VPHL: A Verified Partial-Correctness Logic for Probabilistic Programs [1]

Robert Rand [2] Steve Zdancewic [3]

Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA

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.

Keywords: Hoare Logic, Formal Verification, Coq, Probabilistic Programming, Non-termination

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 [6,7,10,20]. 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 [9].

[1] This material is based in part upon work supported by NSF Grant No. CCF-1421193.

[2] Email:rrand@seas.upenn.edu
[3] Email:stevez@cis.upenn.edu

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

As probabilistic Hoare logics are increasingly used to verify critical code (for example, in the Easycrypt project [2]), 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 1 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 $\frac{1}{2}$.

In section 2, we formalize our notion of a distribution. In the following section we introduce our simple probabilistic imperative language *PrImp*. Sections 4–7 deal with *VPHL* itself, with a particular focus on the If and While rules. Following that (section 8), we discuss how termination analysis can be combined with our partial correctness assertions to yield precise characterizations of generally undecidable problems. We conclude with an example of our logic in practice, discussion of possible extensions to the logic, and a review of the related work in the area.

We will occasionally refer to an expanded version of this paper [21], which includes additional examples and discussion of alternative If and While rules. The underlying Coq development is online at https://github.com/rnrand/VPHL.

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.

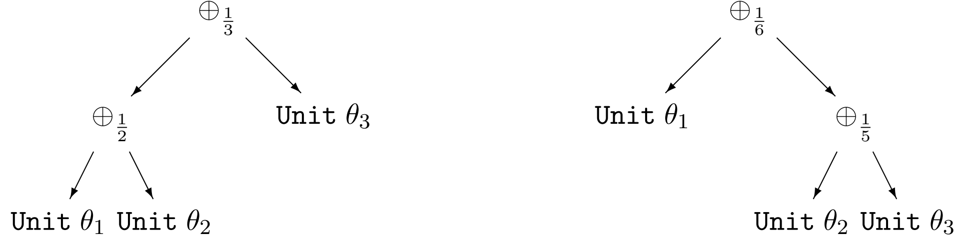
We define distributions (ranged over by Θ) inductively by means of a weighted tree structure. The leaves of the distribution contain states (mappings from identifiers to numeric and boolean values), denoted θ, and we use `Unit` θ to lift a state to a one-element distribution. The *combine* operator $\oplus_p$ 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.

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. We can represent this distribution with

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

Fig. 1. *PrImp* Expressions and the Probabilistic Assertions of *VPHL*

either of the following trees:



which correspond to $(\mathtt{Unit}\ \theta_1 \oplus_{\frac{1}{2}} \mathtt{Unit}\ \theta_2) \oplus_{\frac{1}{3}} \mathtt{Unit}\ \theta_3$ and $\mathtt{Unit}\ \theta_1 \oplus_{\frac{1}{6}} (\mathtt{Unit}\ \theta_2 \oplus_{\frac{1}{5}} \mathtt{Unit}\ \theta_3)$, respectively.

This gives rise to our notion of distribution equivalence. We say that $\Theta_1 \equiv \Theta_2$ if the total weight given to every θ in the distributions' supports is equal.[4] 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 (Commutativity) $\Theta_1 \oplus_p \Theta_2 \equiv \Theta_2 \oplus_{1-p} \Theta_1$

Lemma 2.2 (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 (Merge)

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

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

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

We can define this inductively as follows:

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

We can use this construction to derive all the standard laws of discrete probability theory (see the expanded paper and Coq development for more details).

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),

[4] 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.

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

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

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

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

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

Fig. 2. Operational Semantics for *PrImp*

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 $b(\theta_1) = \texttt{t}$, $b(\theta_2) = \texttt{f}$ and $b(\theta_3) = \texttt{t}$. Hence, $Pr_\Theta(b) = \frac{1}{6} + 0 + \frac{2}{3} = \frac{5}{6}$.

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 [19], 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:

(i) *PrImp* should contain an embedding of a deterministic programming language.

(ii) Deterministic commands should preserve probabilities.

(iii) 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 (iii) (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* $c, p, \Theta_1, \Theta_2, \Theta_1', \Theta_2'$,
$$c \;/\; \Theta_1 \oplus_p \Theta_2 \Downarrow \Theta_1' \oplus_p \Theta_2' \iff c \;/\; \Theta_1 \Downarrow \Theta_1' \wedge c \;/\; \Theta_2 \Downarrow \Theta_2'.$$

Lemma 3.2 (Step Determinism) *For any* $c, \Theta, \Theta', \Theta''$,
$$c \;/\; \Theta \Downarrow \Theta' \wedge c \;/\; \Theta \Downarrow \Theta'' \Longrightarrow \Theta' = \Theta''.$$

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

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

Fig. 3. The Basic Hoare Logic Rules

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:

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

$$y\ \texttt{:=}\ \texttt{toss}(\tfrac{1}{2});\ \texttt{while}\ y\ \texttt{do}\ y\ \texttt{:=}\ \texttt{toss}(\tfrac{1}{2}) \tag{2}$$

This is partly motivated by our approach to the Hoare If rule (see figure 4 later in this paper). We separately reason about each branch of the `if` statement, and then take the conjunction of their conclusions. If either of these conclusions is only vacuously true (since the branch doesn't terminate), and the program is deemed to terminate, our conjunction will be false. Hence, we call such a program non-terminating as well.

5 Basic Hoare Logic Rules

We can now introduce *VPHL* itself. Figure 3 presents the basic rules of *VPHL*. The rules for `if` and `while` commands are presented in figures 4 and 5, 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 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 $P \lhd_p^y$, read"P conditioned on y with probability p", inductively as follows:

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

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

Lemma 5.2 *For all P, $P\lhd_p^y$ entails P.*

We can show this by simply *marginalizing* over y in each atomic proposition. For instance, $P(b \wedge y) = \frac{1}{5}$ and $P(b \wedge \neg y) = \frac{2}{5}$ imply that $P(b) = \frac{1}{5} + \frac{2}{5} = \frac{3}{5}$. We will use this technique, often implicitly, throughout the paper.

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 4 we present such a rule.

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

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

Fig. 4. The Deterministic and Probabilistic If Rules

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:

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, $P_1 \mid y$ and $P_1 \mid \neg y$ (shown in figure 4) 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* $[\frac{1}{(1-p)} * P_2](\Theta_2)$

$$\frac{P \to D \qquad \begin{array}{c}\{P \wedge \lceil y \rceil\}\ c\ \{P\} \\ \{D \wedge \lceil y \rceil\}\ c\ \{D \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}\end{array} \qquad D \text{ is non-probabilistic}}{\{P \wedge (\lceil y \rceil \vee \lceil \neg y \rceil)\}\ \texttt{while}\ y\ \texttt{do}\ c\ \ \{P \wedge \lceil \neg y \rceil\}}\ \text{While}$$

Fig. 5. The While Rule

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.

The reader may observe that the rules of *VPHL* are scale-invariant – that is, they do not depend on the value p on the right hand side of $Pr(X) = p$. This suggest that instead of scaling our P_1 and P_2, we could simply reason about $P_1 \,|\, y$ and $P2 \,|\, \neg y$ separately. Indeed, with minor restrictions such an If rule would be sound, and we discuss this rule in the expanded paper.

7 The While Rule

In order to explain and justify the form of our While rule (see figure 5), 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:

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

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 5) 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, we have a nice subset of non-probabilistic assertions (that is, assertions where every probability is either zero or one) that do satisfy this property.

Lemma 7.1 *For any* non-probabilistic *assertion* P*,* $P(\Theta_1 \oplus_p \Theta_2)$ *implies* $P(\Theta_1)$ *and* $P(\Theta_2)$ *for any* $p \in (0,1)$*.*

Our While rule therefore requires a non-probabilistic invariant that preserves the determinism of the guard 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, for example when analyzing a random walk on an n-vertex graph. We separately reason about the main invariant, which may include probabilistic propositions.

8 Reasoning About Probabilistically Terminating Programs

Consider again the following simple program from section 4.

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

We call this program $c_{partial}$. According to the operational semantics of *PrImp*, this program doesn't terminate, hence $\{P\}\ c_{partial}\ \{Q\}$ is valid (though generally not derivable in *VPHL*) for any assertions P and Q. However, if we consider the truly probabilistic program (one with a seed of random numbers and probabilistic steps) that corresponds to $c_{partial}$, we know that $Pr(x = 4 \wedge y) = 1$ upon successful termination. More generally, for an arbitrary `else` branch which doesn't reassign y we know the following:

$$Pr(x = 4 \wedge y) = \begin{cases} 1 & \text{if the \texttt{else} branch terminates with probability 0} \\ \frac{2}{3} & \text{if the \texttt{else} branch terminates with probability 1} \\ p \in (\frac{2}{3}, 1) & \text{otherwise} \end{cases}$$

This suggest that if we have information about the probabilities of each branch terminating, we can combine this with a straightforward analysis of each branch to derive precise probabilities for the outcomes. This requires that (1) every branch terminates with probability zero or one and (2) that every branch is analyzed independently and has its own post-conditions.

We now note three important features of our logic.

(i) Each branch of an `if` statement has its own, independent derivation

(ii) The post-conditions of `if` statements explicitly mention the value of the guard

(iii) The While rule requires that the loop terminates with probability 0 or 1

Hence, with the simple restrictions that the guards on `if` statements are not reassigned in the program or eliminated via the consequence rule in the Hoare

logic derivation [5], we can combine our logic with termination analysis to precisely characterize probabilistically terminating programs.

Consider the following example, where c_1 through c_4 are programs that do not modify y_1, y_2 or y_3 and are not guaranteed to terminate:

$$
\begin{array}{l}
y_1 \texttt{ := toss}(p);\ y_2 \texttt{ := toss}(q);\ y_3 \texttt{ := toss}(r);\\
\texttt{if } y_1 \texttt{ then}\\
\quad \texttt{if } y_2 \texttt{ then } c_1; x \texttt{ := } 1 \texttt{ else } c_2; x \texttt{ := } 2\\
\texttt{else}\\
\quad \texttt{if } y_3 \texttt{ then } c_3; x \texttt{ := } 3 \texttt{ else } c_4; x \texttt{ := } 4
\end{array}
$$

Assuming that c_1, c_2, c_3 and c_4 are susceptible to analysis by *VPHL* (i.e. their `while` loops terminate or loop deterministically), we should be able to derive the following post-condition:

$$
\begin{array}{lclc}
Pr(x = 1 \wedge y_1 \wedge y_2) = pq & \wedge & Pr(x = 2 \wedge y_1 \wedge \neg y_2) = p(1-q) & \wedge\\
Pr(x = 3 \wedge \neg y_1 \wedge y_3) = (1-p)r & \wedge & Pr(x = 4 \wedge \neg y_1 \neg \wedge y_3) = (1-p)(1-r) &
\end{array}
$$

Now imagine we know that only c_3 never halts. Then the prior probabilities of x taking on the values 1, 2, and 4 are pq, $p(1-q)$ and $(1-p)(1-r)$ respectively, corresponding to the values in our post-condition. The prior probability of the program looping is $(1-p)r$. 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: 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:

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

where u is a reserved boolean variable.

[5] This restriction requires carefully managing the scope of our reasoning to avoid conflicting assertions. For example, if one branch is non-terminating, we could use our While rule to derive false and thereby an incorrect assertion about another branch of the program. In the case where we can move all subsequent commands inside If statements, the form of the If rule ensures this scoping for us. However, in the general case with loops, the scoping is more difficult to enforce and is left to the user of the logic.

We can prove the associated Hoare rule [6]

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

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: $\forall k < N, Pr(i = k) = 1 \to Pr(b) = f(k)$, 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 examples we analyzed.

10 Simulating A 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 N. Instead let's prove the simple case of a uniform distribution for $N = 3$. In order to use our If rule we will replace u with u_1 and u_2.

Algorithm 1 *UNIFORM*(3)

```
u1 := toss(1/3);
if u1 then
    x := 3
else
    u2 := toss(1/2)
    if u2 then
        x := 2
    else
        x := 1
    end if
end if
```

We will now try to prove that $\{P\}\ x \texttt{ := UNIFORM}(N)\ \{P \lhd^x_N\}$ for an atomic proposition P of the form $Pr(b) = p$. We will implicitly use the consequence rule to 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 will occasionally make this explicit by means of an

[6] 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.

arrow symbol ($\rightarrow$). We show sub-derivations indented and inline, as is common for Hoare logic analysis:

Algorithm 2 Uniform Derivation

$\{Pr(b) = p\}$
u_1 := toss($1/3$);
$\{Pr(u_1) = 1/3 \wedge Pr(b \wedge u_1) = p/3 \wedge Pr(b \wedge \neg u_1) = 2p/3\}$
if u_1 **then**
  $\{3/1 * [Pr(b) = p/3] \wedge \lceil u_1 \rceil\} \rightarrow \{Pr(b \wedge 3 = 3) = p\}$
  x := 3
  $\{Pr(b \wedge x = 3) = p\}$
else
  $\{3/2 * [Pr(b) = 2p/3] \wedge \lceil \neg u_1 \rceil\} \rightarrow \{Pr(b) = p\}$
  u_2 := toss($1/2$);
  $\{Pr(u_2) = 1/2 \wedge Pr(b \wedge u_2) = p/2 \wedge Pr(b \wedge \neg u_2) = p/2\}$
  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
  $\{Pr(b \wedge x = 2 \wedge u_2) = p/2 \wedge Pr(b \wedge x = 1 \wedge \neg u_2) = p/2\}$
end if
$\{Pr(b \wedge x = 3 \wedge u_1) = p/3 \wedge Pr(b \wedge x = 2 \wedge u_2 \wedge \neg u_1) = p/3$
$\wedge Pr(b \wedge x = 1 \wedge \neg u_2 \wedge \neg u_1) = p/3\}$

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.

In the expanded version of this paper [21], we show how this proof is simplified by means of an alternative If rule that doesn't require scaling. We further illustrate the use of the While rule by analyzing a random walk.

11 Related Work

The most significant work in representing distributions in Coq was made by the ALEA project [18] based on the work of [1]. 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 [4]. 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 a foundational paper by C. A. R. Hoare [12]. The first attempt to extend this style of reasoning to

probabilistic programs appeared in Ramshaw's thesis [20], which was based upon Kozen Semantics [14] 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 [10] has a similar construction for the If rule but with the following ? operator:

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

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 $b \mathbin{?} \Theta = \{(\theta_1, \frac{1}{2}), (\theta_3, \frac{1}{4})\}$ and $\neg b \mathbin{?} \Theta = \{(\theta_2, \frac{1}{4})\}$. Then $b \mathbin{?} P(\Theta)$ asserts that there is a state Θ' such that $b \mathbin{?} \Theta' = \Theta$ 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. [6] 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 := $x + 1$), 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 $P \mid y$, 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 [7]. 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 [17] (formalized in HOL4 [13] and Isabelle/HOL [8]),, Dynamic Logic [11,15] and Kleene Algebra with Tests [16]. We refer the reader to Vasquez et al. [22] for a comparison of Probabilistic Hoare Logic and pGCL and to the related work section of Chadha et al. [6] for a broader discussion of the approaches to probabilistic verification.

Finally, related to the Certicrypt project mentioned above [4], the EasyCrypt cryptographic tool [2] 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. [3], pRHL is based upon Benton's Relational Hoare Logic [5] and uses a technique called "lifting" to avoid talking directly about probabilities allow us to derive probabilities within sub-distributions. pHL (discussed briefly in the Easycrypt tutorial [2] 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.

A 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 [6]. 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. Adding such rules to our logic could have a substantial impact on its usability.

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] Audebaud, P. and C. Paulin-Mohring, *Proofs of randomized algorithms in Coq*, Science of Computer Programming **74** (2009), pp. 568–589.

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

[3] Barthe, G., B. Grégoire, S. Heraud and S. Z. Béguelin, *Computer-aided security proofs for the working cryptographer*, in: *Advances in Cryptology–CRYPTO 2011*, Springer, 2011 pp. 71–90.

[4] Barthe, G., B. Grégoire and S. Zanella Béguelin, *Formal certification of code-based cryptographic proofs*, in: *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09 (2009), pp. 90–101.

[5] Benton, N., *Simple relational correctness proofs for static analyses and program transformations*, in: *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04 (2004), pp. 14–25.

[6] Chadha, R., L. Cruz-Filipe, P. Mateus and A. Sernadas, *Reasoning about probabilistic sequential programs*, Theoretical Computer Science **379** (2007), pp. 142–165.

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

[8] Cock, D., *Verifying probabilistic correctness in Isabelle with pGCL*, in: *Proceedings of the 7th Systems Software Verification*, Sydney, Australia, 2012, pp. 1–10.

[9] Coq Development Team, "The Coq Reference Manual, version 8.4," (2012), available electronically at http://coq.inria.fr/doc.

[10] Den Hartog, J. and E. P. de Vink, *Verifying probabilistic programs using a Hoare like logic*, International Journal of Foundations of Computer Science **13** (2002), pp. 315–340.

[11] Feldman, Y. A. and D. Harel, *A probabilistic dynamic logic*, in: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82 (1982), pp. 181–195.

[12] Hoare, C. A. R., *An axiomatic basis for computer programming*, Communications of the ACM **12** (1969), pp. 576–580.

[13] Hurd, J., A. McIver and C. Morgan, *Probabilistic guarded commands mechanized in HOL*, Theoretical Computer Science **346** (2005), pp. 96–112.

[14] Kozen, D., *Semantics of probabilistic programs*, Journal of Computer and System Sciences **22** (1981), pp. 328–350.

[15] Kozen, D., *A probabilistic pdl*, Journal of Computer and System Sciences **30** (1985), pp. 162–178.

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

[17] Morgan, C. and A. McIver, *pGCL: Formal reasoning for random algorithms*, South African Computer Journal (1999), pp. 14–27.

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

[19] Pierce, B. C., 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.

[20] Ramshaw, L. H., "Formalizing the Analysis of Algorithms," Ph.D. thesis, Stanford University (1979).

[21] Rand, R. and S. Zdancewic, *VPHL: A verified partial-correctness logic for probabilistic programs (expanded version)*, Technical Report MS-CIS-15-06, University of Pennsylvania (2015).

[22] Vásquez, M. D., N. Wolovick and P. R. D'Argenio, *Probabilistic Hoare-like logics in comparison*, Technical report, Tech. rep. Universidad Nacional de Córdoba (2004).