplus \underline{f'/g'} = \underline{f + f'/g + g'}$$

and likewise a scalar multiplication $\odot$ *by*

$$h \odot \underline{f/g} = \underline{h \cdot f / h \cdot g}\,.$$

Notice that there is a crucial difference between f/g and $\underline{f/g}$: The expression f/g is a pointwise fraction, i.e.

$$\frac{f}{g} = \lambda\sigma.\ \frac{f(\sigma)}{g(\sigma)}\,,$$

which is potentially undefined, namely if $g(\sigma) = 0$. Moreover, we have

$$\frac{1}{1} = \frac{\frac{1}{2}}{\frac{1}{2}}\,.$$

On the other hand, a conditional expectation $\underline{f/g}$ is merely a *pair* of expectations, which is *interpreted* as a fraction:

$$\underline{f/g} \qquad \text{is interpreted as} \qquad \lambda\sigma.\ \begin{cases} \frac{f(\sigma)}{g(\sigma)}, & \text{if } g(\sigma) \neq 0 \\ \text{undefined}, & \text{if } g(\sigma) = 0. \end{cases}$$

However, formally, a conditional expectation is *not* a fraction. In particular,

$$\underline{1/1} \neq \underline{1/2 / 1/2}\,.$$

As conditional expectations are just pairs, $\underline{0/0}$ is a perfectly well–defined mathematical object, whereas the definedness of $0/0$ is at least controversial.

The partial order $\trianglelefteq$ on conditional expectations enables monotonic reasoning about total correctness. The order corresponds naturally to our „fractional interpretation“ in the sense that for all states σ we have that

$$\underline{f/g} \trianglelefteq \underline{f'/g'} \qquad \text{implies} \qquad \frac{f(\sigma)}{g(\sigma)} \leq \frac{f'(\sigma)}{g'(\sigma)}$$

in case that both fractions are defined (i.e. in case that $g'(\sigma) > 0$, which by $g \geq g'$ implies that $g(\sigma) > 0$). This means that overapproximations in the sense of the partial order $\trianglelefteq$ are indeed overapproximations of the sought–after conditional expected value.

For reasoning about conditional expected values yielded by cpGCL programs, we define an expectation transformer that acts on conditional expectations, i.e. on $\mathbb{C}$, as follows:

C	$\mathsf{cwp}\,[C]\left(\frac{f}{g}\right)$
`skip`	$\frac{f}{g}$
`diverge`	$\frac{0}{1}$
$x := E$	$\frac{f[x/E]}{g[x/E]}$
$x :\approx \mu$	$\frac{\lambda\sigma.\ \int_{\mathsf{Vals}} \left(\lambda v.\ f\left(\sigma[x \mapsto v]\right)\right) d\,\mu_\sigma}{\lambda\sigma.\ \int_{\mathsf{Vals}} \left(\lambda v.\ g\left(\sigma[x \mapsto v]\right)\right) d\,\mu_\sigma}$
`observe`(φ)	$[\varphi] \odot \frac{f}{g}$
$C_1 \fatsemi C_2$	$\mathsf{cwp}\,[\![C_1]\!]\left(\mathsf{cwp}\,[\![C_2]\!]\left(\frac{f}{g}\right)\right)$
`if`$(\varphi)\{C_1\}$`else`$\{C_2\}$	$[\varphi] \odot \mathsf{cwp}\,[\![C_1]\!]\left(\frac{f}{g}\right) \oplus [\neg\varphi] \odot \mathsf{cwp}\,[\![C_2]\!]\left(\frac{f}{g}\right)$
$\{C_1\}\,[p]\,\{C_2\}$	$p \odot \mathsf{cwp}\,[\![C_1]\!]\left(\frac{f}{g}\right) \oplus (1-p) \odot \mathsf{cwp}\,[\![C_2]\!]\left(\frac{f}{g}\right)$
`while`$(\varphi)\{C'\}$	$\mathsf{lfp}_{\trianglelefteq}\ \frac{X}{Y}.\ [\neg\varphi] \odot \frac{f}{g} \oplus [\varphi] \odot \mathsf{cwp}\,[\![C']\!]\left(\frac{X}{Y}\right)$

Table 8.1: The conditional weakest preexpectation transformer. The least fixed point for the while loop is understood in terms of the partial order $\trianglelefteq$.

DEFINITION 8.3 (Conditional wp [Jan+15a; Olm+18]):

A. *The conditional weakest preexpectation transformer*

$$\mathsf{cwp}\,[\![C]\!]\colon \quad \mathbb{C} \to \mathbb{C}$$

is defined according to the rules in Table 8.1.

B. *We call the function*

$$\overset{\mathsf{cwp}}{\langle\varphi,C\rangle}\Phi_{\frac{f}{g}}\left(\frac{X}{Y}\right) = [\neg\varphi] \odot \frac{f}{g} \oplus [\varphi] \cdot \mathsf{cwp}\,[\![C]\!]\left(\frac{X}{Y}\right)$$

the cwp–characteristic function of `while`$(\varphi)\{C\}$ *with respect to postexpectation* $\frac{f}{g}$*. If either of* cwp*,* φ*,* C*, or* $\frac{f}{g}$ *are clear from the context, we omit them from* Φ*.*

The rules for the cwp transformer basically calculate wp in the first component and wlp in the second component (cf. Section 4.1). For the `observe`(φ) statement, the indicator function of the observed evidence φ is multiplied to both components. In effect that reduces the expected value in the first component (the numerator) as well as the normalization factor in the second component (the denominator). We will study how these rules behave for loops in more detail later in this chapter.

If we want to use the cwp calculus to reason about the conditional expected value of f after executing program C, given that no observation is violated, we have to determine $\mathsf{cwp}\,[\![C]\!]\left(\frac{f}{1}\right)$. Below, we give an example:

EXAMPLE 8.4 (Conditional Weakest Preexpectation Reasoning [Olm+18]): Assume we want to compute the expected value of the expression $10+x$ after executing program C given by

$$
\begin{aligned}
&\{x := 0\}\,[1/2]\,\{x := 1\}\,;\\
&\texttt{if}\,(x = 1)\,\{\\
&\qquad \{y := 0\}\,[1/2]\,\{y := 2\}\\
&\}\texttt{else}\{\\
&\qquad \{y := 0\}\,[4/5]\,\{y := 3\}\\
&\}\,;\\
&\texttt{observe}\,(y = 0)
\end{aligned}
$$

That means, we have to reason about $\mathrm{cwp}\,[\![C]\!]\left(\frac{10+x}{1}\right)$. Reusing our annotation style from earlier in this thesis (see Example 2.11), i.e. we write

$$
\begin{aligned}
&/\!/\!/\ \frac{f''}{g''}\\
&/\!/\!/\ \frac{f'}{g'}\\
&C\\
&/\!/\!/\ \frac{f}{g}
\end{aligned}
$$

to expresses that $\frac{f'}{g'} = \mathrm{cwp}\,[\![C]\!]\left(\frac{f}{g}\right)$ and moreover that $\frac{f''}{g''} = \frac{f'}{g'}$, we can annotate the above program as shown in Figure 8.2 (read from bottom to top). The calculation of $\mathrm{cwp}\,[\![C]\!]\left(\frac{10+x}{1}\right)$ gives $\frac{27/4}{13/20}$. As for an *interpretation*, we can say that the conditional expected value of $10+x$, given that the observation is not violated, is for any initial state

$$
\frac{\frac{27}{4}}{\frac{13}{20}} = \frac{27\cdot 20}{13\cdot 4} = \frac{135}{13} \approx 10.38\,.
$$

8.2.2 *Conditional Weakest Liberal Preexpectations*

While we considered total correctness above, we now consider partial correctness: Given event F and program C, we ask: What is the *conditional probability* that C either diverges or terminates in a state satisfying F, *given* that no observation encountered along the computation is violated. Put in terms of the three sets A, B, and C (see Section 8.1), we are interested in

$$
\frac{\mathsf{Pr}\big((F \cup \mathrm{C}) \cap \neg \mathrm{B}\big)}{\mathsf{Pr}(\neg \mathrm{B})} = \frac{\mathsf{Pr}(F \cup \mathrm{C})}{\mathsf{Pr}(\mathrm{A} \cup \mathrm{C})}\,.
$$

$$/\!/\!/\ \underline{\tfrac{27}{4}\big/\tfrac{13}{20}}$$

$$/\!/\!/\ \underline{4\big/\tfrac{4}{10}} \oplus \underline{\tfrac{11}{4}\big/\tfrac{1}{4}}$$

$$/\!/\!/\ \tfrac{4}{10} \odot \underline{10\big/1} \oplus \tfrac{1}{4} \odot \underline{11\big/1}$$

$$/\!/\!/\ \tfrac{1}{2} \odot \tfrac{4}{5} \odot \underline{10\big/1} \oplus \tfrac{1}{2} \odot \tfrac{1}{2} \odot \underline{11\big/1}$$

$$/\!/\!/\ \tfrac{1}{2} \odot \Big([0=1] \odot \tfrac{1}{2} \odot \underline{10+0\big/1} \oplus [0\neq 1] \odot \tfrac{4}{5} \odot \underline{10+0\big/1}\Big)$$
$$\oplus\ \tfrac{1}{2} \odot \Big([1=1] \odot \tfrac{1}{2} \odot \underline{10+1\big/1} \oplus [1\neq 1] \odot \tfrac{4}{5} \odot \underline{10+1\big/1}\Big)$$

$\{x := 0\}\ [1/2]\ \{x := 1\}$;

$$/\!/\!/\ [x=1] \odot \tfrac{1}{2} \odot \underline{10+x\big/1} \oplus [x\neq 1] \odot \tfrac{4}{5} \odot \underline{10+x\big/1}$$

if $(x = 1)$ {

$$/\!/\!/\ \tfrac{1}{2} \odot \underline{10+x\big/1}$$

$$/\!/\!/\ \tfrac{1}{2} \odot [0=0] \odot \underline{10+x\big/1} \oplus \tfrac{1}{2} \odot [2=0] \odot \underline{10+x\big/1}$$

$\{y := 0\}\ [1/2]\ \{y := 2\}$

$$/\!/\!/\ [y=0] \odot \underline{10+x\big/1}$$

} else {

$$/\!/\!/\ \tfrac{4}{5} \odot \underline{10+x\big/1}$$

$$/\!/\!/\ \tfrac{4}{5} \odot [0=0] \odot \underline{10+x\big/1} \oplus \tfrac{1}{5} \odot [3=0] \odot \underline{10+x\big/1}$$

$\{y := 0\}\ [4/5]\ \{y := 3\}$

$$/\!/\!/\ [y=0] \odot \underline{10+x\big/1}$$

};

$$/\!/\!/\ [y=0] \odot \underline{10+x\big/1}$$

observe $(y = 0)$

$$/\!/\!/\ \underline{10+x\big/1}$$

Figure 8.2: Conditional weakest preexpectation annotations for Example 8.4.

The equality comes about because all computation paths that terminate successfully do not terminate in the $\lightning$ state. The conditional probability above constitutes our notion of *conditional weakest liberal preexpectations*.

Again, our approach will be to calculate the numerator and the denominator of the above fraction separately. We notice that both the numerator as well as the denominator represent probabilities. We will hence reason about pairs of expectations in $\mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1}$. We call such pairs *one–bounded conditional expectations*.

DEFINITION 8.5 (One–bounded Cond. Expect. [Jan+15a; Olm+18]**):**

A. *The set of one–bounded conditional expectations, denoted $\mathbb{C}_{\leq 1}$, is defined as the set of conditional expectations, where both components are one–bounded expectations, i.e.*

$$\mathbb{C}_{\leq 1} \;=\; \mathbb{E}_{\leq 1} \times \mathbb{E}_{\leq 1}\,.$$

Obviously, we have $\mathbb{C}_{\leq 1} \subset \mathbb{C}$. We use the same notation for pairs in $\mathbb{C}_{\leq 1}$ as for pairs in $\mathbb{C}$. A complete lattice on $\mathbb{C}_{\leq 1}$ is induced by the partial order $\trianglelefteq$, given by

$$\underline{f \,/\, g} \;\trianglelefteq\; \underline{f' \,/\, g'} \qquad \text{iff} \qquad f \preceq f' \quad \text{and} \quad g \preceq g'\,.$$

Notice that (in contrast to the order $\trianglelefteq$ on $\mathbb{C}$) the order on the second components is the same as the order of the first components. The least and the greatest element in the complete lattice $(\mathbb{C}_{\leq 1}, \trianglelefteq)$ are given by

$$\underline{0 \,/\, 0} \quad \text{and} \quad \underline{1 \,/\, 1}\,,$$

respectively, where $0, 1 \in \mathbb{E}_{\leq 1}$. The supremum of a subset $S \subseteq \mathbb{C}_{\leq 1}$ (with respect to the order $\trianglelefteq$) is constructed pointwise by

$$\sup_{\trianglelefteq} S \;=\; \underline{\sup_{\preceq} \left\{ f \;\middle|\; \underline{f \,/\, g} \in S \right\} \Big/ \sup_{\preceq} \left\{ g \;\middle|\; \underline{f \,/\, g} \in S \right\}}\,,$$

where the two suprema on the right–hand–side are understood with respect to the partial order $\preceq$.

The partial order $\trianglelefteq$ will be used for defining conditional weakest liberal preexpectations as greatest fixed points. Unfortunately, the partical order $\trianglelefteq$ does not correspond as naturally to our „fractional interpretation“ as the partial order $\trianglelefteq$ on $\mathbb{C}$ did. This is because we have that

$$\underline{f \,/\, g} \;\trianglelefteq\; \underline{f' \,/\, g'} \quad \text{neither implies} \quad \frac{f(\sigma)}{g(\sigma)} \leq \frac{f'(\sigma)}{g'(\sigma)} \quad \text{nor} \quad \frac{f(\sigma)}{g(\sigma)} \geq \frac{f'(\sigma)}{g'(\sigma)}\,.$$

C	$\mathsf{cwlp}\,[\![C]\!]\left(\underline{f/g}\right)$
`skip`	$\underline{f/g}$
`diverge`	$\underline{1/1}$
$x := E$	$\underline{f[x/E]/g[x/E]}$
$x :\approx \mu$	$\underline{\lambda\sigma.\ \int_{\mathsf{Vals}}\left(\lambda v.\ f(\sigma[x\mapsto v])\right)d\,\mu_\sigma \,/\, \lambda\sigma.\ \int_{\mathsf{Vals}}\left(\lambda v.\ g(\sigma[x\mapsto v])\right)d\,\mu_\sigma}$
`observe`$\,(\varphi)$	$[\varphi]\odot\underline{f/g}$
$C_1\,;\,C_2$	$\mathsf{cwlp}\,[\![C_1]\!]\left(\mathsf{cwlp}\,[\![C_2]\!]\left(\underline{f/g}\right)\right)$
`if`$\,(\varphi)\,\{C_1\}\,$`else`$\,\{C_2\}$	$[\varphi]\odot\mathsf{cwlp}\,[\![C_1]\!]\left(\underline{f/g}\right)\ \oplus\ [\neg\varphi]\odot\mathsf{cwlp}\,[\![C_2]\!]\left(\underline{f/g}\right)$
$\{C_1\}\,[p]\,\{C_2\}$	$p\odot\mathsf{cwlp}\,[\![C_1]\!]\left(\underline{f/g}\right)\ \oplus\ (1-p)\odot\mathsf{cwlp}\,[\![C_2]\!]\left(\underline{f/g}\right)$
`while`$(\varphi)\{C'\}$	$\mathsf{gfp}_{\trianglelefteq}\ \underline{X/Y}.\ [\neg\varphi]\odot\underline{f/g}\ \oplus\ [\varphi]\odot\mathsf{cwlp}\,[\![C']\!]\left(\underline{X/Y}\right)$

Table 8.2: The conditional weakest liberal preexpectation transformer. The greatest fixed point for the while loop is understood in terms of the partial order $\trianglelefteq$.

Thus, the partial order $\trianglelefteq$ is not as suitable for monotonic reasoning about conditional probabilities. While the transformer we present shortly will indeed be $\trianglelefteq$–monotonic, this is hardly of any use because over– or underapproximating a result with respect to $\trianglelefteq$ does not necessarily give an over– or underapproximation of the sought–after conditional probability.

We now present the conditional weakest liberal preexpectation transformer that acts on $\mathbb{C}_{\leq 1}$ and enables reasoning about conditional probabilities.

Definition 8.6 (Conditional wlp [Jan+15a; Olm+18]):
The conditional weakest liberal preexpectation transformer

$$\mathsf{cwlp}\,[\![C]\!]\colon\quad \mathbb{C}_{\leq 1}\to\mathbb{C}_{\leq 1}$$

is defined according to the rules in Table 8.2.

The rules for the cwlp transformer basically calculate wlp in both components (cf. Section 4.1.2). As with cwp, for the `observe`$\,(\varphi)$ statement, the indicator function of φ is multiplied to both components, which in effect reduces the probability in the first component (the numerator) as well as the normalization factor in the second component (the denominator). We will study how cwlp behaves for loops in more detail in the next section.

If we want to use the cwlp calculus to reason about the conditional probability that an event F is established after executing program C, given that no observation is violated, we have to determine $\mathsf{cwlp}\,[\![C]\!]\left(\underline{[F]/1}\right)$.

8.3 CONDITIONING AND LOOPS

The interplay of conditioning and loops is a particularly intricate matter when attempting to give semantics to probabilistic programs with conditioning. Particular care must be taken, for instance, when conditioning *inside* a loop. However, even by syntactically forbidding conditioning inside loops we do not mitigate all problems.

8.3.1 *The* cwp *Interpretation for Total Correctness*

We first turn our attention to how our cwp transformer for total correctness behaves for loops. As an admittedly pointed — but on the other hand very demonstrative — example, consider the program C, given by

$$
\begin{aligned}
&x := 1 \,⨟\\
&\texttt{while}(x = 1)\{\\
&\quad \{x := 1\}\,[1/2]\,\{x := 0\}\\
&\quad \texttt{observe}\,(x = 1)\\
&\}\,.
\end{aligned}
$$

This program has exactly one diverging run, namely the one in which x is set to 1 infinitely often. This run occurs with probability 0. Inside the loop, x is set to 1 or 0 each with probability a half, but thereafter, we condition on the event that x was set to 1. In effect, we thus condition on the only diverging run, i.e. on an event that occurs with probability 0.

If we now ask, for instance, for the conditional probability that the above program terminates, given that no observation is violated, we would expect this to be the undefined fraction „0/0“. To see that our transformer indeed behaves as expected, let us reason about cwp $[\![C]\!]\left(\underline{1}/\underline{1}\right)$. Since the cwp transformer is backward–moving, we first need to study how our transformer behaves on the loop of program C. For that, we will perform the fixed point iteration for the cwp–characteristic function Φ of the loop with respect to postexpectation $\underline{1}/\underline{1}$, given by

$$
\Phi\left(\underline{X}/\underline{Y}\right) \;=\; [x \neq 1] \odot \underline{1}/\underline{1} \;\oplus\; [x = 1] \odot \tfrac{1}{2} \odot \underline{X[x/1]}/\underline{Y[x/1]}\,.
$$

Iterating Φ on the least element $\underline{0}/\underline{1}$ then gives:

$$
\begin{aligned}
\Phi\left(\underline{0}/\underline{1}\right) \;&=\; [x \neq 1] \odot \underline{1}/\underline{1} \;\oplus\; [x = 1] \odot \tfrac{1}{2} \odot \underline{0[x/1]}/\underline{1[x/1]}\\
&=\; [x \neq 1] \odot \underline{1}/\underline{1} \;\oplus\; [x = 1] \odot \underline{0}/\underline{\tfrac{1}{2}}
\end{aligned}
$$

$$
\begin{aligned}
\Phi^2\left(\underline{0}/\underline{1}\right) \;=\; &[x \neq 1] \odot \underline{1}/\underline{1}\\
&\oplus\; [x = 1] \odot \tfrac{1}{2} \odot \left([1 \neq 1] \odot \underline{1}/\underline{1} \;\oplus\; [1 = 1] \odot \underline{0}/\underline{\tfrac{1}{2}}\right)
\end{aligned}
$$

$$
\begin{aligned}
&= [x \neq 1] \odot \underline{1 \,/\, 1} \;\oplus\; [x = 1] \odot \underline{0 \,/\, \tfrac{1}{4}} \\
\Phi^3\left(\underline{0 \,/\, 1}\right) &= [x \neq 1] \odot \underline{1 \,/\, 1} \\
&\qquad \oplus\; [x = 1] \odot \tfrac{1}{2} \odot \left([1 \neq 1] \odot \underline{1 \,/\, 1} \;\oplus\; [1 = 1] \odot \underline{0 \,/\, \tfrac{1}{4}} \right) \\
&= [x \neq 1] \odot \underline{1 \,/\, 1} \;\oplus\; [x = 1] \odot \underline{0 \,/\, \tfrac{1}{8}} \\
&\;\;\vdots \\
\Phi^n\left(\underline{0 \,/\, 1}\right) &= [x \neq 1] \odot \underline{1 \,/\, 1} \;\oplus\; [x = 1] \odot \underline{0 \,/\, \tfrac{1}{2^n}}, \quad \text{for } n \geq 1 \\
&= \underline{[x \neq 1] \,/\, [x \neq 1] \;+\; [x = 1] \cdot \tfrac{1}{2^n}} \\
&\;\;\vdots \\
\Phi^\omega\left(\underline{0 \,/\, 1}\right) &= \underline{[x \neq 1] \,/\, [x \neq 1]}
\end{aligned}
$$

The last line is the conditional weakest preexpectation of the loop. Finally, we have to calculate

$$\mathsf{cwp}\,[\![x := 1]\!]\left(\underline{[x \neq 1] \,/\, [x \neq 1]}\right) \;=\; \underline{0 \,/\, 0}\,,$$

which is the conditional weakest preexpectation of the whole program C.

As we can see, we get the conditional expectation $\underline{0 \,/\, 0}$ as the result of our calculations, which corresponds exactly to what we would expect, namely „0/0“, except that our preexpectation is a a perfectly well–defined mathematical object. Our conditional preexpectation $\underline{0 \,/\, 0}$ tells us on the one hand that the probability of terminating and not violating any observations is 0, and on the other hand that the probability to not violate any observations is also 0.

In order to understand how the cwp transformer behaves for loops on a more abstract level, let us revisit our definition of the cwp transformer for loops and our definition of the order $\trianglelefteq$. By closer inspection, we can notice that for the fixed point iteration of the cwp–characteristic function we have

$$\underset{\langle \varphi, C \rangle}{\mathsf{cwp}}\Phi^n_{\underline{f \,/\, g}}\left(\underline{0 \,/\, 1}\right) \;=\; \underline{\underset{\langle \varphi, C \rangle}{\mathsf{wp}}\Phi^n_f(0) \,\Big/\, \underset{\langle \varphi, C \rangle}{\mathsf{wlp}}\Phi^n_g(1)}\,.$$

Thus, the first components of the chain

$$\underline{0 \,/\, 1} \;\trianglelefteq\; \underset{\langle \varphi, C \rangle}{\mathsf{cwp}}\Phi_{\underline{f \,/\, g}}\left(\underline{0 \,/\, 1}\right) \;\trianglelefteq\; \underset{\langle \varphi, C \rangle}{\mathsf{cwp}}\Phi^2_{\underline{f \,/\, g}}\left(\underline{0 \,/\, 1}\right) \;\trianglelefteq\; \cdots$$

give the ascending chain

$$0 \;\preceq\; \underset{\langle \varphi, C \rangle}{\mathsf{wp}}\Phi_f(0) \;\preceq\; \underset{\langle \varphi, C \rangle}{\mathsf{wp}}\Phi^2_f(0) \;\preceq\; \underset{\langle \varphi, C \rangle}{\mathsf{wp}}\Phi^3_f(0) \;\preceq\; \cdots$$

and the second components give the descending chain

$$1 \;\geq\; {}^{\mathsf{wlp}}_{\langle \varphi, C\rangle}\Phi_g(1) \;\geq\; {}^{\mathsf{wlp}}_{\langle \varphi, C\rangle}\Phi^2_g(1) \;\geq\; {}^{\mathsf{wlp}}_{\langle \varphi, C\rangle}\Phi^3_g(1) \;\geq\; \cdots\,.$$

For the above to make sense and be well–defined, we need to define both transformers $\mathsf{wp}\,[\![\texttt{observe}\,(\varphi)]\!]$ and $\mathsf{wlp}\,[\![\texttt{observe}\,(\varphi)]\!]$. We can do this by

$$\mathsf{wp}\,[\![\texttt{observe}\,(\varphi)]\!]\,(f) \;=\; [\varphi]\cdot f \;=\; \mathsf{wlp}\,[\![\texttt{observe}\,(\varphi)]\!]\,(f)\,.$$

When taking the limits of the latter two chains above, we see that our definition of cwp corresponds to a

$$\frac{\mathsf{wp}\,[\![C]\!]\,(f)}{\mathsf{wlp}\,[\![C]\!]\,(1)}$$

interpretation of conditional expected values, i.e. we indeed normalize only on the probability of not violating any observations and explicitly account for diverging runs that do not violate any observations.

Because of the difficulties that arise when trying to understand conditioning *within* loops, some authors have the opinion that conditioning within loops should be forbidden altogether. This, however, does not eradicate the need to exercise great caution when combining loops and conditioning. Consider for that the, again very unsubtle yet descriptive, program C', given by

$$\{x := 2\}\,[1/2]\,\{\texttt{diverge}\}\,.$$

This program has exactly one diverging run, namely the one in which `diverge` is executed. This run occurs with probability $1/2$. As the program is `observe`–free, the probability to not violate any observation is 1.

If we now for instance ask for the conditional expected value of x, given that no observation is violated, we would thus expect this to be

$$\frac{\overbrace{\tfrac{1}{2}\cdot 2}^{\text{EV of } x \text{ in non–diverging run}} \;+\; \overbrace{\tfrac{1}{2}\cdot 0}^{\text{EV of } x \text{ in diverging run}}}{\underbrace{1}_{\text{Probability of no observation violation}}} \;=\; \frac{1}{1} \;=\; 1\,.$$

Indeed, if we use the cwp transformer and calculate $\mathsf{cwp}\,[\![C']\!]\left(\underline{x}/\underline{1}\right)$, this yields

$$\mathsf{cwp}\,[\![C']\!]\left(\underline{x}/\underline{1}\right) \;=\; \tfrac{1}{2}\odot\left(\underline{2}/\underline{1}\oplus\underline{0}/\underline{1}\right) \;=\; \underline{1}/\underline{1}\,.$$

Since the program C' is `observe`–free, we would expect that the result from our cwp transformer is backward–compatible to what the standard wp transformer would yield. And indeed, the interpretation of $\underline{1}/\underline{1}$, namely as $1/1 = 1$, agrees with $\mathsf{wp}\,[\![C']\!]\,(x) = 1$.

8.3.2 *The Nori Interpretation*

While we just saw that our cwp transformer gives a reasonable, expected, and backward–compatible result when combining conditioning and loops, previous attempts to give semantics to probabilistic programs would have yielded a different result. The semantics of Nori *et al.* [Nor+14] follows a

$$\frac{\mathsf{wp}\,[\![C]\!]\,(f)}{\mathsf{wp}\,[\![C]\!]\,(1)}$$

interpretation (in fact: Nori *et al.* define the semantics of a probabilistic program with conditioning to be *the fraction* above), i.e. Nori *et al.* normalize on the probability of not violating any observations *and successfully terminating*. We call this the *Nori interpretation* of conditional expected values. Put in terms of the three sets A, B, and C we described in Section 8.1, the Nori interpretation expresses the quantity

$$\frac{\mathsf{EV}\big(f\cdot[\neg \mathrm{B}]\big)}{\mathsf{Pr}(\neg \mathrm{B}\cap \mathrm{A})} = \frac{\mathsf{EV}\big(f\cdot[\mathrm{A}]+f\cdot[\mathrm{C}]\big)}{\mathsf{Pr}\big((\mathrm{A}\cup \mathrm{C})\cap \mathrm{A}\big)} = \frac{\mathsf{EV}(f\cdot[\mathrm{A}])}{\mathsf{Pr}(\mathrm{A})}\,,$$

which as we can see does not renormalize to the event that no observation was violated, but instead to the potentially *less likely* event that the program terminates successfully.

For the above–mentioned program C', given by

$$\{x := 2\}\,[1/2]\,\{\texttt{diverge}\}\,,$$

the Nori interpretation would yield

$$\frac{\overbrace{\frac{1}{2}\cdot 2}^{\text{EV of } x \text{ in non–diverging run}} + \overbrace{\frac{1}{2}\cdot 0}^{\text{EV of } x \text{ in diverging run}}}{\underbrace{\frac{1}{2}}_{\substack{\text{Probability of no observation violation}\\ \text{and termination}}}} = \frac{1}{\frac{1}{2}} = 2$$

as the conditional expected value of x. The semantics of Nori *et al.* is hence not backward–compatible on `observe`–free programs, since

$$\mathsf{wp}\,[\![C']\!]\,(x) = 1 \neq 2\,.$$

Instead, the Nori semantics is only *conditionally backward–compatible*, namely whenever the program in question terminates almost–surely. We believe that this is undesirable. If we believe that conditioning should renormalize to *all* runs that do not violate any observation, the cwp interpretation should be preferred over the Nori interpretation.

Another undesirable fact is that the Nori interpretation impedes monotonic reasoning: Overapproximating $^{\mathsf{wp}[\![C]\!](f)}/_{\mathsf{wp}[\![C]\!](1)}$ calls for overapproximating $\mathsf{wp}[\![C]\!](f)$ and underapproximating $\mathsf{wp}[\![C]\!](1)$. The latter, however, is difficult, as we have extensively discussed through Sections 5.2.3 to 5.2.6.

Overapproximating the result of the cwp interpretation, on the other hand, is easier: Overapproximating $^{\mathsf{wp}[\![C]\!](f)}/_{\mathsf{wlp}[\![C]\!](1)}$ calls for overapproximating $\mathsf{wp}[\![C]\!](f)$ and underapproximating $\mathsf{wlp}[\![C]\!](1)$, both of which can be done by means of simple invariant–based techniques, see Sections 5.2.1 and 5.2.2. We discuss this further in Section 8.6.

As another descriptive example, consider the program `diverge`. We have

$$\mathsf{cwp}[\![\texttt{diverge}]\!]\left(\underline{f/1}\right) = \underline{0/1}\,.$$

The 0 comes from the fact that the program diverges and hence cannot produce any mass contributing to an expected value of f evaluated in a final state. The 1 on the other hand also comes from the fact that `diverge` is a shorthand for `while(true){skip}`, which is `observe`–free and thus the probability to not violate any observation while executing this loop is 1. Our approach thus yields a $0/1$–interpretation for the conditional expected value of f, given that no observation is violated, whereas the Nori interpretation would leave us with the problematic „0%“ case. Indeed, Nori *et al.* would in this case define the semantics of `diverge` to be the *undefined fraction* 0%.

8.3.3 *The* cwlp *Interpretation for Partial Correctness*

Besides total correctness, we also address partial correctness by means of the cwlp transformer. Nori *et al.* do not consider this. Again revisiting the program `diverge`, our cwlp transformer yields

$$\mathsf{cwlp}[\![\texttt{diverge}]\!]\left(\underline{[F]/1}\right) = \underline{1/1}\,.$$

The first 1 now comes from the fact that the probability that `diverge` either „establishes F“ or diverges while at the same time not violating any observations is 1. The second 1 comes just from the probability to not violate any observation. Our approach thus yields a $1/1$–interpretation for the conditional probability to either diverge or terminate successfully and establish event F, given that no observation is violated.

Again, in order to understand how the cwlp transformer behaves for loops on a more abstract level, let us revisit our definition of the cwlp transformer for loops and our definition of the order $\trianglelefteq$. By closer inspection, we can notice that the fixed point iteration of the cwlp–characteristic function gives

$$\underset{\langle\varphi,C\rangle}{\mathsf{cwlp}}\Phi^n_{\underline{f/g}}\left(\underline{0/1}\right) = \underline{\underset{\langle\varphi,C\rangle}{\mathsf{wlp}}\Phi^n_f(0)}\Big/\overline{\underset{\langle\varphi,C\rangle}{\mathsf{wlp}}\Phi^n_g(1)}\,.$$

Thus, the first component of the chain

$$\underline{1/1} \;\preceq\; {}^{\mathsf{cwlp}}_{\langle\varphi,C\rangle}\Phi_{\underline{f/g}}\left(\underline{1/1}\right) \;\preceq\; {}^{\mathsf{cwlp}}_{\langle\varphi,C\rangle}\Phi^2_{\underline{f/g}}\left(\underline{1/1}\right) \;\preceq\; \cdots$$

gives the descending chain

$$1 \;\geq\; {}^{\mathsf{wlp}}_{\langle\varphi,C\rangle}\Phi_f(1) \;\geq\; {}^{\mathsf{wlp}}_{\langle\varphi,C\rangle}\Phi^2_f(1) \;\geq\; {}^{\mathsf{wlp}}_{\langle\varphi,C\rangle}\Phi^3_f(1) \;\geq\; \cdots$$

and the second component gives the descending chain

$$1 \;\geq\; {}^{\mathsf{wlp}}_{\langle\varphi,C\rangle}\Phi_g(1) \;\geq\; {}^{\mathsf{wlp}}_{\langle\varphi,C\rangle}\Phi^2_g(1) \;\geq\; {}^{\mathsf{wlp}}_{\langle\varphi,C\rangle}\Phi^3_g(1) \;\geq\; \cdots\,.$$

When taking the limits of the latter two chains, we see that our definition of cwlp corresponds to a

$$\frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{\mathsf{wlp}\,[\![C]\!]\,(1)}$$

interpretation of conditional weakest liberal preexpectations, i.e. we indeed consider the probability to either diverge or establish some event and normalize only on the probability of not violating any observations, explicitly accounting for diverging runs that do not violate any observations.

8.3.4 *A Fourth Interpretation*

So far, we have seen three possibilities to combine wp and wlp into a fraction. We have also studied under which circumstances they make sense and how the quantity they express should be interpreted. There exists a fourth possibility to combine wp and wlp into a fraction, namely

$$\frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{\mathsf{wp}\,[\![C]\!]\,(1)}\,.$$

We have not studied that possibility yet — and for a good reason: This fourth interpretation is not meaningful. As for an illustration, consider the program

$$\{\texttt{skip}\}\,[1/2]\,\{\texttt{diverge}\}\,.$$

If we now ask, for instance, for the conditional *probability* of diverging or terminating, given that the program does not violate any observations and terminates, the above interpretation would yield

$$\frac{\overbrace{\tfrac{1}{2}\cdot 1}^{\text{Prob. of div. or term. in non–div. run}} + \overbrace{\tfrac{1}{2}\cdot 1}^{\text{Prob. of div. or term. in div. run}}}{\underbrace{\tfrac{1}{2}}_{\text{Probability of no observation violation and termination}}} \;=\; \frac{1}{\frac{1}{2}} \;=\; 2\,,$$

The cwp interpretation:	**The Nori interpretation:**
$\frac{\mathsf{wp}\,[\![C]\!]\,(f)}{\mathsf{wlp}\,[\![C]\!]\,(1)}$	$\frac{\mathsf{wp}\,[\![C]\!]\,(f)}{\mathsf{wp}\,[\![C]\!]\,(1)}$
(for general total correctness)	(for total correctness, *only for a.–s. terminating programs*)
The cwlp interpretation:	**The fourth interpretation:**
$\frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{\mathsf{wlp}\,[\![C]\!]\,(1)}$	$\frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{\mathsf{wp}\,[\![C]\!]\,(1)}$
(for partial correctness)	(*nonsensical, can yield probabilities* > 1)

Figure 8.3: The four possibilities of combining wp and wlp in order to make up a conditional expected value of f after executing program C.

which is a „probability“ larger than 1 and hence nonsensical.

Put in terms of the three sets A, B, and C we described in Section 8.1, the fourth interpretation expresses the quantity

$$\frac{\mathsf{Pr}\big((F \cup \mathrm{C}) \cap \neg \mathrm{B}\big)}{\mathsf{Pr}(\neg \mathrm{B} \cap \mathrm{A})} = \frac{\mathsf{Pr}(F \cup \mathrm{C})}{\mathsf{Pr}(\mathrm{A})}\,,$$

which does not describe a conditional probability.

An overview of all possible interpretations can be found in Figure 8.3. To summarize, all four interpretations agree on almost–surely terminating programs. This, however, is hard to ask from a programmer, as the halting problem is already undecidable for deterministic programs. For non–almost–surely terminating programs, the Nori interpretation is not backward–compatible to the standard wp calculus for `observe`–free programs, which is arguably undesirable. The fourth interpretation is not at all meaningful.

The cwp and the cwlp interpretation are suitable for reasoning about total and partial correctness of probabilistic programs with conditioning, regardless of the program's termination behavior. Moreover, the two transformers are backward–compatible with wp and wlp, respectively.

8.4 CONDITIONING AND NONDETERMINISM

Nondeterminism is a powerful means for underspecifying program behavior. By a nondeterministic choice, we can state that a program should behave either in this or that way, but definitely in one of the two specified

ways. However, while underspecified program behavior might be good for modeling purposes, a nondeterministic program *cannot be executed*, i.e. it does not describe a single algorithm but rather a set of algorithms.

Probabilistic programs, as understood in this section, describe algorithmic procedures that construct complex probability distributions. Nondeterminism does not quite fit into this picture. A probabilistic program with nondeterminism would give a set of probability distributions and may thus not be very relevant for modeling probability distributions. In fact, Gordon *et al.* state that „the use of nondeterminism as a modeling tool for representing unknown quantities in probabilistic programs is not common" [Gor+14].

Combining nondeterministic and probabilistic behavior is well–known to be problematic [Pan01; MOW04; Mis06; VW06; AR08; CS09; Bai+14]. For probabilistic programs with observations, we add yet another problem.

There are (at least) two ways to interpret nondeterministic choice: As an *adversary* or as *alternative implementations*. If we take upon the latter point of view, we take the stance that a nondeterministic choice $\{C_1\} \,\square\, \{C_2\}$ can be replaced by either C_1 or C_2, which is a standard assumption in program refinement [BW89]. Under this mild assumption, and even for loop–free programs, we can show that we cannot define a demonic expectation transformer *by induction on the program structure*, i.e. we cannot simply add a line to Table 8.1 in order to have a cwp rule for nondeterministic choice. We will not treat the proof of this result in this thesis as it is somewhat technical, but refer to [Jan+15a] and [Olm+18] for a detailed treatment.

8.5 HEALTHINESS CONDITIONS

Just like the classical expectation transformers we studied in Chapter 4, conditional expectation transformers too enjoy several properties like continuity, monotonicity, etc. (often called healthiness conditions, cf. Section 4.2). In the following, we present some of those.

8.5.1 *Continuity*

Continuity is perhaps one of the most fundamental properties that conditional expectation transformers enjoy because it ensures for instance well–definedness of the conditional expectation transformer semantics of while loops. A conditional expectation transformer $\mathcal{T} : \mathbb{C} \to \mathbb{C}$ is called $\trianglelefteq$–continuous iff for any chain of expectations $S = \{s_0 \trianglelefteq s_1 \trianglelefteq s_2 \trianglelefteq \ldots\} \subseteq \mathbb{C}$ we have

$$\mathcal{T}(\sup_{\trianglelefteq} S) \;=\; \sup_{\trianglelefteq} \mathcal{T}(S)\,;$$

see Definition A.2 for more details. $\blacktriangleleft$–continuity is defined analogously. Both conditional expectation transformers we have presented in this chapter are continuous with respect to their associated partial order.

THEOREM 8.7 (Continuity of cwp and cwlp [Olm+18]):
Let C be a cpGCL *program. Then:*

A. $\mathsf{cwp}\,[\![C]\!]$ *is* $\trianglelefteq$*–continuous.*

B. $\mathsf{cwlp}\,[\![C]\!]$ *is* $\blacktriangleleft$*–continuous.*

Proof. By induction on the structure of C. Q.E.D.

The importance of continuity for well–defined semantics of loops can be sketched as follows: For any loop–free program C, continuity of $\mathsf{cwp}\,[\![C]\!]$ ensures that the characteristic function of the loop $\texttt{while}(\varphi)\{C\}$ (that has C as its loop body) is also continuous. This ensures by the Kleene fixed point theorem (Theorem A.5) that the characteristic function has a least fixed point, which in turn ensures that $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]$ is well–defined. The fact that the transformer $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]$ itself is also continuous ensures well–defined expectation transformer semantics of nested loops.

8.5.2 *Decoupling*

In Section 8.3 have already hinted at the fact that cwp and cwlp can in some way be decoupled into wp and wlp. If we define

$$\begin{aligned}\mathsf{wp}\,[\![\texttt{observe}(\varphi)]\!]\,(f) &= [\varphi]\cdot f\\ \mathsf{wp}\,[\![\texttt{observe}(\varphi)]\!]\,(g) &= [\varphi]\cdot g\,,\end{aligned}$$

then we can make this statement more formal:

THEOREM 8.8 (Decoupling of cwp and cwlp [Jan+15a; Olm+18]):
Let $C \in$ pGCL. Then:

A. $\mathsf{cwp}\,[\![C]\!]\left(\underline{f}\,/\,\overline{g}\right) = \underline{\mathsf{wp}\,[\![C]\!]\,(f)}\,/\,\overline{\mathsf{wlp}\,[\![C]\!]\,(g)}$

B. $\mathsf{cwlp}\,[\![C]\!]\left(\underline{f}\,/\,\overline{g}\right) = \underline{\mathsf{wlp}\,[\![C]\!]\,(f)}\,/\,\overline{\mathsf{wlp}\,[\![C]\!]\,(g)}$

Proof. By induction on the structure of C. Q.E.D.

8.5.3 *Strictness*

The strictness property of weakest preexpectation transformers states that

$$\mathsf{wp}\,[\![C]\!]\,(0) = 0\,;$$

see Theorem 4.14 A. More abstractly, the least element 0 in the complete lattice $(\mathbb{E}, \preceq)$ is mapped to itself. Dually, the costrictness property of weakest liberal preexpectation transformers states that

$$\mathsf{wlp}\,[\![C]\!]\,(1) = 1\,;$$

see Theorem 4.14 B. Again put more abstractly, the greatest element 1 in the complete lattice $(\mathbb{E}_{\leq 1}, \preceq)$ is mapped to itself.

Conditional expectation transformers are strict and costrict only in a more concrete sense. The following theorem holds:

THEOREM 8.9 ((Co)strictness of cwp and cwlp [Jan+15a; Olm+18]): *Let $C \in \mathsf{pGCL}$. Then:*

A. $\mathsf{cwp}\,[\![C]\!]\left(\frac{0}{1}\right) = \frac{0}{g}$, *where* $g = \mathsf{wlp}\,[\![C]\!]\,(1)$. *Thus, the interpretation of* $\mathsf{cwp}\,[\![C]\!]\left(\frac{0}{1}\right)$ *in any state* σ, *i.e. the fraction* $0/g(\sigma)$, *is either* 0 *or undefined.*

B. $\mathsf{cwlp}\,[\![C]\!]\left(\frac{1}{1}\right) = \frac{g}{g}$, *where* $g = \mathsf{wlp}\,[\![C]\!]\,(1)$. *Thus, the interpretation of* $\mathsf{cwlp}\,[\![C]\!]\left(\frac{1}{1}\right)$ *in any state* σ, *i.e. the fraction* $g(\sigma)/g(\sigma)$, *is either* 1 *or undefined.*

Proof. For proving the strictness of cwp in the sense of Theorem 8.9 A., consider the following:

$$\begin{aligned}
&\mathsf{cwp}\,[\![C]\!]\left(\tfrac{0}{1}\right)\\
&= \tfrac{\mathsf{wp}\,[\![C]\!]\,(0)}{\mathsf{wlp}\,[\![C]\!]\,(1)} && \text{(by decoupling, Theorem 8.8 A.)}
\end{aligned}$$

For proving the costrictness of cwlp in the sense of Theorem 8.9 B., consider the following:

$$\begin{aligned}
&\mathsf{cwlp}\,[\![C]\!]\left(\tfrac{1}{1}\right)\\
&= \tfrac{\mathsf{wlp}\,[\![C]\!]\,(1)}{\mathsf{wlp}\,[\![C]\!]\,(1)} && \text{(by decoupling, Theorem 8.8 B.)}
\end{aligned}$$

Q.E.D.

The above version of strictness tells us that the conditional expected value of the constantly 0 random variable after executing a program C, given that no observation is violated is either 0, or the probability to not valuate any observations is 0. Costrictness on the other hand tells us that the conditional probability to either terminate or not, given that no observation is violated is either 1, or the probability to not valuate any observations is 0.

Notice that cwp is not strict in the more abstract sense we have mentioned in the beginning, i.e. the least element in the underlying partial order is mapped to itself: In particular, the transformer $\mathsf{cwp}\,[\![\mathtt{observe}\,(\varphi)]\!]$ is not strict in that sense, since

$$\mathsf{cwp}\,[\![\mathtt{observe}\,(\varphi)]\!]\left(\tfrac{0}{1}\right) \;=\; [\varphi]\cdot\tfrac{0}{1} \;=\; \tfrac{0}{[\varphi]} \;\neq\; \tfrac{0}{1}\,.$$

Dually, the transformer cwlp is not costrict in the more abstract sense, i.e. the greatest element in the underlying partial order is mapped to itself: Again, the transformer $\mathsf{cwlp}\,[\![\mathtt{observe}\,(\varphi)]\!]$ is not costrict in that sense, since

$$\mathsf{cwlp}\,[\![\mathtt{observe}\,(\varphi)]\!]\left(\tfrac{1}{1}\right) \;=\; [\varphi]\cdot\tfrac{1}{1} \;=\; \tfrac{[\varphi]}{[\varphi]} \;\neq\; \tfrac{1}{1}\,.$$

8.5.4 *Conservativity*

A main advantage of the cwp interpretation of conditional weakest preexpectations (see Section 8.3.1) over the Nori interpretation (see Section 8.3.2) is that the cwp interpretation is backward compatible with classical weakest preexpectations (see Section 4.1.1) for programs that do not contain any observations. Dually, the cwlp interpretation (see Section 8.3.3) is backward compatible with classical weakest liberal preexpectations (see Section 4.1.2). Formally, we can state this as follows:

THEOREM 8.10 (Conservativity of cwp and cwlp [Jan+15a; Olm+18]): *Let $C \in \mathsf{cpGCL}$ be a an* `observe`*–free program. Then:*

A. $\mathsf{cwp}\,[\![C]\!]\left(\frac{f}{1}\right) = \frac{f'}{g'}$ implies $\frac{f'}{g'} = \mathsf{wp}\,[\![C]\!]\,(f)$.

B. $\mathsf{cwlp}\,[\![C]\!]\left(\frac{f}{1}\right) = \frac{f'}{g'}$ implies $\frac{f'}{g'} = \mathsf{wlp}\,[\![C]\!]\,(f)$.

Proof. For the proof of Theorem 8.10 A. consider

$$\begin{aligned}
&\mathsf{cwp}\,[\![C]\!]\left(\frac{f}{1}\right)\\
&= \frac{\mathsf{wp}\,[\![C]\!]\,(f)}{\mathsf{wlp}\,[\![C]\!]\,(1)} && \text{(by decoupling, Theorem 8.8 A.)}\\
&= \frac{\mathsf{wp}\,[\![C]\!]\,(f)}{1} && \text{(by costrictness, Theorem 4.14 B.)}
\end{aligned}$$

and

$$\frac{\mathsf{wp}\,[\![C]\!]\,(f)}{1} = \mathsf{wp}\,[\![C]\!]\,(f)\,.$$

For the proof of Theorem 8.10 B. consider

$$\begin{aligned}
&\mathsf{cwlp}\,[\![C]\!]\left(\frac{f}{1}\right)\\
&= \frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{\mathsf{wlp}\,[\![C]\!]\,(1)} && \text{(by decoupling, Theorem 8.8 B.)}\\
&= \frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{1} && \text{(by costrictness, Theorem 4.14 B.)}
\end{aligned}$$

and

$$\frac{\mathsf{wlp}\,[\![C]\!]\,(f)}{1} = \mathsf{wlp}\,[\![C]\!]\,(f)\,.$$ Q.E.D.

Note that conservativity holds for the Nori interpretation only if the program C terminates almost–surely, which can be nontrivial to assert (see Chapter 6).

8.5.5 *Feasibility*

The property that McIver & Morgan call *feasibility* states that preexpectations cannot become „too large“ [MM05]. The notion of feasibility makes sense for

bounded expectations $f \in \mathbb{E}_{\leq \exists b}$ only (cf. Section 8.5.5). A similar property holds for conditional expectations, too.

A first property we observe for that is that — informally speaking — whenever a conditional postexpectation is not of the form $a/0$, for $a > 0$, then the corresponding preexpectation is also not of that form. This means that if a conditional preexpectation results in a division by zero, it is always of the form „zero divided by zero". Formally, we have the following lemma:

Lemma 8.11:
Let $C \in$ pGCL. Then

$$\forall \sigma \in \Sigma\colon g(\sigma) = 0 \implies f(\sigma) = 0 \quad \text{and} \quad \mathsf{cwp}\,[\![C]\!]\left(\underline{f/g}\right) = \underline{f'/g'}$$
$$\text{implies} \qquad \forall \sigma \in \Sigma\colon g'(\sigma) = 0 \implies f'(\sigma) = 0\,.$$

Proof. By induction on the structure of C. Q.E.D.

The second property is more closely related to the notion of feasibility as studied in Section 4.2.3. Informally speaking, it states that if the interpretation f/g of a conditional postexpectation $\underline{f/g}$ is bounded by a constant b then the interpretation of the corresponding preexpectation is also bounded by b. Formally, we have the following theorem:

Theorem 8.12 (Feasibility of cwp):
Let $C \in$ pGCL. Then

$$\forall \sigma \in \Sigma\colon g(\sigma) > 0 \implies \frac{f(\sigma)}{g(\sigma)} \leq b \quad \text{and} \quad \mathsf{cwp}\,[\![C]\!]\left(\underline{f/g}\right) = \underline{f'/g'}$$
$$\text{implies} \qquad \forall \sigma \in \Sigma\colon g'(\sigma) > 0 \implies \frac{f'(\sigma)}{g'(\sigma)} \leq b\,.$$

Proof. By induction on the structure of C. The only case that is not straightforward is the case for probabilistic choice. Let

$$\mathsf{cwp}\,[\![C_1]\!]\left(\underline{f/g}\right) = \underline{f'/g'} \quad \text{and} \quad \mathsf{cwp}\,[\![C_2]\!]\left(\underline{f/g}\right) = \underline{f''/g''}$$

By the induction hypothesis, we have

$$\forall \sigma \in \Sigma\colon g'(\sigma) > 0 \implies \frac{f'(\sigma)}{g'(\sigma)} \leq b$$
$$\text{and} \quad \forall \sigma \in \Sigma\colon g''(\sigma) > 0 \implies \frac{f''(\sigma)}{g''(\sigma)} \leq b$$

$$\text{implies} \quad \forall \sigma \in \Sigma\colon g'(\sigma) > 0 \implies f'(\sigma) \leq g'(\sigma) \cdot b \qquad (\dagger)$$
$$\text{and} \quad \forall \sigma \in \Sigma\colon g''(\sigma) > 0 \implies f''(\sigma) \leq g''(\sigma) \cdot b\,.$$

For the induction step for probabilistic choice we have

$$\begin{aligned}
&\mathsf{cwp}\,[\![\{\,C_1\,\}\,[p]\,\{\,C_2\,\}]\!]\left(\underline{f/g}\right)\\
&= p\cdot \mathsf{cwp}\,[\![C_1]\!]\left(\underline{f/g}\right) + (1-p)\cdot \mathsf{cwp}\,[\![C_2]\!]\left(\underline{f/g}\right)\\
&= \underline{p\cdot f' + (1-p)\cdot f'' \,/\, p\cdot g' + (1-p)\cdot g''}\,.
\end{aligned}$$

We now fix a state σ in which we interpret the above conditional expectation. If $g'(\sigma) = 0$, then by Lemma 8.11 we also have $f'(\sigma) = 0$. In this case, we can immediately appeal to the induction hypothesis for C_2 as we then have

$$\begin{aligned}
\frac{p\cdot f'(\sigma) + (1-p)\cdot f''(\sigma)}{p\cdot g'(\sigma) + (1-p)\cdot g''(\sigma)} &= \frac{p\cdot 0 + (1-p)\cdot f''(\sigma)}{p\cdot 0 + (1-p)\cdot g''(\sigma)}\\
&= \frac{(1-p)\cdot f''(\sigma)}{(1-p)\cdot g''(\sigma)}\\
&= \frac{f''(\sigma)}{g''(\sigma)}\\
&\leq b\,. \qquad \text{(by I.H. on } C_2\text{)}
\end{aligned}$$

The reasoning is analogous if either $g''(\sigma) = 0$, $p = 0$, or $p = 1$.

In the remaining cases we reason as follows:

$$\begin{aligned}
&\frac{p\cdot f'(\sigma) + (1-p)\cdot f''(\sigma)}{p\cdot g'(\sigma) + (1-p)\cdot g''(\sigma)} \overset{!}{\leq} b\\
\text{iff}\quad & p\cdot f'(\sigma) + (1-p)\cdot f''(\sigma) \leq p\cdot g'(\sigma)\cdot b + (1-p)\cdot g''(\sigma)\cdot b\\
\text{implied by}\quad & p\cdot f'(\sigma) + (1-p)\cdot f''(\sigma) \leq p\cdot f'(\sigma) + (1-p)\cdot f''(\sigma) \quad (\text{by } \dagger)\\
\text{iff}\quad & 0 \leq 0 \qquad \text{Q.E.D.}
\end{aligned}$$

8.5.6 *Monotonicity*

Monotonicity is another fundamental property of conditional expectation transformers. A conditional expectation transformer $\mathcal{T}$ is called $\trianglelefteq$–monotonic iff for any two conditional expectations $\underline{f/g}, \underline{f'/g'} \in \mathbb{C}$, we have that

$$\underline{f/g} \trianglelefteq \underline{f'/g'} \quad \text{implies} \quad \mathcal{T}\left(\underline{f/g}\right) \trianglelefteq \mathcal{T}\left(\underline{f'/g'}\right);$$

see Definition A.3 for more details. $\trianglerighteq$–monotonicity is defined analogously. All conditional expectation transformers we have presented are monotonic with respect to their associated partial order:

THEOREM 8.13 (Monotonicity of cwp and cwlp [Jan+15a; Olm+18]): *Let $C \in$ pGCL. Then:*

1. cwp *is $\trianglelefteq$–monotonic.*

2. cwlp *is* $\blacktriangleleft$*–monotonic.*

Proof. Every continuous function is monotonic, see Theorem A.4. Q.E.D.

Monotonicity is not just a healthiness condition but plays an important role in reasoning about programs, namely for compositional reasoning: Imagine two programs C_1 and C_2 and a postexpectation f such that

$$\mathsf{cwp}\,[\![C_1]\!]\left(\underline{f}/\overline{1}\right) \;\trianglelefteq\; \mathsf{cwp}\,[\![C_2]\!]\left(\underline{f}/\overline{1}\right).$$

Then monotonicity ensures that if we put the components C_1 and C_2 into some context $C \fatsemi \ldots\ldots$, then we can be certain that

$$\mathsf{cwp}\,[\![C \fatsemi C_1]\!]\left(\underline{f}/\overline{1}\right) \;\trianglelefteq\; \mathsf{cwp}\,[\![C \fatsemi C_2]\!]\left(\underline{f}/\overline{1}\right),$$

since $\mathsf{cwp}\,[\![C \fatsemi C_i]\!]\left(\underline{f}/\overline{g}\right) = \mathsf{cwp}\,[\![C]\!]\left(\mathsf{cwp}\,[\![C_i]\!]\left(\underline{f}/\overline{1}\right)\right)$, for $i \in \{1, 2\}$. Note that by construction of $\trianglelefteq$ this implies that the conditional expected value of f, given that no observations fail, is higher after executing $C \fatsemi C_2$ than it is after executing $C \fatsemi C_1$. $\blacktriangleleft$–monotonicity of cwlp — unfortunately — does not come with such a meaningful interpretation, see Section 8.2.2.

8.5.7 *Linearity*

Linearity of expectation transformers plays a prominent role in McIver & Morgan's studies on the classical expectation transformer wp. Our cwp transformer is *not* linear in the sense that

$$\mathsf{cwp}\,[\![C]\!]\left(a \odot \underline{f}/\overline{g} \oplus \underline{f'}/\overline{g'}\right) \;\neq\; a \odot \mathsf{cwp}\,[\![C]\!]\left(\underline{f}/\overline{g}\right) \oplus \mathsf{cwp}\,[\![C]\!]\left(\underline{f'}/\overline{g'}\right). \quad \lightning$$

However, by decoupling of cwp and linearity of wp, we can see that cwp is linear in the *interpretation* of its result, i.e. we have [Jan+15a; Olm+18]:

$$\begin{aligned}
&\mathsf{cwp}\,[\![C]\!]\left(\underline{a \cdot f + g}/\overline{g'}\right) \\
&= \underline{\mathsf{wp}\,[\![C]\!]\,(a \cdot f + g)}/\overline{\mathsf{wlp}\,[\![C]\!]\,(g')} && \text{(by decoupling, Theorem 8.8 A.)} \\
&= \underline{a \cdot \mathsf{wp}\,[\![C]\!]\,(f) + \mathsf{wp}\,[\![C]\!]\,(g)}/\overline{\mathsf{wlp}\,[\![C]\!]\,(g')}\,. \\
& && \text{(by linearity of wp, Theorem 4.21 C.)}
\end{aligned}$$

8.6 PROOF RULES FOR LOOPS

As for weakest preexpectations and expected runtimes, reasoning about loops is a difficult tasks in probabilistic program verification. We have seen in Chapter 5, how invariants can help with this sort of reasoning. In particular, we saw that invariants precisely capture the principles of induction and coinduction.

In this section, we will show how we can reason about conditional preexpectations of loops by means of *conditional invariants*, which basically capture the same notion of invariance as for weakest preexpectations. We will present inductive methods for proving upper bounds on conditional weakest preexpectations of loops, ω–rules for proving lower bounds, and a method for refining obtained bounds.

8.6.1 *Invariants*

The concept of invariants that we employ for the proof rules we present in this section is the same as for weakest preexpectation reasoning, see Section 5.1. The notion of conditional invariants is defined as follows:

Definition 8.14 (Conditional Invariants [Olm+18]):
Let Φ be the cwp*–characteristic function of* `while`$(\varphi)\{C\}$ *with respect to postexpectation $\frac{f}{g} \in \mathbb{C}$. Then $\frac{I}{H} \in \mathbb{C}$ is called a conditional invariant of* `while`$(\varphi)\{C\}$ *with respect to conditional postexpectation $\frac{f}{g}$, iff*

$$\Phi\left(\frac{I}{H}\right) \trianglelefteq \frac{I}{H} .$$

8.6.2 *Induction for Conditional Weakest Preexpectations*

Since conditional weakest preexpectations are defined as least fixed points of continuous functions on complete partial orders, we can make use of the induction principle that we discussed in Section 5.2.1 in order to reason about upper bounds on conditional expected values. Formally, the induction principle states that if $(D, \sqsubseteq)$ is a complete partial order and $\Phi\colon D \to D$ is a continuous self–map on D. Then

$$\forall\, d \in D\colon \quad \Phi(d) \sqsubseteq d \quad \text{implies} \quad \mathsf{lfp}\,\Phi \sqsubseteq d .$$

Applied to the cwp calculus, the induction principle immediately gives us the following proof rule:

Theorem 8.15 (Induction for Upper Bounds on cwp [Olm+18]):
Let $\frac{I}{H} \in \mathbb{C}$ be a conditional superinvariant of `while`$(\varphi)\{C\}$ *with respect to postexpectation $\frac{f}{g}$ (see Definition 8.14). Then*

$$\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\frac{f}{g}\right) \trianglelefteq \frac{I}{H} .$$

Proof. This is an instance of Park's Lemma (see Lemma A.6): Simply choose complete partial order $(\mathbb{C}, \trianglelefteq)$ and continuous function ${}^{\mathsf{cwp}}_{\langle\varphi,C\rangle}\Phi_{\frac{f}{g}}$. Q.E.D.

It is worthwhile to repeat here that the conclusion of Theorem 8.15, namely

$$\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\frac{f}{g}\right) \trianglelefteq \frac{I}{H} .$$

is not only meaningful in the sense of the partial order $\trianglelefteq$ on conditional expectations, i.e. $\underline{I \,/\, H}$ approximates $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$ from above. Also the *interpretation* $\underline{I \,/\, H}$ as an actual quotient approximates the interpretation of $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$ as a quotient from above, i.e.

$$\Phi\left(\underline{I \,/\, H}\right) \trianglelefteq \underline{I \,/\, H} \quad \text{and} \quad \underline{f' \,/\, g'} \coloneqq \mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$$
$$\text{implies} \qquad \frac{f'(\sigma)}{g'(\sigma)} \leq \frac{I(\sigma)}{H(\sigma)}\,,$$

for all σ with $g'(\sigma) > 0$. This demonstrates that conditional invariants are a useful notion for overapproximating the actual conditional expected value.

8.6.3 *Coinduction for Conditional Weakest Liberal Preexpectations*

Analogously to the coinduction rule for classical weakest liberal preexpectations (see Section 5.2.2), there is in theory a coinduction rule for obtaining lower bounds on conditional weakest liberal preexpectations since those are defined as greatest fixed points. However, the lower bound obtained from this theoretically existent coinduction rule would be a lower bound with respect to the partial order $\blacktriangleleft$. Unfortunately, this partial order does not give us an lower bound on the actual conditional probability that we seek for (see Section 8.2.2). We will thus not discuss the coinduction rule for cwlp any further as it is of little practical use.

8.6.4 *ω–Rules*

As is the case for weakest preexpectations or expected runtimes, reasoning about lower bounds of conditional weakest preexpectations is difficult since no inductive or coinductive proof principle is available. For weakest preexpectation reasoning and expected runtimes, we thus resorted to so–called ω–rules which employ ω–invariants (see Section 5.2.4 and Section 7.6.5). The same principle is applicable to conditional weakest preexpectations:

Theorem 8.16 (Lower Bounds on cwp from ω–Invariants [Kam+16]): *Let Φ be the* cwp*–characteristic function of* $\texttt{while}(\varphi)\{C\}$ *with respect to postexpectation $\underline{f \,/\, g}$ and let*

$$\underline{I_0 \,/\, H_0} \trianglelefteq \underline{I_1 \,/\, H_1} \trianglelefteq \underline{I_2 \,/\, H_2} \trianglelefteq \ldots\,,$$

with $\underline{I_0 \,/\, H_0} = \underline{0 \,/\, 1}$ be a monotonically increasing sequence of conditional expectations such that for all $n \in \mathbb{N}$

$$\underline{I_{n+1} \,/\, H_{n+1}} \trianglelefteq \Phi\left(\underline{I_n \,/\, H_n}\right).$$

Then

$$\sup_{n\in\mathbb{N}} \underline{I_n \,/\, H_n} \;\trianglelefteq\; \mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right).$$

Proof. Analogous to the proof of Theorem 5.9 A. Q.E.D.

Just like for weakest preexpectations or expected runtimes, it is necessary to find the limit of such an ω–invariant in order to actually gain some insights from applying Theorem 7.21. That basically just shifts to problem of obtaining bounds into the realm of real analysis. For further remarks on the — in my personal opinion — poor usability and usefulness and on the expendability of ω–rules for upper bounds, see the remarks on ω–rules for weakest preexpectation reasoning in Section 5.2.4.

8.6.5 *Bound Refinement*

We saw in Section 5.2.7 that once we have obtained by some means some bound — be it upper or lower — on a preexpectation of a loop, we have a chance of refining and thereby tightening this bound fairly easily. Since this technique is rooted in fixed point theory and conditional weakest preexpectations of loops are defined as least fixed points, the same technique can be applied to cwp:

Theorem 8.17 (Bound Refinement for cwp):
Let Φ be the cwp*–characteristic function of* $\texttt{while}(\varphi)\{C\}$ *with respect to postexpectation* $\underline{f \,/\, g}$*. Moreover, let* $\underline{I \,/\, H}$ *be an upper bound on the preexpectation* $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$*, such that* $\Phi(\underline{I \,/\, H}) \trianglelefteq \underline{I \,/\, H}$*.*

Then $\Phi(\underline{I \,/\, H})$ *is also an upper bound on* $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$*. Moreover, whenever* $\Phi(\underline{I \,/\, H}) \neq \underline{I \,/\, H}$*, then* $\Phi(\underline{I \,/\, H})$ *is an even tighter upper bound on* $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$ *than* $\underline{I \,/\, H}$*.*

Dually, if $\underline{I \,/\, H}$ *is a lower bound, such that* $\underline{I \,/\, H} \trianglelefteq \Phi(\underline{I \,/\, H})$*, then* $\Phi(\underline{I \,/\, H})$ *is also a lower bound; and whenever* $\Phi(\underline{I \,/\, H}) \neq \underline{I \,/\, H}$*, then* $\Phi(\underline{I \,/\, H})$ *is an even tighter lower bound than* $\underline{I \,/\, H}$*.*

Proof. Analogous to the proof of Theorem 5.15. Q.E.D.

The particular bound refinement of Theorem 8.17 can of course be continued ad infinitum: For instance, if $\underline{I \,/\, H}$ is an upper bound on the preexpectation $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$ with $\Phi(\underline{I \,/\, H}) \trianglelefteq \underline{I \,/\, H}$, then so is $\Phi(\underline{I \,/\, H})$ but also $\Phi^2(\underline{I \,/\, H})$, $\Phi^3(\underline{I \,/\, H})$, and so on. In fact, for increasing n, the sequence $\Phi^n(\underline{I \,/\, H})$ is decreasing and *converges* to an upper bound on $\mathsf{cwp}\,[\![\texttt{while}(\varphi)\{C\}]\!]\left(\underline{f \,/\, g}\right)$ that is *below (or equal to)* $\underline{I \,/\, H}$. For more details, see Section 5.2.7.

8.7 FUTURE AND RELATED WORK

Assertions from classical programming languages correspond to the tests in Kozen's probabilistic propositional dynamic logic [Koz85] and are probably the most closely related concept to observations in probabilistic programming. Both `observe` (φ) and `assert` (φ) block all program executions violating φ. However, `observe` (φ) normalizes the unblocked executions with respect to the total probability mass of all non–violating executions. `assert` (φ), on the other hand, does not renormalize, yielding a subdistribution with a total probability mass of possibly less than one.

A different — more quantitative — interpretation of assertions in probabilistic programming is studied by Sampson *et al.* [Sam+14]. There, assertions are accompanied by a confidence value c and a probability value p. The meaning of such a quantitative assertion is that with confidence c, the assertion holds with probability (at least) p. Assertions in probabilistic programming have also been considered by Chakarov & Sankaranarayanan [CS13].

Beyond assertions, Bichsel *et al.* have extended our work by considering *exceptions* [BGV18]. These result in error states that are neither due to observation violation nor nontermination but rather due to other undesired program behavior, which can explicitly be witnessed. Furthermore, Bichsel *et al.* extend pGCL by a `score` statement, which allows to increase or decrease the probability of specific program executions. The `score` statement can be regarded as a generalization of the `observe` statement and in fact renders `observe` statements obsolete. In the following, we list other related work, sorted by topic, and point to directions for future work.

Measure vs. expectation transformers. Giving semantics to probabilistic programs can be done either in terms of forward moving measure transformers or in terms of backward moving expectation transformers. One of the first measure transformer semantics for probabilistic programs with conditioning was given by Borgström *et al.* [Bor+11]. We, on the other hand, have presented in this chapter an expectation transformer semantics. A semantics similar to ours has been provided by Nori *et al.* [Nor+14].

Nontermination. The main difference between our work and the work of Nori *et al.* is that our expectation transformers explicitly account for nonterminating program behavior, whereas Nori *et al.* normalize only to terminating runs. We presented a thorough comparison in Section 8.3.

Several works on probabilistic programs assume almost–sure or even *certain* termination, see e.g. [Bor+11; CMR13; Hur+14; Sam+14]. For some applications, restricting to almost–sure termination is understandable. In general, this restriction should not be made: For instance, Icard argues that cognitive models based on probabilistic programs should not be restricted to almost–surely terminating programs, for *both theoretical and practical rea-*

sons [Ica17]. Therefore, semantics of a general–purpose probabilistic programming language with conditioning should account for nontermination.

Computability. Inference for probabilistic programs is obviously undecidable, once we consider programs with loops (on the degree of undecidability, see Part III). For probabilistic programs with conditioning, the situation is even more subtle: Ackerman *et al.* have shown that one can construct two *computable* random variables X and Y, such that the conditional probability $\Pr(X \mid Y)$ is *not computable* [AFR11]. In other words: The operation of conditioning itself is what already introduces noncomputability.

In light of the above results, probabilistic inference in the presence of conditioning is arguably a difficult task, which renders overapproximations using our invariant–based rules even more useful.

Nondeterminism. In Section 8.4, we have sketched that we cannot come up with a conditional expectation transformer for nondeterministic programs that is constructed by induction on the structure of the program (for more details, see [Jan+15a; Olm+18]). Intuitively, an inductive expectation transformer can only look into the future (from a program execution time line point of view), but not into the past. For conditional expected values, however, looking into the past appears to be necessary, as, amongst many others, Chen & Sanders noticed [CS09]:

> *[S]tudies have revealed an unsuspected subtlety in the interaction between nondeterministic and probabilistic choices that can be summarised: the demon resolving the nondeterministic choice has memory of previous state changes, whilst the probabilistic choice is made spontaneously.*

This insight is also 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 [AR08; Bai+14].

Program transformation and slicing. Most program transformations for probabilistic programs, such as slicing [Hur+14] aim to accelerate the Markov Chain Monte Carlo analysis. In [Jan+15a] and [Olm+18], the two papers this chapter is based upon, two program transformations that eliminate and one that introduces `observe` statements are presented:

The first transformation „hoists“ the `observe` statements upwards through the program while updating the probabilistic choices. This technique is similar in spirit to [Nor+14]. The result of the hoisting process is a semantically equivalent `observe`–free program.

The second transformation basically recreates rejection sampling: It introduces one outer while loop around the original program. This while loop

executes the original program and sets a flag if an observation is violated during execution. If an observation has been violated, the outer loop reruns the original program. This process is repeated until no observation has been violated. The idea to rerun a program until all observations are passed is also used by [Bai+14] to automate the verification of conditioned temporal logic formulas in Markov models.

The second program transformation has been successfully applied in reasoning about expected sampling times of Bayesian networks [Bat+18b]. It was found that for certain large networks, obtaining a *single sample* can take *millions of years* in expectation using rejection sampling. Since rejection sampling is the *de facto* semantics for inference on most practical probabilistic programming languages, this connection shows that our cwp semantics is a real alternative to rejection sampling.

A third transformation is to replace an independent and identically distributed loop by an `observe` statement, i.e. in principle reversing the second transformation. This has strong resemblances with arguments made in textbooks on randomized algorithms, see e.g., [Sho09, Theorem 9.3.(iii)].

Random assignments from continuous distributions. The measure transformer semantics of Borgström *et al.* [Bor+11] includes sampling from continuous distributions like Gaussians, etc., and also the consequential possibility of conditioning on zero–probability events. However, Borgström *et al.* do not consider unbounded loops or unbounded recursion. We believe it would be a promising direction for future work to develop an expectation transformer semantics that can cope with both sampling from continuous distributions and possible nontermination. Such an endeavor will most likely involve in some way the disintegration theorem [Wike]. This theorem is already subject to intensive research on semantics of probabilistic programs [SR17; CJ17; Koz18].

Conditional expected runtimes. A second direction for future work we propose is to marry the ert calculus from Chapter 7 for reasoning about expected runtimes and the cwp calculus from this chapter in order to obtain a calculus for reasoning about *conditional expected runtimes*. This would enable reasoning about expected runtimes of randomized algorithms restricted to certain situations of interest.

Questions like these can lead to extremely counterintuitive situations. As a simple example, consider how often in expectation we need to throw a die in order to get a 6, *given* that all throws yield an even number. Surprisingly, the answer is 1.5, which is twice as fast as throwing a 3–sided die with numbers 2, 4, and 6 [Jin18].