VPHL: A Verified Partial-Correctness Logic for Probabilistic Programs (Expanded Version)

Robert Rand
Computer and Information Sciences
University of Pennsylvania
`rrand@seas.upenn.edu`

Steve Zdancewic
Computer and Information Sciences
University of Pennsylvania
`stevez@cis.upenn.edu`

Abstract

We introduce a Hoare-style logic for probabilistic programs, called *VPHL*, that has been formally verified in the Coq proof assistant. *VPHL* features propositional, rather than additive, assertions and a simple set of rules for reasoning about these assertions using the standard axioms of probability theory. *VPHL*'s assertions are *partial correctness assertions*, meaning that their conclusions are dependent upon (deterministic) program termination. The underlying simple probabilistic imperative language, *PrImp*, includes a probabilistic toss operator, probabilistic guards and potentially-non-terminating while loops.

1 Introduction

Hoare logic has long been used as a means of formally verifying programs and several papers have presented variants to reason about probabilistic programs [1, 2, 3, 4]. There are significant differences between these approaches but all embrace certain design choices: They reason about sub-distributions instead of full distributions and they restrict the possibility of non-termination. The first limits us from introducing assertions like $Pr(2 = 2) = 1$, which frequently precedes the variable assignment x `:=` 2, into our deductions. It also prevents us from taking the complements of our probabilities, which is critical for probabilistic reasoning. Eliminating `while` loops, or restricting us to those guaranteed to terminate, not only limits the kinds of programs we can analyze, it removes a core feature of Hoare logic: partial correctness assertions. We introduce a Verified Probabilistic Hoare Logic (*VPHL*) that reasons exclusively about full distributions and applies to potentially non-terminating programs. Importantly, *VPHL* is itself formally verified in the Coq proof assistant [5]. As probabilistic Hoare logics are increasingly used to verify critical code (for example, in the Easycrypt project [6]), it is important that the logic itself should rest on the firmest foundations.

Classical Hoare logic reasons about *program states*, mappings from identifiers (or variables) to values. Commands are partial functions from one state to another: for example x `:=` 1 takes a state θ to a state that maps x to one and is otherwise identical. The triple $\{x = 1\}\ c\ \{z = 3\}$ asserts that if a state maps x to 1 and then we run the program c from that state, the resulting state will map z to 3, assuming that the program terminates.

In designing a Hoare logic for probabilistic programs, we would like to introduce triples with probabilistic propositions such as $\{Pr(x = 1) > \frac{1}{2}\}\ c\ \{Pr(z = 3) = \frac{2}{3}\}$. Since states are deterministic (a state either maps x to 1 or it doesn't) we will have to reason about *state distributions* the set of states a program may be in at a given point, and their associated weights (or probabilities). For example, if we toss a fair coin T in a deterministic state, we arrive at a state distribution: One state has T mapped to heads, and the other has T mapped to tails; each state has a weight of $\frac{1}{2}$.

We want to formally verify a probabilistic Hoare logic in the Coq proof assistant. Our first task, then, is to formalize the notion of a distribution and the relevant rules of probability theory in this environment (section 2). We then introduce a simple probabilistic imperative language *PrImp* that operates upon distributions (section 3). The Hoare logic rules we introduce will make claims about *PrImp* programs; and all soundness results will be with respect to *PrImps*'s operational semantics.

Sections 4–7 deal with the Verified Probabilistic Hoare Logic (*VPHL* itself. Sections 4 and 5 introduce the semantics of *VPHL* and the basic rules of the logic, including the rule for coin flips, which introduces probability into Hoare assertions.

Conditioning on a probabilistic guard is traditionally the most difficult part of reasoning about probabilistic programs [1]. Consequently, a major part of our Coq development is devoted to proving the soundness of our If rules. We present two distinct verified If rules based on different philosophies for reasoning about fragments of the distribution in sections 6 and 6.1. We also discuss their relative advantages and disadvantages, and how they compare to other If rules in the literature.

We take a novel approach to reasoning about While loops. We pass two distinct proof obligations onto the user: first, to show that the program terminates with either probability one or zero, and then to show that every iteration of the loop preserves some probabilistic invariant. This is intentionally limiting, as are all attempts to reason about loops in the literature [1, 2], and it reflects the limitations of *PrImp* itself. However, in section 8, we show that our Hoare triples can be interpreted in the broader context of programs with probabilistic termination to make precise statements about the possible outcomes of a program.

Finally, in sections 9 and 12, we discuss how *VPHL* provides a foundation for further work in both Hoare logic and program verification. Our main reasons for formally verifying *VPHL* in Coq are to establish our soundness results on the level of the underlying Calculus of Inductive Constructions and to make our definitions and proofs precise and accessible to the research community at large.[1] This formalization further allows us to formalize and compare the expressivity of the existing Hoare logics, and expand our logic (or even language) to deal with an array of domain specific problems. We consider this degree of precision and unity essential for the further development and application of Hoare logic to verifying probabilistic programs.

2 Modeling Distributions

To begin, we need to formalize our notions of distributions and state distributions. A *distribution with finite support* is a multiset of elements $\{x_1, x_2, \ldots, x_n\}$ along with associated weights $\{w_1, w_2, \ldots, w_n\}$ in which every $w_i \in [0,1]$ and $\sum_{i=1}^{n} w_i = 1$. In our development, we will be concerned exclusively with distributions over program states.

We define distributions (ranged over by Θ) inductively by means of a weighted tree structure. The leaves of the distribution contain states (mappings from identifiers to numeric and boolean values), denoted θ, and we use `Unit` θ to lift a state to a one-element distribution. The *combine* operator $\oplus_p$ takes two trees Θ_1 and Θ_2 and combines them to make one bigger tree, associating weight $p \in (0,1)$ to Θ_1 and weight $(1-p)$ to Θ_2.

[1]The full Coq development is available at `https://github.com/rnrand/VPHL`

For example, suppose we want to give θ_1 and θ_2 a weight of $\frac{1}{6}$ each and give the remaining two-thirds of the weight to θ_3. One possible representation of this distribution is the tree

$$\begin{tikzcd}
& & \oplus_{\frac{1}{3}} \arrow[dl] \arrow[dr] & \\
& \oplus_{\frac{1}{2}} \arrow[dl] \arrow[d] & & \texttt{Unit}\ \theta_3 \\
\texttt{Unit}\ \theta_1 & \texttt{Unit}\ \theta_2 & &
\end{tikzcd}$$

which corresponds to $(\texttt{Unit}\ \theta_1 \oplus_{\frac{1}{2}} \texttt{Unit}\ \theta_2) \oplus_{\frac{1}{3}} \texttt{Unit}\ \theta_3$.

Note that this is not the only possible way to represent this distribution. We define a notion of *distribution equivalence* and say that $\Theta_1 \equiv \Theta_2$ if the total weight given to every θ in the distributions' supports is equal.[2]

We make use of the following three lemmas in our development, which hold for all distributions Θ_i and $p, q, r \in [0, 1]$:

Lemma 2.1 (Distribution Commutativity)

$$\Theta_1 \oplus_p \Theta_2 \equiv \Theta_2 \oplus_{1-p} \Theta_1$$

Lemma 2.2 (Distribution Associativity)

$$(\Theta_1 \oplus_q \Theta_2) \oplus_p \Theta_3 \equiv \Theta_1 \oplus_{pq} (\Theta_2 \oplus_{\frac{p(1-q)}{1-pq}} \Theta_3)$$

Lemma 2.3 (Distribution Merge)

$$(\Theta_1 \oplus_q \Theta_2) \oplus_p (\Theta_3 \oplus_r \Theta_4) \equiv (\Theta_1 \oplus_{q'} \Theta_3) \oplus_{p'} (\Theta_2 \oplus_{r'} \Theta_4)$$

where

$$p' = pq + (1-p)r, q' = \frac{pq}{pq + (1-p)r}, r' = \frac{p(1-q)}{r(p-1) - pq + 1}$$

We illustrate the right-associative version of our preceding distribution below, and encourage the readers to satisfy themselves that the math checks out.

$$\begin{tikzcd}
& \oplus_{\frac{1}{6}} \arrow[dl] \arrow[dr] & & \\
\texttt{Unit}\ \theta_1 & & \oplus_{\frac{1}{5}} \arrow[d] \arrow[dr] & \\
& & \texttt{Unit}\ \theta_2 & \texttt{Unit}\ \theta_3
\end{tikzcd}$$

We now define *probability* over distributions. In the general case, for any distribution X over $\mathcal{X}$ and function $f : \mathcal{X} \to \mathit{Bool}$, $Pr_X(f) = \sum\{w_i : f(x_i) = \mathtt{t}\}$.

[2]In our Coq development we do not posit the decidability of state equivalence, hence distribution equivalence is defined in terms of boolean predicates over states, for which we can state equivalent lemmas.

$$\begin{aligned}\mathcal{A} &::= n \mid v \mid \mathcal{A}+\mathcal{A} \mid \mathcal{A}-\mathcal{A} \mid \mathcal{A}*\mathcal{A}\\ \mathcal{B} &::= \mathtt{t} \mid \mathtt{f} \mid \mathcal{A}=\mathcal{A} \mid \mathcal{A}<\mathcal{A} \mid \neg\mathcal{B} \mid \mathcal{B}\wedge\mathcal{B} \mid \mathcal{B}\vee\mathcal{B}\\ \mathcal{P},\mathcal{Q} &::= Pr(\mathcal{B})=p \mid Pr(\mathcal{B})<p \mid Pr(\mathcal{B})>p \mid \mathcal{P}\wedge\mathcal{P} \mid \mathcal{P}\vee\mathcal{P}\end{aligned}$$

Figure 1: *PrImp* Expressions and the Probabilistic Assertions of *VPHL*

We can define this inductively as follows:

$$Pr_{(\mathtt{Unit}\ x)}(f) = \begin{cases}1 & \text{if } f(x)=\mathtt{t}\\ 0 & \text{if } f(x)=\mathtt{f}\end{cases}$$
$$Pr_{(X_1\oplus_p X_2)}(f) = p\cdot Pr_{X_1}(f) + (1-p)\cdot Pr_{X_2}(f).$$

In our logic, the elements of the distributions will always be states. The probability functions f are the lifted boolean expressions $\mathcal{B}$ from *PrImp* (see figure 1), where $\mathcal{B}(\theta)$ is the value of $\mathcal{B}$ in the given state. For example, let Θ be our distribution above and suppose that $\theta_1(x)=2$, $\theta_2(x)=1$ and $\theta_3(x)=2$. Consider the boolean expression $b\equiv(x=2)$. Then $b(\theta_1)=\mathtt{t}$, $b(\theta_2)=\mathtt{f}$ and $b(\theta_3)=\mathtt{t}$. Hence, $Pr_\Theta(b)=\frac{1}{6}+0+\frac{2}{3}=\frac{5}{6}$.

We prove and make use of the following principles of probability theory. Note that Tautology and Complement are not true of sub-distributions, and hence inapplicable to other Hoare logics [1, 2, 3, 4].

Lemma 2.4 (Normality) *$\forall\Theta,b$, $0\le Pr_\Theta(b)\le 1$.*

Lemma 2.5 (Tautology) *$\forall\Theta$, if b is a tautology, $Pr_\Theta(b)=1$.*

Lemma 2.6 (Contradiction) *$\forall\Theta$, if b is a contradiction, $Pr_\Theta(b)=0$.*

Lemma 2.7 (Equivalence) *$\forall\Theta$, if $b_1\leftrightarrow b_2$, $Pr_\Theta(b_1)=Pr_\Theta(b_2)$.*

Lemma 2.8 (Complement) *$\forall\Theta,b$, $Pr_\Theta(b)=1-Pr_\Theta(\neg b)$.*

Lemma 2.9 (Disjunction) *$\forall\Theta,b_1,b_2$, $Pr_\Theta(b_1\vee b_2)=Pr_\Theta(b_1)+Pr_\Theta(b_2)-Pr_\Theta(b_1\wedge b_2)$.*

We also make extensive use of two useful lemmas specific to our representation of distributions:

Lemma 2.10 (Combine) *$\forall b,p,q,\Theta_1,\Theta_2$, if $Pr_{\Theta_1}(b)=p$ and $Pr_{\Theta_2}(b)=p$ then $Pr_{\Theta_1\oplus_q\Theta_2}(b)=p$.*

Lemma 2.11 (Split) *$\forall\Theta_1,\Theta_2,b,q$, if $p\in\{0,1\}$ and $Pr_{\Theta_1\oplus_q\Theta_2}(b)=p$ then $Pr_{\Theta_1}(b)=p$ and $Pr_{\Theta_2}(b)=p$.*

$$\text{Skip}\ \frac{}{\texttt{skip} \ / \ \texttt{Unit}\ \theta \Downarrow \texttt{Unit}\ \theta} \qquad \frac{c_1 \ / \ \texttt{Unit}\ \theta \Downarrow \Theta' \qquad c_2 \ / \ \Theta' \Downarrow \Theta''}{c_1; c_2 \ / \ \texttt{Unit}\ \theta \Downarrow \Theta''}\ \text{Sequence}$$

$$\text{Assignment}\ \frac{\theta(a) = n}{x \ \texttt{:=}\ a \ / \ \texttt{Unit}\ \theta \Downarrow \texttt{Unit}\ \theta[n/x]} \qquad \frac{\theta(b) = b_0}{y \ \texttt{:=}\ b \ / \ \texttt{Unit}\ \theta \Downarrow \texttt{Unit}\ \theta[b_0/x]}\ \text{Boolean Assignment}$$

$$\text{If True}\ \frac{\theta(y) = \texttt{t} \qquad c_1 \ / \ \texttt{Unit}\ \theta \Downarrow \Theta'}{\texttt{if}\ y\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \ / \ \texttt{Unit}\ \theta \Downarrow \Theta'} \qquad \frac{\theta(y) = \texttt{f} \qquad c_2 \ / \ \texttt{Unit}\ \theta \Downarrow \Theta'}{\texttt{if}\ y\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \ / \ \texttt{Unit}\ \theta \Downarrow \Theta'}\ \text{If False}$$

$$\text{While End}\ \frac{\theta(y) = \texttt{f}}{\texttt{while}\ y\ \texttt{do}\ c \ \ / \ \texttt{Unit}\ \theta \Downarrow \texttt{Unit}\ \theta} \qquad \frac{\theta(y) = \texttt{t} \quad c \ / \ \texttt{Unit}\ \theta \Downarrow \Theta' \quad \texttt{while}\ y\ \texttt{do}\ c \ \ / \ \Theta' \Downarrow \Theta''}{\texttt{while}\ y\ \texttt{do}\ c \ \ / \ \texttt{Unit}\ \theta \Downarrow \Theta''}\ \text{Loop}$$

$$\text{Combine}\ \frac{c \ / \ \Theta_1 \Downarrow \Theta_1' \qquad c \ / \ \Theta_2 \Downarrow \Theta_2'}{c \ / \ \Theta_1 \oplus_p \Theta_2 \Downarrow \Theta_1' \oplus_p \Theta_2'} \qquad \frac{p \in (0,1)}{y \ \texttt{:=}\ \texttt{toss}(p) \ / \ \texttt{Unit}\ \theta \Downarrow \theta[\texttt{t}/y] \oplus_p \theta[\texttt{f}/y]}\ \text{Toss}$$

Figure 2: Operational Semantics for *PrImp*

3 A Simple Probabilistic Imperative Language

VPHL will let us prove assertions about *PrImp*, a probabilistic variant of the simple imperative language *Imp* from Software Foundations [7], with the addition of a coin flip operator `toss`. We present the big-step operational semantics of *PrImp* in figure 2 where $c \ / \ \Theta \Downarrow \Theta'$ means that if we evaluate c in the state distribution Θ we arrive at the new distribution Θ'. *PrImp*'s semantics are designed to satisfy the following principles:

1. *PrImp* should contain an embedding of a deterministic programming language.
2. Deterministic commands should preserve probabilities.
3. Any program with a non-terminating branch should not terminate.

We satisfy these principles by "lifting" *Imp* such that every command behaves in its traditional way on `Unit` (or single-state) distributions. The Combine rule recursively applies a given command to all states in the support, terminating if and only if every such state terminates on the command, satisfying principle 3 (we discuss the rationale for this specification in the following section). The new command p `:=` `toss`(y) splits a `Unit` distribution into two `Unit` states, the first with weight p and y set to `t` and the second with weight $(1-p)$ and y set to `f`.

The following two lemmas follow directly from our operational semantics but are worth making explicit:

Lemma 3.1 (Decomposition) *For any* $c, p, \Theta_1, \Theta_2, \Theta_1', \Theta_2'$,

$$c \ / \ \Theta_1 \oplus_p \Theta_2 \Downarrow \Theta_1' \oplus_p \Theta_2' \iff c \ / \ \Theta_1 \Downarrow \Theta_1' \land c \ / \ \Theta_2 \Downarrow \Theta_2'.$$

Lemma 3.2 (Step Determinism) *For any* $c, \Theta, \Theta', \Theta''$,

$$c \ / \ \Theta \Downarrow \Theta' \land c \ / \ \Theta \Downarrow \Theta'' \implies \Theta' = \Theta''.$$

$$\mathcal{ND} ::= Pr(\mathcal{B}) = p \mid Pr(\mathcal{B}) < p \mid Pr(\mathcal{B}) > p \mid \mathcal{ND} \wedge \mathcal{ND}$$
$$\mathcal{NP} ::= Pr(\mathcal{B}) = 0 \mid Pr(\mathcal{B}) = 1 \mid \mathcal{NP} \wedge \mathcal{NP} \mid \mathcal{NP} \vee \mathcal{NP}$$

Figure 3: Non-Disjunctive and Non-Probabilistic Assertions

4 Hoare Logic Semantics

We now have sufficient background to formally define our Hoare triples. Let us begin with the formal definition of a classical (non-probabilistic) Hoare triple. Here P and Q are *assertions* about states, or more formally, mappings from states to propositions:

Definition 4.1 *We say that a classical Hoare Triple* $\{P\}\ c\ \{Q\}$ *is* valid *if* $\forall \theta, \theta' : P(\theta) \wedge c\ /\ \theta \Downarrow \theta'$ *implies* $Q(\theta')$.

We call P the *precondition* and Q the *postcondition*.

In our Hoare logic, all assertions relate directly to probabilities, using the probabilistic assertions defined in figure 1. Note that the arithmetic and boolean expressions are the same ones used in our definition of probability over state distributions above, and in the commands of *PrImp* itself. In particular $[Pr(\mathcal{B}) = p](\Theta)$ translates into $Pr_\Theta(\mathcal{B}) = p$ and likewise for $<$ and $>$. $\wedge$ and $\vee$ distribute as you would expect, and we use $\neq, \leq, \geq, \neg$ and $\rightarrow$ as abbreviations for the corresponding disjunctions. We express that a given boolean expression b is true throughout a distribution by $Pr(b) = 1$, which we abbreviate $\lceil b \rceil$, and call these assertions *non-probabilistic*.

Definition 4.2 *We say that a Hoare Triple* $\{P\}\ c\ \{Q\}$ *is* valid in PrImp *if* $\forall \Theta, \Theta' : P(\Theta) \wedge c\ /\ \Theta \Downarrow \Theta'$ *implies* $Q(\Theta')$.

As mentioned in the previous section, a *PrImp* `while` loop terminates on a given distribution if and only if it terminates on every state in the distribution's support. Hence, the following two programs do not terminate in our language, even though the second would traditionally terminate with probability 1:

$$y \texttt{ := toss}(\tfrac{2}{3});\ \texttt{if } y \texttt{ then } x := 4 \texttt{ else loop()} \tag{1}$$
$$y \texttt{ := toss}(\tfrac{1}{2});\ \texttt{while } y \texttt{ do } y \texttt{ := toss}(\tfrac{1}{2}) \tag{2}$$

This is motivated by our desire to analyze distributions instead of sub-distributions. If we declared a program to be terminating on $\Theta_1 \oplus_p \Theta_2$ even though Θ_2 does not terminate, we would corrupt our partial correctness assertions: Any time we derived a postcondition for two parts of the state (as in our If rule, figure 5), we would have to account for their possible vacuity when trying to combine them, leading to very weak (or empty) postconditions. However, in section 8, we discuss how our Hoare logic can be combined with termination analysis to reason about probabilistically terminating programs.

Before introducing *VPHL* itself, it's worth calling out two fragments of our assertion logic, the *non-disjunctive* and *non-probabilistic* fragments in figure 3. As implied by their names, the non-disjunctive fragment $\mathcal{ND}$ does not feature disjunctions, and the non-probabilistic fragment $\mathcal{NP}$ only asserts probabilities of 0 or 1. We make two observations about these fragments:

$$\frac{P' \to P \qquad \{P\}\ c\ \{Q\} \qquad Q \to Q'}{\{P'\}\ c\ \{Q'\}}\ \text{Consequence} \qquad \frac{\{P\}\ c_1\ \{Q\} \qquad \{Q\}\ c_2\ \{R\}}{\{P\}\ c_1;\ c_2\ \{R\}}\ \text{Sequence}$$

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}}\ \text{Skip} \qquad \frac{}{\{P[z \mapsto e]\}\ z\ \texttt{:=}\ e\ \{P\}}\ \text{Assignment} \qquad \frac{y \text{ free in } P}{\{P\}\ y\ \texttt{:=}\ \texttt{toss}(p)\ \{P \lhd^y_p\}}\ \text{Toss}$$

Figure 4: The Basic Hoare Logic Rules

Lemma 4.1 *For any* non-disjunctive *assertion P, $P(\Theta_1) \wedge P(\Theta_2)$ implies that $P(\Theta_1 \oplus_p \Theta_2)$ for any $p \in (0,1)$.*

Lemma 4.2 *For any* non-probabilistic *assertion P, $P(\Theta_1 \oplus_p \Theta_2)$ implies $P(\Theta_1)$ and $P(\Theta_2)$ for any $p \in (0,1)$.*

The converse is not true in either case, as the following simple counterexamples will show:

The non-disjunctive probabilistic assertion $Pr(b) = \frac{1}{2}$ is true of $\Theta_1 \oplus_{\frac{1}{2}} \Theta_2$ when $Pr_{\Theta_1}(b) = \frac{1}{2}$ and $Pr_{\Theta_2}(b) = 0$.

The disjunctive non-probabilistic assertion $Pr(b_1) = 1 \vee Pr(b_2) = 1$ is true both of Θ_1 and Θ_2 when b_1 is true throughout Θ_1 and b_2 is true throughout Θ_2. In this case, $Pr_{\Theta_1 \oplus_{\frac{1}{2}} \Theta_2}(b_1) = \frac{1}{2}$ and likewise for b_2.

Lemmas 4.2 and 4.1 lead to the following observation:

Lemma 4.3 *For any* non-disjunctive, non-probabilistic *assertion P and $p \in (0,1)$, $P(\Theta_1 \oplus_p \Theta_2 \leftrightarrow (P(\Theta_1) \wedge P(\Theta_2))$.*

Lemma 4.2 will play a significant role in the development of our While rule in section 7.

5 Basic Hoare Logic Rules

We can now introduce *VPHL* itself. Figure 4 presents the basic rules of *VPHL*. The rules for `if` and `while` commands are presented in figures 5 and 7, with discussion deferred until later in the paper. We will present our soundness result up front:

Theorem 5.1 (Soundness) *All of the* VPHL *rules presented in this paper are sound with respect to the semantics of* PrImp*.*

We prove this theorem in the Coq development, where each rule is individually verified to be sound.

We do not claim completeness (that is, that everything possible to derived can be derived via our Hoare logic) in this paper, though we do demonstrate the usability of *VPHL* via examples (section 10).

The basic rules (with the exception of Toss) are preserved from classical Hoare logic. The `toss` command assigns an identifier y to either `t` or `f`, with probability p and $(1-p)$ respectively. For our Hoare logic, we restrict y from appearing in the precondition P. Since a freshly tossed y is

$$\frac{\{P \wedge \lceil y \rceil\}\ c_1\ \{Q\} \qquad \{P \wedge \lceil \neg y \rceil)\}\ c_2\ \{Q\}}{\{P \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}\ \texttt{if}\ y\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q\}}\ \text{Deterministic If}$$

$$\frac{\begin{array}{ll} 0 < p < 1 & y \text{ unassigned in } c_1, c_2 \\ \{\frac{1}{p} * P_1 \wedge \lceil y \rceil\}\ c_1\ \{\frac{1}{p} * Q_1\} & \{\frac{1}{1-p} * P_2 \wedge \lceil \neg y \rceil\}\ c_2\ \{\frac{1}{1-p} * Q_2\}\end{array}}{\{Pr(y) = p \wedge P_1 \,|\, y \wedge P_2 \,|\, \neg y\}\ \texttt{if}\ y\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q_1 \,|\, y \wedge Q_2 \,|\, \neg y\}}\ \text{Probabilistic If}$$

Figure 5: The Deterministic and Probabilistic If Rules

necessarily independent of all previous probabilities, we can then update all statements of the form $Pr(b) = q$ with $Pr(b \wedge y) = pq$.

We define $P \triangleleft_p^y$, read"P conditioned on y with probability p", inductively as follows:

$$\begin{aligned}
(Pr(b) = q) \triangleleft_p^y &\equiv Pr(b \wedge y) = pq \wedge Pr(b \wedge \neg y) = (1-p)q \\
(Pr(b) < q) \triangleleft_p^y &\equiv Pr(b \wedge y) < pq \wedge Pr(b \wedge \neg y) < (1-p)q \\
(Pr(b) > q) \triangleleft_p^y &\equiv Pr(b \wedge y) > pq \wedge Pr(b \wedge \neg y) > (1-p)q \\
(P_1 \wedge P_2) \triangleleft_p^y &\equiv P_1 \triangleleft_p^y \ \wedge P_2 \triangleleft_p^y \\
(P_1 \vee P_2) \triangleleft_p^y &\equiv P_1 \triangleleft_p^y \ \vee P_2 \triangleleft_p^y
\end{aligned}$$

Conveniently, the resulting rule is lossless. When we apply the Toss rule $\{P\}\ y$ `:=` `toss`(p) $\{P \triangleleft_p^y\}$, any proposition that was true in the precondition will remain true in the postcondition. To formalize this:

Lemma 5.1 *For all P, $P \triangleleft_p^y$ entails P.*

We can show this by simply marginalizing over y in each atomic proposition.

6 Conditioning on Probabilistic Guards

The `if` command is the most difficult to reason about for straightforward reasons. Unlike the previous commands, which behave identically on all of the states in the distribution's support, the `if` command will run one command wherever the guard is true, and another whenever the guard is false.

In the simplest case, the value of the guard will take on the same value throughout the distribution (we say that the guard is *deterministic*), so it's sensible to have a usable If rule, specific to that case. In figure 5 we present such a rule.

The deterministic If rule is based on the standard If rule from Hoare logic. It requires an expression of the form $Pr(y) = 1 \vee Pr(\neg y) = 1$ (written $\lceil y \rceil \vee \lceil \neg y \rceil$) in the precondition, and allows us to reason about the execution of c_1 as if y were true throughout, and to reason about c_2 as if y were false throughout. Provided that these both allow us to conclude Q we can conclude Q about the `if` statement itself. This rule (along with the quasi-deterministic While rule introduced below) proves useful for a variety of probabilistic programs that exclusively use non-probabilistic guards.

This should give us an idea of how to deal with the probabilistic case, where $0 < Pr(y) < 1$. We prove the following distribution equivalence lemma using the commutativity, associativity and merge rules of Section 2:

$$\frac{\begin{array}{ccc} all_0\ P_1 \to all_0\ Q_1 & & all_0\ P_2 \to all_0\ Q_2 \\ \{P_1 \,|\, y\}\ c_1\ \{Q_1 \,|\, y\} & y \text{ absent in } c_1, c_2 & \{P_2 \,|\, \neg y\}\ c_2\ \{Q_2 \,|\, \neg y\} \end{array}}{\{P_1 \,|\, y \wedge P_2 \,|\, \neg y\}\ \texttt{if}\ y\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q_1 \,|\, y \wedge Q_2 \,|\, \neg y\}}\ \text{Unscaled If}$$

Figure 6: A Scaling-Free Variant of the If Rule

Lemma 6.1 *For any state distribution Θ where $Pr(y) = p$ and $p \in (0, 1)$, $\exists \Theta_1, \Theta_2$ such that $Pr_{\Theta_1}(y) = 1$, $Pr_{\Theta_2}(y) = 0$ and $\Theta \equiv \Theta_1 \oplus_p \Theta_2$.*

We can now reason about the sub-distribution in which y is true and the one in which it's false separately, and then recombine their postconditions into a single postcondition.

In order to ensure that we're reasoning about the right part of the distribution, we split the assertion P into two parts, $P_1 \,|\, y$ and $P_1 \,|\, \neg y$ (shown in figure 5) as defined below. Any part of the assertion that does not mention a value for the guard is discarded.

$$\begin{aligned} (Pr(X) = q) \,|\, y &\equiv Pr(X \wedge y) = q \\ (Pr(X) < q) \,|\, y &\equiv Pr(X \wedge y) < q \\ (Pr(X) > q) \,|\, y &\equiv Pr(X \wedge y) > q \\ (P_1 \wedge P_2) \,|\, y &\equiv P_1 \,|\, y \wedge P_2 \,|\, y \\ (P_1 \vee P_2) \,|\, y &\equiv P_1 \,|\, y \vee P_2 \,|\, y \end{aligned}$$

For reasoning about sub-distributions, we take the approach of scaling both sub-distributions up to complete distributions, allowing us to reason normally about both without having to restrict our logic. We achieve this via the scaling operator $*$ that scales up all of probabilities in preconditions and postconditions by $\frac{1}{p}$ or $\frac{1}{1-p}$ where p is the probability of the guard. Since we're looking to reason about sub-distributions (not just sub-assertions), we need the following lemmas:

Lemma 6.2 *For any Θ_1, Θ_2 where $Pr_{\Theta_1}(y) = 1$ and $Pr_{\Theta_2}(y) = 0$, $[P_1 \,|\, y](\Theta_1 \oplus_p \Theta_2)$ if and only if $[\frac{1}{p} * P](\Theta_1)$*

Lemma 6.3 *For any Θ_1, Θ_2 where $Pr_{\Theta_1}(y) = 1$ and $Pr_{\Theta_2}(y) = 0$, $[P_2 \,|\, \neg y](\Theta_1 \oplus_p \Theta_2)$ if and only if $[\frac{1}{(1-p)} * P_2](\Theta_2)$*

We demonstrate that if $P \,|\, y$ holds of the full distribution, then $p{*}P$ holds of the sub-distribution in which y is true. We can then reason about the application of c_1 and c_2 to those distributions separately. In order to combine the two postconditions, we add back the $|\, y$ and $|\, \neg y$s. (That is, we conjoin the values of the guard y to all the probabilistic expression, ensuring that they do not overlap). In order to prevent statements of the form $P(y \wedge \neg y) = p$ from appearing in the combined postconditions, we restrict y from being reassigned in c_1 or c_2.

6.1 Conditionals without Scaling

If we look back at our basic *VPHL* rules in figure 4, we will notice something interesting: None of them depend on the real values on the right hand sides of their atomic assertions. This suggest using an If rule (figure 6) that applies to both probabilistic and deterministic guards and doesn't require scaling distributions at all. Instead, we reason separately about arbitrary distributions in which $P_1 \,|\, y$ and $P_2 \,|\, \neg y$ are true. We have the following lemmas to correspond to the lemmas 6.2 and 6.3 in the previous section:

Lemma 6.4 *For any $\Theta_1, \Theta_2, \Theta_2'$ where $Pr_{\Theta_1}(y) = 1$ and $Pr_{\Theta_2}(y) = Pr_{\Theta_2'}(y) = 0$, $[P_1 \,|\, y](\Theta_1 \oplus_p \Theta_2)$ if and only if $[P_1 \,|\, y](\Theta_1 \oplus_p \Theta_2')$*

Lemma 6.5 *For any $\Theta_1, \Theta_1', \Theta_2'$ where $Pr_{\Theta_1}(y) = Pr_{\Theta_1'}(y) = 1$ and $Pr_{\Theta_2}(y) = 0$, $[P_2 \,|\, \neg y](\Theta_1 \oplus_p \Theta_2)$ if and only if $[P_2 \,|\, \neg y](\Theta_1' \oplus_p \Theta_2)$*

These lemmas say that we can ignore the half of the distribution in which y is false when reasoning about c_1 and ignore the half of the distribution in which y is true when reasoning about c_2, and the conclusions will not interfere with one another. (That is, provided that the truth-value of y is preserved in all the states of the support.) We take this approach in reasoning about $\{P_1 \,|\, y\}\ c_1\ \{Q_1 \,|\, y\}$ and $\{P_2 \,|\, \neg y\}\ c_2\ \{Q_2 \,|\, \neg y\}$ separately.

Let's illustrate this via a simple deduction:

$$\frac{\begin{array}{c}\{Pr(b \wedge y) = \frac{1}{3}\}\ x \texttt{ := } 1\ \{Pr(b \wedge x = 1 \wedge y) = \frac{1}{3}\} \\ \{Pr(b \wedge \neg y) = \frac{1}{4}\}\ x \texttt{ := } 2\ \{Pr(b \wedge x = 2 \wedge \neg y) = \frac{1}{4}\}\end{array}}{\begin{array}{c}\{Pr(b \wedge y) = \frac{1}{3} \wedge Pr(b \wedge \neg y) = \frac{1}{4}\} \\ \texttt{if } y \texttt{ then } x \texttt{ := } 1 \texttt{ else } x \texttt{ := } 2 \\ \{Pr(b \wedge x = 1 \wedge y) = \frac{1}{3} \wedge Pr(b \wedge x = 2 \wedge \neg y) = \frac{1}{4}\}\end{array}}$$

(We implicitly weaken $b \wedge y$ to $b \wedge 1 = 1 \wedge y$ before applying the assignment rule in the first case, and similarly in the second. Note that we don't know the probability of y in this deduction but both branch's conclusions are preserved).

Conveniently, even when focusing on part of a distribution, we never lose our tautology and complement axioms. Consider the following program:

`if` y `then skip else` (y' `:= f;while` y' `do skip`)

We have two competing interests here:

In general, when entering an `else` branch, we only want to to reason about the sub-distribution in which y is false, which we would achieve by only modifying the internals of $P \,|\, \neg y$, as in the previous example. At the same time, to analyze the loop we would very much like to know that $Pr(\neg y') = 1$, which follows from y' `:= f`. The unscaled If rule satisfies both interests by allowing us to use $Pr(\neg y') = 1$ in the derivation of Q_2 while restricting $Pr(\neg y') = 1$ from appearing in Q_2 itself.

Now consider this variation on the above:

`if` y `then skip else` (y' `:=` y`;while` y' `do skip`)

In principle this should be identical to the previous example, except that $0 < Pr(y) < 1$ according to our precondition, rather than 0 as expected from being in the `else` branch. To avoid this problem, we restrict y from appearing in either branch – we can replace all instance of y in the else branch with `f` before doing the program analysis.

Finally, consider the following worrisome Hoare triple:

$$\begin{array}{l}\{P(z = 1 \wedge y) = 1 \wedge P(z = 0 \wedge \neg y) = 0\} \\ \texttt{if } y \texttt{ then skip else } \mathit{loop}() \\ \{P(z = 1 \wedge y) = 1 \wedge P(\texttt{f} \wedge \neg y) > 1\}\end{array}$$

$$\frac{P \to D \qquad \begin{array}{c}\{P \wedge \lceil y \rceil\}\ c\ \{P\} \\ \{D \wedge \lceil y \rceil\}\ c\ \{D \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}\end{array} \qquad D \in \mathcal{NP}}{\{P \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}\ \texttt{while}\ y\ \texttt{do}\ c\ \ \{P \wedge \lceil \neg y \rceil\}}\ \text{While}$$

Figure 7: The While Rule

In principle $\{P\}$ $loop()$ $\{Pr(\mathtt{f} \wedge \neg y) > 1\}$ is a valid triple, for any P. The overall deduction is invalid, though, since the `else` branch is dead code. Note that *VPHL* would not allow us to derive this triple in practice since it has no rule that depends on non-termination, but the validity of the triple is still problematic for our soundness results. In principle, we could avoid this problem by simply restricting the rule to probabilistic cases, as above, which ensures that neither branch is dead code. More simply, though, we can use a simple syntactic restriction on the postcondition.

We define the all_0 predicate on assertion, which denotes that all equalities have zeros on the right hand side:

$$\begin{array}{ll} all_0\ (Pr(b) = 0) & \\ all_0\ (Pr(b) < p) & \text{if } 0 < p \leq 1 \\ all_0\ (P_1 \vee P_2) & \text{if } all_0\ P1 \\ all_0\ (P_1 \vee P_2) & \text{if } all_0\ P2 \\ all_0\ (P_1 \wedge P_2) & \text{if } all_0\ P1 \wedge all_0\ P_2 \end{array}$$

Lemma 6.6 *For any Θ where $Pr_\Theta(y) = 0$, $[P \,|\, y](\Theta) \Leftrightarrow all_0\ P$*

In essence, the restriction that $all_0\ P_1 \to all_0\ Q_1$ says that if $P_1 \,|\, y$ does not imply that $Pr(y) > 0$ then neither can $Q_1 \,|\, y$. Hence, if no part of the support takes the “then” branch of the if statement, $Q_1 \,|\, y$ will be trivially true. The same holds by symmetry for P_2, Q_2 and the “else” branch.

Note that the rules of *VPHL*, with the restricted Unscaled If postcondition, implicitly enforce this restriction, so the all_0 properties do not need to be checked in practice.

One we have the all_0 syntactic restriction, we can relax the limitation on our regular If rule (figure 5) that requires $0 < p < 1$, replacing it with the all_0 restrictions from figure 6. In the case where $p \in \{0, 1\}$ this causes a term with a zero denominator to appear on the right hand side of some sub-derivation, but that can be safely treated as an arbitrary number, and the all_0 condition ensures that no invalid claim (or undefined expression) appears in our main derivation.

7 The While Rule

In order to explain and justify the form of our While rule (see figure 7), let us consider the classical Hoare logic While rule, and why it fails for probabilistic programs:

$$\frac{\{P \wedge y\}\ c\ \{P\}}{\{P\}\ \texttt{while}\ y\ \texttt{do}\ c\ \ \{P \wedge \neg y\}}$$

Consider the following Hoare triple:

```
{$Pr(y') = \frac{1}{2} \wedge \lceil y \to y' \rceil$}
while $y$ do ($y'$ := toss($\frac{1}{2}$); $y$ := ($y' \wedge i < 5$); $i$++)
{$Pr(y') = \frac{1}{2} \wedge \lceil y \to y' \rceil \wedge \lceil \neg y \rceil$}
```

There are two major problems with the derivation of this triple. The first is that a contradiction appears in the precondition: $Pr(y') = \frac{1}{2} \wedge \lceil y \to y' \rceil \wedge \lceil y \rceil$ implies that $Pr(y') = \frac{1}{2}$ and $Pr(y') = 1$. Moreover, the post condition is false – if initially $i = 1$ we run the loop up to 5 times, and the probability of y' coming out as true is $2^{-5} = \frac{1}{32}$.

This illustrates that probabilistic invariants are not guaranteed to hold if different branches of the distribution traverse the loop a different number of times. Hence we introduce the restriction (figure 7) that the value of y must remain non-probabilistic (that is, either $\lceil y \rceil$ or $\lceil \neg y \rceil$) upon the completion of every iteration loop, ensuring that if the loop terminates, all branches terminate concurrently.

Or rather, all branches *should* terminate concurrently but we run into some difficulty: While P may hold for a given distribution $\Theta_1 \oplus_p \Theta_2$ and be sufficient to preserve both P and the determinism of y upon running c, there's no guarantee that either Θ_1 or Θ_2 satisfy P and thereby preserve determinism. However, as noted in section 4 we have a nice subset of non-probabilistic assertions that do satisfy this property and should themselves be sufficient to maintain that the guard's determinism according to the following unproven conjecture:

Conjecture 7.1 *For every command c, if P is an invariant for c that preserves the determinism of y, then P implies some non-probabilistic invariant D that also preserves the determinism of y.*

(Note that this conjecture relates to completeness, not soundness. Also, though it is of interest, it is not assumed anywhere in our Coq development.)

The rationale here is simple: If y is deterministic given P, there must be some weakest sub-assertion of P that prevents y from either being assigned to a probabilistic value, or being reassigned with some probability $p \in (0, 1)$. However, generating this implied invariant isn't always straightforward. Consider the following program:

```
$x$ := $0$;
while $y$ do
  if $z$ then $x$ := $x + m$ else $x$ := $x + n$;
  $y$ := $x < 10$
```

The following complicated precondition ensures that y remains deterministic:

$$\big(Pr(z \wedge m = 2 \wedge n = 1) = \tfrac{1}{3} \wedge Pr(z \wedge m = n = 2) = \tfrac{2}{3}\big) \vee$$
$$\big(Pr(y \wedge m = n = 1) = \tfrac{3}{4} \wedge Pr(\neg y \wedge m = n = 1) = \tfrac{1}{4}\big)$$

which implies by marginalization that $(z \wedge m = 2) \vee (m = n = 1)$.

The first possibility states that we always take the first branch of the If statement and m has a set value. The second states that we take either branch with probability $\frac{1}{2}$ but the branches are identical and deterministically increment x by 1. In either case, x is assigned to a non-probabilistic value which uniquely determines the value of y.

$$\dfrac{\dfrac{1}{p} * P_1 \to D \qquad \begin{array}{c}\{\frac{1}{p} * P_1 \wedge \lceil y \rceil\}\ c\ \{\frac{1}{p} * P_1\} \\ \{D \wedge \lceil y \rceil\}\ c\ \{D \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}\end{array} \qquad \begin{array}{c}\text{fresh } y' \\ D \in \mathcal{NP}\end{array}}{\{Pr(y) = p \wedge P_1 \,|\, y \wedge P_2 \,|\, \neg y\}\ y' \texttt{ := } y;\ \texttt{while } y \texttt{ do } c\ \ \{Q_1 \,|\, y' \wedge P_2 \,|\, \neg y' \wedge \lceil \neg y \rceil\}} \text{ G. While}$$

Figure 8: The Generalized While Rule

Additionally, we can show show that generating the strongest deterministic sub-assertion from a probabilistic one can take exponential time in the worst case.

Consider the following probabilistic assertion in CNF form:

$$\bigwedge_{i=0}^{n} \begin{pmatrix} Pr(x = k_{3n}) & = \frac{1}{n} \vee \\ Pr(x = k_{3n+1}) & = \frac{1}{n} \vee \\ Pr(x = k_{3n+2}) & = \frac{1}{n} \end{pmatrix}$$

for some set of distinct k values.

By marginalization, there are 3^n distinct sets of possible values x can take. Where $n = 3$, $Pr(x \in \{k_0, k_3, k_6\}) = 1 \ \vee Pr(x \in \{k_0, k_3, k_7\}) = 1 \vee \ldots$.

Since we have yet to prove Conjecture 7.1 in the general case, we pass this proof obligation onto the Hoare logic rule, which requires a non-probabilistic invariant to be exhibited along with the probabilistic one. In practice (as opposed to in general) this proves to be fairly straightforward, and often quite convenient. For example, we may only need to show that our counter deterministically takes on a specific value in $\{0, 1, \ldots, n\}$ and that y depends only on i, as in our Random Walk example in section 10.2. We separately reason about the main invariant, which may include probabilistic propositions.

7.1 Generalizing the While Rule

The While rule presented in the previous section is limited in that it requires the guard to be deterministic prior to entering the loop, and upon every iteration of the loop. This is an unnecessarily strong requirement: We can easily reason about a probabilistic guard that is assigned deterministically in the loop (see figure 8). (In fact, we could in principle extend this further, to a guard that is assigned probabilistically for some finite number of iterations n, but not for arbitrary n. Instead, such programs would need to be unrolled into a form we can analyze.) This is shown via a simple loop unrolling of the following form, with a fresh variable y' used as the guard:

$$y' \texttt{ := } y;\ \texttt{while } y \texttt{ do } c \ \equiv$$
$$y' \texttt{ := } y;\ \texttt{if } y' \texttt{ then } (\texttt{while } y \texttt{ do } c\ )\ \texttt{else skip}$$

The general form of the While rule is then immediate from the application of the Assignment, If and Deterministic While rules.

8 Reasoning About Probabilistically Terminating Programs

While our Hoare triples are technically vacuously true for any non-terminating program, we can combine analysis of termination behavior with our Hoare logic to achieve surprisingly precise characterizations of probabilistically terminating programs.

Consider the following simple program.

$$y \texttt{ := toss}(\tfrac{2}{3});\texttt{if } y \texttt{ then } x \texttt{ := } 4 \texttt{ else loop()}$$

Note that this program is non-terminating in our language. Hence, any Hoare triple containing this command is valid in our logic. However, this isn't the same as saying that any triple is *derivable* in our logic, as our logic provides no rules to derive false from non-termination (or to derive falsehood at all).

We can, however, still apply our Hoare logic to this program and conclude (vacuously, one might claim) the following: $Pr(y \wedge x = 4) = \frac{2}{3} \wedge Pr(\neg y) = \frac{1}{3}$

Is this useful? If we were reasoning about sub-distributions in the manner of Ramshaw [1] and Den Hartog [2] we could make the *frequency* claim $Fr(y \wedge x = 4) = \frac{2}{3}$ which posits some sub-distribution of weight $\frac{2}{3}$ in which $y = \mathtt{t}$ and $x = 4$.

We can interpret our Hoare triples similarly. Our logic, as established, gives us precise probabilities for $y \wedge x = 4$ provided that all branches terminate. If the `else` branch doesn't necessarily terminate, the analysis of the `if` branch is still valid and vice-versa. Hence, we can conclude that $Pr(y \wedge x = 4) \geq \frac{2}{3}$.

In fact, we can do better. Consider the following program, where c_1, c_2, c_3 and c_4 are deterministic sub-programs that do not modify y_1, y_2 or y_3 and are not guaranteed to terminate:

```
$y_1$ := toss($p$); $y_2$ := toss($q$); $y_3$ := toss($r$);
if $y_1$ then
  if $y_2$ then $c_1$; $x$ := 1 else $c_2$; $x$ := 2
else
  if $y_3$ then $c_3$; $x$ := 3 else $c_4$; $x$ := 4
```

We can illustrate the program with the following tree, corresponding to a program execution, where the branching represents forking on the associated boolean guard:

$$\begin{tikzcd}
 & & & y_1 \arrow[dll, "p"'] \arrow[drr, "(1-p)"] & & & \\
 & y_2 \arrow[dl, "q"'] \arrow[dr, "(1-q)"] & & & & y_3 \arrow[dl, "r"'] \arrow[dr, "(1-r)"] & \\
x := 1 & & x := 2 & & x := 3 & & x := 4
\end{tikzcd}$$

Note that we've annotated the guards on each `if` statement. This is appropriate, as our logic itself conditions the chosen path on each fork on y or $\neg y$. Further note that given our restrictions on the Hoare While rule, every branch terminates with probability one, or doesn't terminate at all. Using this information, along with knowledge about which branches terminate, we can deduce both the prior probability of achieving any given outcome (including non-termination) and posterior probabilities upon termination for every outcome.

To take one example, assume only the branch that assigns $x := 3$ is non-terminating. Then the prior probabilities of x taking on the value 1, 2, or 4 are pq, $p(1-q)$ and $(1-p)(1-r)$ respectively, along with a $(1-p)r$ chance of non-termination. Upon the program's successful termination, we have $x = 1$ with probability $\frac{pq}{1-(1-p)r}$ and similarly for the other terminating outcomes.

On the other hand, if we don't know whether any of the branches terminate we can't come to any conclusion regarding the probabilities of the outcomes. This isn't surprising, in fact it follows directly from the fundamental results of computability theory. In the general case, the probability of any proposition upon program termination depends directly upon the weight of the branches that do terminate, reducing the problem of assigning probabilities directly to the Halting Problem.

9 Extending VPHL

Before we use *VPHL* to verify a few sample programs, we will introduce some notations that will make our task easier. The first is drawing from a uniform distribution. We define `UNIFORM` as syntactic sugar for a series of tosses:

$$\begin{aligned}
&x \texttt{ := UNIFORM}(1) \equiv x \texttt{ := } 1\\
&x \texttt{ := UNIFORM}(N) \equiv u \texttt{ := toss}(\tfrac{1}{N});\\
&\quad \texttt{if } u \texttt{ then } x \texttt{ := } N \texttt{ else } x \texttt{ := UNIFORM}(N-1)
\end{aligned}$$

where u is a reserved boolean variable.

We can prove the associated Hoare rule[3]

$$\frac{u, x \text{ free in } P}{\{P\}\ x \texttt{ := UNIFORM}(N)\ \{P \lhd^{x}_{N}\}}\ \text{Uniform}$$

where $P\lhd^{x}_{N}$ is the analogue to $P\lhd^{y}_{p}$ with $P(b) = p\lhd^{x}_{N} \equiv P(b \wedge x = 1) = \frac{1}{N} \wedge \cdots \wedge P(b \wedge x = N) = \frac{1}{N}$.

Another useful feature missing from the specification of *VPHL* is the ability to include identifiers on the right side of probabilistic assertions. There is an obvious reason for this: Probabilistic Assertions refer to a distribution, and different states in that distribution may map a given identifier to different values. At the same time, some identifiers will be deterministically set in our program, and we might like to reference those. We can express the desired assertions as follows: $\forall k < N, Pr(i = k) = 1 \rightarrow Pr(b) = f(k)$, where b is any boolean expression, i is the identifier we want to include in the probability, and f is the real-valued function that we want to depend on i. The universal quantifier here represents a series of conjunctions.

Similarly, we would like to be able to say that an identifier i deterministically takes on some value in $\{1, 2, \ldots, n\}$. We write this as $i \in \{1, 2, \ldots n\}$ which is shorthand for $\lceil i = 1 \rceil \vee \lceil i = 2 \rceil \vee \cdots \vee \lceil i = n \rceil$.

Both of the above constructs require us to have an upper bound on the possible values for i, but this is often the case, as in the following examples.

10 Examples

10.1 Uniform Distribution

As a simple demonstration of our logic, let us attempt to prove the Hoare logic rule above. Proving the rule in the general case would require both induction over the precondition and induction over

[3]This is meant to illustrate what we can in principle prove in the logic, though this rule, and the examples given have not been verified in Coq.

n. Instead let's prove the simple case of a uniform distribution for $N = 3$. In order to use our If rules we'll replace u with u_1 and u_2.

Algorithm 1 *UNIFORM*(3)

```
$u_1$ := toss($\frac{1}{3}$);
if $u_1$ then
    $x$ := 3
else
    $u2$ := toss($\frac{1}{2}$)
    if $u_2$ then
        $x$ := 2
    else
        $x$ := 1
    end if
end if
```

We will first try our scaled If rule from figure 5. We will implicitly transform $Pr(b) = p$ to $Pr(b \wedge 2 = 2) = p$ or $Pr(b) = p \wedge Pr(\mathtt{t}) = 1$ as needed throughout the program. We show sub-derivations indented and inline, as is common for Hoare logic analysis:

Algorithm 2 Uniform Derivation – Standard If Rule

```
$\{Pr(b) = p\}$
$u_1$ := toss($\frac{1}{3}$);
$\{Pr(u_1) = \frac{1}{3} \wedge Pr(b \wedge u_1) = \frac{p}{3} \wedge Pr(b \wedge \neg u_1) = \frac{2p}{3}\}$
if $u_1$ then
    $\{Pr(b) = p\}$
    $x$ := 3
    $\{Pr(b \wedge x = 3) = p\}$
else
    $\{Pr(b) = p\}$
    $u2$ := toss($\frac{1}{2}$);
    $\{Pr(u_2) = \frac{1}{2} \wedge Pr(b \wedge u_2) = \frac{p}{2} \wedge Pr(b \wedge \neg u_2) = \frac{p}{2}\}$
    if $u_2$ then
        $\{Pr(b) = p\}$ $x$ := 2 $\{Pr(b \wedge x = 2) = p\}$
    else
        $\{Pr(b) = p\}$ $x$ := 1 $\{Pr(b \wedge x = 1) = p\}$
    end if
    $\{Pr(b \wedge x = 2 \wedge u_2) = \frac{p}{2} \wedge Pr(b \wedge x = 1 \wedge \neg u_2) = \frac{p}{2}\}$
end if
$\{Pr(b \wedge x = 3 \wedge u_1) = \frac{p}{3} \wedge Pr(b \wedge x = 2 \wedge u_2 \wedge \neg u_1) = \frac{p}{3} \wedge Pr(b \wedge x = 1 \wedge \neg u_2 \wedge \neg u_1) = \frac{p}{3}\}$
```

Note that the u_i's are still present in the conclusion. We could in principle conclude only that $Pr(b \wedge x = 1) = \frac{1}{3} \wedge Pr(b \wedge x = 2) = \frac{1}{3} \wedge Pr(b \wedge x = 3) = \frac{1}{3}$ by carrying $P(b) = p$ all the way to the postcondition and then noticing that the three conjuncts add up the probability of b itself, but typically we want to maintain the values of the guards, and we don't wish to complicate things unnecessarily.

Note also how we reason separately about every branch of the program, with the guards removed from assertions and the assertions scaled up to their original values. This makes local reasoning straightforward, but creates something of a disconnect between the reasoning inside the If statement, and that outside.

Algorithm 3 Uniform Derivation – Unscaled If Rule

```
{$Pr(b) = p$}
$u_1$ := toss($\frac{1}{3}$);
{$Pr(b \wedge u_1) = \frac{p}{3} \wedge Pr(b \wedge \neg u_1) = \frac{2p}{3}$}
if $u_1$ then
    {$Pr(b \wedge u_1) = \frac{p}{3}$}
    $x$ := 3
    {$Pr(b \wedge u_1 \wedge x = 3) = \frac{p}{3}$}
else
    {$Pr(b \wedge \neg u_1) = \frac{2p}{3}$}
    $u2$ := toss($\frac{1}{2}$);
    {$Pr(b \wedge \neg u_1 \wedge u_2) = \frac{p}{3} \wedge Pr(b \wedge \neg u_1 \wedge \neg u_2) = \frac{p}{3}$}
    if $u_2$ then
        {$Pr(b \wedge \neg u_1 \wedge u_2) = \frac{p}{3}$}
        $x$ := 2
        {$Pr(b \wedge \neg u_1 \wedge u_2 \wedge x = 2) = \frac{p}{3}$}
    else
        {$Pr(b \wedge \neg u_1 \wedge \neg u_2) = \frac{p}{3}$}
        $x$ := 1
        {$Pr(b \wedge \neg u_1 \wedge \neg u_2 \wedge x = 1) = \frac{p}{3}$}
    end if
    {$Pr(b \wedge \neg u_1 \wedge u_2 \wedge x = 2) = \frac{p}{3} \wedge Pr(b \wedge \neg u_1 \wedge \neg u_2 \wedge x = 1) = \frac{p}{3}$}
end if
{$Pr(b \wedge u_1 \wedge x = 3) = \frac{p}{3} \wedge Pr(b \wedge \neg u_1 \wedge u_2 \wedge x = 2) = \frac{p}{3} \wedge Pr(b \wedge \neg u_1 \wedge \neg u_2 \wedge x = 1) = \frac{p}{3}$}
```

Now consider the same proof using our unscaled if rule from figure 6. As shown in algorithm 3, we always keep track of the probabilities in the main program. That can make the reasoning inside loops somewhat more verbose, and occasionally more complex, but it makes transitioning in and out of loops much simpler. On any branch of the loop we simply take the half of the assertion relevant to that branch, and upon exiting we conjunct the two assertions together. This makes tracking the flow of the program quite a bit easier, and avoids some degree of scaling overhead.

10.2 Random Walks

PrImp is particularly well suited to modeling a random walk on a graph, where the position on the graph can be represented by a single number.

Consider the following algorithm, discussed in Adler et al. [8]: We have a hunter h and a rabbit r in a complete k-vertex graph. The rabbit moves in an unknown pattern, which may take into account its own location, the hunter's and the current step i. It may also be probabilistic. We want to show that if the hunter moves according to a random walk, uniformly choosing a vertex at each step and moving there, he has a $1-(\frac{k-1}{k})^n$ chance of catching the rabbit within n steps, regardless of the rabbit's strategy.

We represent our knowledge of the rabbit's movement by the Hoare triple $\{P\}\ c_R\ \{P \triangleleft_{\mathcal{R}}\}$ where $\triangleleft_{\mathcal{R}}$ is the same as $P(b) = p \triangleleft^x_N$, except that the right hand sides can take on any values r_i such that $\sum_1^k r_i = 1$.

Algorithm 4 Rabbit Hunting

```
$i$ := 0
$y$ := $i \neq n$
$won$ := f
while y do
    $c_R$
    $h$ := UNIFORM($k$)
    $won$ := $won \lor h = r$
    $i$ := $i + 1$
    $y$ := $i \neq n$
end while
```

We first present the deterministic invariant:

Algorithm 5 Deterministic Invariant

```
1: $\{i \in \{0,\dots,n\} \land \lceil y \leftrightarrow (i \neq n)\rceil \land \lceil y \rceil\} \to$
2: $\{i \in \{0,\dots,n\} \land \lceil i \neq n \rceil\} \to$
3: $\{i+1 \in \{0,\dots,n\}\}$
4: $c_R$
5: $\{i+1 \in \{0,\dots,n\}\}$
6: $h$ := UNIFORM($k$)
7: $\{(i+1 \in \{0,\dots,n\})\triangleleft^h_k\} \to$
8: $\{i+1 \in \{0,\dots,n\}\}$
9: $won$ := $won \lor h = r$
10: $\{i+1 \in \{0,\dots,n\}\}$
11: $i$ := $i + 1$
12: $\{i \in \{0,\dots,n\}\} \to$
13: $\{i \in \{0,\dots,n\} \land \lceil (i \neq n) \leftrightarrow (i \neq n)\rceil\}$
14: $y$ := $i \neq n$
15: $\{i \in \{0,\dots,n\} \land \lceil y \leftrightarrow (i \neq n)\rceil\} \to$
16: $\{i \in \{0,\dots,n\} \land \lceil y \leftrightarrow (i \neq n)\rceil\} \land (\lceil y \rceil \lor \lceil \neg y \rceil)$
```

Line 2 follows from line 1 by combining statements with probability one, 3 follows from 2 by eliminating the false case from the disjunction and weakening, and 8 follows from 7 by marginalization. 13 introduces a tautology, and 16 concludes from the determinism of i and $y \leftrightarrow (i \neq n)$ that y is also deterministic.

This is similar to the demonstration of the deterministic guard in the previous example. Note that we *can* marginalize over h in line 6, since $Pr(i = j \land (h = 0 \lor h = 0 \lor \cdots \lor h = (k-1))) = 1$

implies that $Pr(i = j) = 1$ (by weakening the inner formula and the normality lemma).

We then verify the main invariant. Due to space constraints we will use $\mathcal{I}$ to represent $i \in \{0, \dots n\}$ and $\mathcal{I}^-$ to represent $i \in \{0, \dots n-1\}$. Likewise, in steps 7 through 10, we leave out the probabilities of r, h and won, needed for marginalization. We then analyze the full program.

Algorithm 6 Main Invariant

```
1: $\{Pr(\neg won) = \frac{k-1}{k}^i \wedge \mathcal{I} \wedge \lceil y \leftrightarrow (i \neq n) \rceil \wedge \lceil y \rceil\} \rightarrow$
2: $\{\mathcal{I}^- Pr(\neg won) = \frac{k-1}{k}^i\}$
3: $c_R$
4: $\{\mathcal{I}^- \bigwedge_{i=1}^{k} Pr(\neg won \wedge r = i) = \frac{k-1}{k}^i r_i\}$
5: $h$ := UNIFORM($k$)
6: $\{\mathcal{I}^- \wedge \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{k} Pr(\neg won \wedge r = i \wedge h = j) = \frac{k-1}{k}^i r_i \frac{1}{k}\} \rightarrow$
7: $\{\mathcal{I}^- \bigwedge_{i=1}^{k} Pr(\neg won \wedge r = h = i) = \frac{k-1}{k}^i \frac{r_i}{k}\} \rightarrow$
8: $\{\mathcal{I}^- \wedge Pr(\neg won \wedge r = h) = \sum_{i=1}^{k} \frac{k-1}{k}^i \frac{r_i}{k}\} \rightarrow$
9: $\{\mathcal{I}^- \wedge Pr(\neg won \wedge r \neq h) = \frac{k-1}{k}^{i+1}\} \rightarrow$
10: $\{\mathcal{I}^- \wedge Pr(\neg(won \vee h = r)) = \frac{k-1}{k}^{i+1}\}$
11: $won$ := $won \vee h = r$
12: $\{\mathcal{I}^- \wedge Pr(\neg won) = \frac{k-1}{k}^{i+1}\}$
13: $i$ := $i + 1$
14: $\{\mathcal{I} \wedge Pr(\neg won) = \frac{k-1}{k}^i\}$
15: $y$ := $i \neq n$
16: $\{Pr(\neg won) = \frac{k-1}{k}^i \wedge \mathcal{I} \wedge \lceil y \leftrightarrow (i \neq n) \rceil\}$
```

Algorithm 7 Rabbit Hunting: Analysis

```
$\{\lceil$t$\rceil\}$
$i$ := 0
$\{\lceil i = 0 \rceil\}$
$y$ := $i \neq n$
$\{\lceil i = 0 \rceil \wedge \lceil y \leftrightarrow (i \neq n) \rceil\}$
$won$ := f
$\{\lceil i = 0 \rceil \wedge \lceil y \leftrightarrow (i \neq n) \rceil \wedge \lceil \neg won \rceil\} \rightarrow$
$\{Pr(\neg won) = \frac{k-1}{k}^i \wedge \mathcal{I} \wedge \lceil y \leftrightarrow (i \neq n) \rceil \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}$
while y do
    $c_R$
    $h$ := UNIFORM($k$)
    $won$ := $won \vee h = r$
    $i$ := $i + 1$
    $y$ := $i \neq n$
end while
$\{Pr(\neg won) = \frac{k-1}{k}^i \wedge \mathcal{I} \wedge \lceil y \leftrightarrow (i \neq n) \rceil \wedge \lceil \neg y \rceil\} \rightarrow$
$\{Pr(won) = 1 - \frac{k-1}{k}^n\}$
```

11 Related Work

The most significant work in representing distributions in Coq was made by the ALEA project [9] based on the work of [10]. ALEA introduces its own axiomatic library for the unit interval and multiple notions of distributions. ALEA is designed to reason directly about probabilistic programs, and forms a foundation of the Certicrypt cryptographic tool [11]. Our goals in this paper were far more limited than ALEA's in terms of what we aimed to represent, namely discrete distributions with finite support. For these, a simple tree based structure of objects proved sufficient and (significantly) easy to reason about. Our unit intervals are based on the Coq real number library, restricted to $[0,1]$.

Hoare Logic for deterministic programs was introduced in [12]. The first attempt to extend this style of reasoning to probabilistic programs appeared in Ramshaw's thesis [1], which was based upon Kozen Semantics [13] and featured the following rule for conditionals:

$$\frac{\{P \mid b\}\ c_1\ \{Q_1\} \qquad \{P \mid \neg b\}\ c_2\ \{Q_2\}}{\{P\}\ \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q_1 + Q_2\}}$$

where $A \mid b$ (pronounced "A restricted to b") breaks up the predicate into "frequencies" (or sub-distributions) in which b is true and in which b is false. The plus operator in the conclusion means that some part of the distribution satisfies Q_1 and another satisfies Q_2. Reasoning about sub-distributions brings with it a number of difficulties that we set out to avoid, including the restriction that $Pr(\texttt{t}) = 1$ is not true in any strict sub-distribution.

Den Hartog and de Vink's logic *pH* [2] has a similar construction for the If rule but with the following ? operator:

$$\frac{\{b\ ?\ P\}\ c_1\ \{Q_1\} \qquad \{\neg b\ ?\ P\}\ c_2\ \{Q_2\}}{\{P\}\ \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q_1 + Q_2\}}$$

Interestingly, ? is primarily defined on state distributions, not assertions. Modifying their example (using multiset notation), if $\Theta = \{(\theta_1, \frac{1}{2}), (\theta_2, \frac{1}{4}), (\theta_3, \frac{1}{4})\}$ where θ_1 and θ_3 satisfy b, then $b\ ?\ \Theta = \{(\theta_1, \frac{1}{2}), (\theta_3, \frac{1}{4})\}$ and $\neg b\ ?\ \Theta = \{(\theta_2, \frac{1}{4})\}$. Then $b\ ?\ P(\Theta)$ asserts that there is a state Θ' such that $b\ ?\ \Theta' = \Theta$ and $P(\Theta)$. How to integrate this with the rest of the logic isn't made clear, but it would need to involve weakening. Again, the postcondition refers to two sub-distributions.

There are two sufficient conditions for application of the *pH* While rule: Either the loop is "terminating" (guaranteed to terminate on all states) or it is "$\langle c, s\rangle$-closed", meaning there is a lower bound on the probability of termination on each iteration. While the first condition seems difficult to guarantee, the latter allows us to reason about a range of programs that lie outside the scope of *PrImp*. On the other hand, it has the significant limitation that it cannot reason about potentially non-terminating programs.

Chadha et al. [4] take an approach that is similar to ours for a language without While loops, and demonstrate the completeness of their logic. They take an interesting approach to the Toss rule, which, like the assignment rule, requires us to rewrite the precondition in the form of the postcondition. For example, we use the following triple to attain a postcondition of $P(y) = \frac{1}{3}$ upon tossing a coin with bias one-third:

$\{\frac{1}{3}P(\texttt{t}) + \frac{2}{3}P(\texttt{f}) = \frac{1}{3}\}\ y\ \texttt{:=}\ \texttt{toss}(\frac{1}{3})\ \{Pr(y) = \frac{1}{3}\}$.

Given that the identifier cannot appear in the precondition here either (unlike assignment, where we might assign $x\ \texttt{:=}\ x + 1$), it's not clear that this adds expressivity, but reasoning backwards in this fashion may help with weakest precondition proofs.

Their If rule takes the following form:

$$\frac{\{P_1\}\ c_1\ \{Pr(X) = p_1\} \qquad \{P_2\}\ c_2\ \{Pr(X) = p_2\}}{\{P_1/b \wedge P_2/\neg b\}\ \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Pr(X) = p_1 + p_2\}}$$

P/b is similar to our $P\,|\,y$, inserting $\wedge\, b$ into all the probabilistic terms in the assertion. However, the sub-assertions are not scaled, instead we again reason about them in a sub-probability space. In order to allow us to combine postconditions, it includes the significant restriction that both branches' postconditions must refer to the same probability – often this will require significant weakening. In order to derive a complex postcondition, we need to apply the If rule repeatedly, and join the results via conjunction and disjunction rules.

Interestingly, this restriction wasn't present in an earlier version of the logic [3]. That version used an alternative If command that took in a free boolean identifier and set it to `t` or `f` depending on which branch was taken. This allowed a postcondition of the form $P_1/y \wedge P_2/\neg y$ (where y was the new identifier) at the cost of a non-standard If rule and the proliferation of fresh identifiers.

There has also been considerable work done in related formal systems, including Probabilistic Guarded Command Language [14], Dynamic Logic [15, 16] and Kleene Algebra with Tests [17]. We refer the reader to Vasquez et al. [18] for a comparison of Probabilistic Hoare Logic and pGCL and to the related work section of Chadha et al. [4] for a broader discussion of the approaches to probabilistic verification.

Finally, related to the Certicrypt project mentioned above [11], the EasyCrypt cryptographic tool [6] is based upon two logical systems: pRHL, a Probabilistic Relational Hoare Logic for reasoning about two programs simultaneously and pHL, a Probabilistic Hoare Logic for reasoning about a single program. Introduced in Barthe et al. [19], pRHL is based upon Benton's Relational Hoare Logic [20] and uses a technique called "lifting" to avoid talking directly about probabilities even as its conclusions relate to them. pHL (discussed briefly in the Easycrypt tutorial [6] and elsewhere) reasons about transitions from one state to another in probabilistic terms, but a full account of its semantics and underlying logic has not yet been published. Easycrypt demonstrates the utility of a probabilistic Hoare logic in a cryptographic setting.

12 Future Work

Both *PrImp* and *VPHL* are limited by design. *PrImp* expressions are limited to boolean and natural numbers; for ease of analysis we haven't included data structures, function calls or recursion, among other language features. *VPHL* is similarly limited, primarily by its lack of quantification. Existential quantifiers would allow us to express crucial ideas like independence: A and B are independent in Θ iff $\exists p, q$ s.t. $Pr(A) = p \wedge Pr(B) = q \wedge Pr(A \wedge B) = pq$.

VPHL is meant to provide the groundwork for the further study of probabilistic Hoare logics and for their application. It is extensible, meaning that we can add new rules without impacting the language. The new Hoare rules would fall into one of two categories: core rules and derived rules.

An core rule is one that is sound within the Hoare logic, but not redundant in the context of the existing rule. Any new core rules would probably be similar in form to the rules in [4]. They would emphasize completeness over usability, and create a set of rules that can be used to deduce any valid Hoare triple.

A derived rule is a rule that can be derived using existing rules from the logic. Though in the strictest sense these rules are redundant, they enable us to efficiently and intuitively reason about programs. Both of these features are often overlooked in Hoare logics, which limits their practical use.

In this paper we've walked a line between usability and completeness, and there is still a long way to go on the usability front. We envision a bevy of rules for a variety of probabilistic constructs, extending what we did with the `UNIFORM` rule above. There are a number of areas, including cryptography, privacy, machine learning and randomized algorithms, which beg for formal analysis methods, and a corresponding universe of domain-specific logics that can be tailored to these problems. *VPHL* should provide a solid foundation for this future work.

References

[1] L. H. Ramshaw, *Formalizing the Analysis of Algorithms.* PhD thesis, Stanford University, 1979.

[2] J. Den Hartog and E. P. de Vink, "Verifying probabilistic programs using a hoare like logic," *International Journal of Foundations of Computer Science*, vol. 13, no. 03, pp. 315–340, 2002.

[3] R. Chadha, P. Mateus, and A. Sernadas, "Reasoning about states of probabilistic sequential programs," in *Computer Science Logic*, pp. 240–255, Springer, 2006.

[4] R. Chadha, L. Cruz-Filipe, P. Mateus, and A. Sernadas, "Reasoning about probabilistic sequential programs," *Theoretical Computer Science*, vol. 379, no. 1, pp. 142–165, 2007.

[5] Coq Development Team, *The Coq Reference Manual, version 8.4*, 2012. Available electronically at `http://coq.inria.fr/doc`.

[6] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, "Easycrypt: A tutorial," in *Foundations of Security Analysis and Design VII*, pp. 146–166, Springer, 2014.

[7] B. C. Pierce, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, and B. Yorgey, *Software Foundations.* 2014. online at `http://www.cis.upenn.edu/~bcpierce/sf/current`.

[8] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking, "Randomized pursuit-evasion in graphs," *Combinatorics, Probability and Computing*, vol. 12, no. 03, pp. 225–244, 2003.

[9] C. Paulin-Mohring, "Alea: A library for reasoning on randomized algorithms in Coq version 7," *Description of a Coq contribution, Université Paris Sud*, 2012.

[10] P. Audebaud and C. Paulin-Mohring, "Proofs of randomized algorithms in Coq," *Science of Computer Programming*, vol. 74, no. 8, pp. 568–589, 2009.

[11] G. Barthe, B. Grégoire, and S. Zanella Béguelin, "Formal certification of code-based cryptographic proofs," *ACM SIGPLAN Notices*, vol. 44, no. 1, pp. 90–101, 2009.

[12] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[13] D. Kozen, "Semantics of probabilistic programs," *Journal of Computer and System Sciences*, vol. 22, no. 3, pp. 328–350, 1981.

[14] A. McIver and C. Morgan, "Developing and reasoning about probabilistic programs in pgcl," in *Refinement Techniques in Software Engineering*, pp. 123–155, Springer, 2006.

[15] D. Kozen, "A probabilistic pdl," *Journal of Computer and System Sciences*, vol. 30, no. 2, pp. 162–178, 1985.

[16] Y. A. Feldman and D. Harel, "A probabilistic dynamic logic," in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pp. 181–195, ACM, 1982.

[17] A. McIver, E. Cohen, and C. Morgan, "Using probabilistic kleene algebra for protocol verification," in *Relations and Kleene Algebra in Computer Science*, pp. 296–310, Springer, 2006.

[18] P. R. M. Vásquez, N. Wolovick, and P. R. DArgenio, "Probabilistic hoare-like logics in comparison," tech. rep., Tech. rep. Universidad Nacional de Córdoba, 2004.

[19] G. Barthe, B. Grégoire, and S. Zanella Béguelin, "Probabilistic relational hoare logics for computer-aided security proofs," in *Mathematics of Program Construction*, pp. 1–6, Springer, 2012.

[20] N. Benton, "Simple relational correctness proofs for static analyses and program transformations," in *ACM SIGPLAN Notices*, vol. 39, pp. 14–25, ACM, 2004.