An Assertion-Based Program Logic for Probabilistic Programs

Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, Pierre-Yves Strub

An Assertion-Based Program Logic for Probabilistic Programs*

Gilles Barthe, Thomas Espitau, Marco Gaboardi
Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub

[1] IMDEA Software Institute, Spain
[2] Université Paris 6, France
[3] University at Buffalo, SUNY, USA
[4] Inria Sophia Antipolis–Méditerranée, France
[5] University College London, UK
[6] École Polytechnique, France

Abstract. Research on deductive verification of probabilistic programs has considered expectation-based logics, where pre- and post-conditions are real-valued functions on states, and assertion-based logics, where pre- and post-conditions are boolean predicates on state distributions. Both approaches have developed over nearly four decades, but they have different standings today. Expectation-based systems have managed to formalize many sophisticated case studies, while assertion-based systems today have more limited expressivity and have targeted simpler examples. We present Ellora, a sound and relatively complete assertion-based program logic, and demonstrate its expressivity by verifying several classical examples of randomized algorithms using an implementation in the EasyCrypt proof assistant. Ellora features new proof rules for loops and adversarial code, and supports richer assertions than existing program logics. We also show that Ellora allows convenient reasoning about complex probabilistic concepts by developing a new program logic for probabilistic independence and distribution law, and then smoothly embedding it into Ellora. Our work demonstrates that the assertion-based approach is not fundamentally limited and suggests that some notions are potentially easier to reason about in assertion-based systems.

1 Introduction

The most mature systems for deductive verification of randomized algorithms are *expectation-based* techniques; seminal examples include PPDL [28] and pGCL [34]. These approaches reason about *expectations*, functions E from states to real numbers,[7] propagating them backwards through a program until they are transformed into a mathematical function of the input. Expectation-based systems are both theoretically elegant [24,16,35,23] and practically useful;

* This is the full version of the paper.

[7] Treating a program as a function from input states s to output distributions $\mu(s)$, the expected value of E on $\mu(s)$ is an expectation.

implementations have verified numerous randomized algorithms [19,21]. However, properties involving multiple probabilities or expected values can be cumbersome to verify—each expectation must be analyzed separately.

An alternative approach envisioned by Ramshaw [37] is to work with predicates over distributions. A direct comparison with expectation-based techniques is difficult, as the approaches are quite different. In broad strokes, assertion-based systems can verify richer properties in one shot and have specifications that are arguably more intuitive, especially for reasoning about loops, while expectation-based approaches can transform expectations mechanically and can reason about non-determinism. However, the comparison is not very meaningful for an even simpler reason: existing assertion-based systems such as [8,18,38] are not as well developed as their expectation-based counterparts.

Restrictive assertions. Existing probabilistic program logics do not support reasoning about expected values, only probabilities. As a result, many properties about average-case behavior are not even expressible.

Inconvenient reasoning for loops. The Hoare logic rule for deterministic loops does not directly generalize to probabilistic programs. Existing assertion-based systems either forbid loops, or impose complex semantic side conditions to control which assertions can be used as loop invariants. Such side conditions are restrictive and difficult to establish.

No support for external or adversarial code. A strength of expectation-based techniques is reasoning about programs combining probabilities and *non-determinism*. In contrast, Morgan and McIver [30] argue that assertion-based techniques cannot support compositional reasoning for such a combination. For many applications, including cryptography, we would still like to reason about a commonly-encountered special case: programs using external or adversarial code. Many security properties in cryptography boil down to analyzing such programs, but existing program logics do not support adversarial code.

Few concrete implementations. There are by now several independent implementations of expectation-based techniques, capable of verifying interesting probabilistic programs. In contrast, there are only scattered implementations of probabilistic program logics.

These limitations raise two points. Compared to expectation-based approaches:

1. Can assertion-based approaches achieve similar expressivity?
2. Are there situations where assertion-based approaches are more suitable?

In this paper, we give positive evidence for both of these points.[8] Towards the first point, we give a new assertion-based logic Ellora for probabilistic programs, overcoming limitations in existing probabilistic program logics. Ellora supports a rich set of assertions that can express concepts like expected values

[8] Note that we do not give mathematically precise formulations of these points; as we are interested in the practical verification of probabilistic programs, a purely theoretical answer would not address our concerns.

and probabilistic independence, and novel proof rules for verifying loops and adversarial code. We prove that Ellora is sound and relatively complete.

Towards the second point, we evaluate Ellora in two ways. First, we define a new logic for proving probabilistic independence and distribution law properties—which are difficult to capture with expectation-based approaches—and then embed it into Ellora. This sub-logic is more narrowly focused than Ellora, but supports more concise reasoning for the target assertions. Our embedding demonstrates that the assertion-based approach can be flexibly integrated with intuitive, special-purpose reasoning principles. To further support this claim, we also provide an embedding of the Union Bound logic, a program logic for reasoning about accuracy bounds [4]. Then, we develop a full-featured implementation of Ellora in the EasyCrypt theorem prover and exercise the logic by mechanically verifying a series of complex randomized algorithms. Our results suggest that the assertion-based approach can indeed be practically viable.

Abstract logic. To ease the presentation, we present Ellora in two stages. First, we consider an abstract version of the logic where assertions are general predicates over distributions, with no compact syntax. Our abstract logic makes two contributions: reasoning for loops, and for adversarial code.

Reasoning about Loops. Proving a property of a probabilistic loop typically requires establishing a loop invariant, but the class of loop invariants that can be soundly used depends on the termination behavior—stronger termination assumptions allows richer loop invariants. We identify three classes of assertions that can be used for reasoning about probabilistic loops, and provide a proof rule for each one:

- arbitrary assertions for *certainly terminating* loops, i.e. loops that terminate in a finite amount of iterations;
- *topologically closed* assertions for *almost surely* terminating loops, i.e. loops terminating with probability 1;
- *downwards closed* assertions for arbitrary loops.

The definition of topologically closed assertion is reminiscent of Ramshaw [37]; the stronger notion of downwards closed assertion appears to be new.

Besides broadening the class of loops that can be analyzed, our rules often enable simpler proofs. For instance, if the loop is certainly terminating, then there is no need to prove semantic side-conditions. Likewise, there is no need to consider the termination behavior of the loop when the invariant is downwards and topologically closed. For example, in many applications in cryptography, the target property is that a "bad" event has low probability: $\Pr[E] \leq k$. In our framework this assertion is downwards and topologically closed, so it can be a loop invariant regardless of the termination behavior.

Reasoning about Adversaries. Existing assertion-based logics cannot reason about probabilistic programs with *adversarial* code. *Adversaries* are special probabilistic procedures consisting of an interface listing the concrete procedures

that an adversary can call (*oracles*), along with restrictions like how many calls an adversary may make. Adversaries are useful in cryptography, where security notions are described using experiments in which adversaries interact with a challenger, and in game theory and mechanism design, where adversaries can represent strategic agents. Adversaries can also model inputs to *online* algorithms.

We provide proof rules for reasoning about adversary calls. Our rules are significantly more general than previously considered rules for reasoning about adversaries. For instance, the rule for adversary used by [4] is restricted to adversaries that cannot make oracle calls.

Metatheory. We show soundness and relative completeness of the core abstract logic, with mechanized proofs in the Coq proof assistant.[9]

Concrete logic. While the abstract logic is conceptually clean, it is inconvenient for practical formal verification—the assertions are too general and the rules involve semantic side-conditions. To address these issues, we flesh out a concrete version of Ellora. Assertions are described by a grammar modeling a two-level assertion language. The first level contains state predicates—deterministic assertions about a single memory—while the second layer contains probabilistic predicates constructed from probabilities and expected values over discrete distributions. While the concrete assertions are theoretically less expressive than their counterparts in the abstract logic, they can already encode common properties and notions from existing proofs, like probabilities, expected values, distribution laws and probabilistic independence. Our assertions can express theorems from probability theory, enabling sophisticated reasoning about probabilistic concepts.

Furthermore, we leverage the concrete syntax to simplify verification.

- We develop an automated procedure for generating pre-conditions of non-looping commands, inspired by expectation-based systems.
- We give syntactic conditions for the closedness and termination properties required for soundness of the loop rules.

Implementation and case studies. We implement Ellora on top of EasyCrypt, a general-purpose proof assistant for reasoning about probabilistic programs, and we mechanically verify a diverse collection of examples including textbook algorithms and a randomized routing procedure. We develop an EasyCrypt formalization of probability theory from the ground up, including tools like concentration bounds (e.g., the Chernoff bound), Markov's inequality, and theorems about probabilistic independence.

Embeddings. We propose a simple program logic for proving *probabilistic independence*. This logic is designed to reason about independence in a lightweight way, as is common in paper proofs. We prove that the logic can be embedded into Ellora, and is therefore sound. Furthermore, we prove an embedding of the Union Bound logic [4].

[9] The formalization is available at `https://github.com/strub/xhl`.

2 Mathematical Preliminaries

As is standard, we will model randomized computations using *sub-distributions*.

Definition 1. *A* sub-distribution *over a set A is defined by a mass function $\mu : A \to [0,1]$ that gives the probability of the unitary events $a \in A$. This mass function must be s.t. $\sum_{a \in A} \mu(a)$ is well-defined and $|\mu| \triangleq \sum_{a \in A} \mu(a) \leq 1$. In particular, the* support $\mathrm{supp}(\mu) \triangleq \{a \in A \mid \mu(a) \neq 0\}$ *is discrete.*[10] *The name "sub-distribution" emphasizes that the total probability may be strictly less than* 1. *When the* weight $|\mu|$ *is equal to* 1, *we call μ a* distribution. *We let* $\mathbf{SDist}(A)$ *denote the set of sub-distributions over A. The probability of an event $E(x)$ w.r.t. a sub-distribution μ, written $\Pr_{x \sim \mu}[E(x)]$, is defined as $\sum_{x \in A | E(x)} \mu(x)$.*

Simple examples of sub-distributions include the *null sub-distribution* $\mathbf{0}$, which maps each element of the underlying space to 0; and the *Dirac distribution centered on x*, written δ_x, which maps x to 1 and all other elements to 0. The following standard construction gives a monadic structure to sub-distributions.

Definition 2. *Let $\mu \in \mathbf{SDist}(A)$ and $f : A \to \mathbf{SDist}(B)$. Then $\mathbb{E}_{a \sim \mu}[f] \in \mathbf{SDist}(B)$ is defined by*

$$\mathbb{E}_{a \sim \mu}[f](b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b).$$

We use notation reminiscent of expected values, as the definition is quite similar.

We will need two constructions to model branching statements.

Definition 3. *Let $\mu_1, \mu_2 \in \mathbf{SDist}(A)$ such that $|\mu_1| + |\mu_2| \leq 1$. Then $\mu_1 + \mu_2$ is the sub-distribution μ such that $\mu(a) = \mu_1(a) + \mu_2(a)$ for every $a \in A$.*

Definition 4. *Let $E \subseteq A$ and $\mu \in \mathbf{SDist}(A)$. Then the restriction $\mu_{|E}$ of μ to E is the sub-distribution such that $\mu_{|E}(a) = \mu(a)$ if $a \in E$ and 0 otherwise.*

Sub-distributions are partially ordered under the pointwise order.

Definition 5. *Let $\mu_1, \mu_2 \in \mathbf{SDist}(A)$. We say $\mu_1 \leq \mu_2$ if $\mu_1(a) \leq \mu_2(a)$ for every $a \in A$, and we say $\mu_1 = \mu_2$ if $\mu_1(a) = \mu_2(a)$ for every $a \in A$.*

We use the following lemma when reasoning about the semantics of loops.

Lemma 1. *If $\mu_1 \leq \mu_2$ and $|\mu_1| = 1$, then $\mu_1 = \mu_2$ and $|\mu_2| = 1$.*

Sub-distributions are stable under pointwise-limits.

[10] We work with discrete distributions to keep measure-theoretic technicalities to a minimum, though we do not see obstacles to generalizing to the continuous setting.

Definition 6. *A sequence* $(\mu_n)_{n\in\mathbb{N}} \in \mathbf{SDist}(A)$ *sub-distributions* converges *if for every* $a \in A$*, the sequence* $(\mu_n(a))_{n\in\mathbb{N}}$ *of real numbers converges. The* limit sub-distribution *is defined as*

$$\mu_\infty(a) \triangleq \lim_{n\to\infty} \mu_n(a)$$

for every $a \in A$*. We write* $\lim_{n\to\infty} \mu_n$ *for* μ_∞*.*

Lemma 2. *Let* $(\mu_n)_{n\in\mathbb{N}}$ *be a convergent sequence of sub-distributions. Then for any event* $E(x)$*, we have:*

$$\forall n \in \mathbb{N}.\ \Pr_{x\sim\mu_\infty}[E(x)] = \lim_{n\to\infty} \Pr_{x\sim\mu_n}[E(x)].$$

Any bounded increasing real sequence has a limit; the same is true of sub-distributions.

Lemma 3. *Let* $(\mu_n)_{n\in\mathbb{N}} \in \mathbf{SDist}(A)$ *be an increasing sequence of sub-distributions. Then, this sequence converges to* μ_∞ *and* $\mu_n \le \mu_\infty$ *for every* $n \in \mathbb{N}$*. In particular, for any event* E*, we have* $\Pr_{x\sim\mu_n}[E] \le \Pr_{x\sim\mu_\infty}[E]$ *for every* $n \in \mathbb{N}$*.*

3 Programs and Assertions

Now, we introduce our core programming language and its denotational semantics.

Programs. We base our development on pWhile, a strongly-typed imperative language with deterministic assignments, probabilistic assignments, conditionals, loops, and an **abort** statement which halts the computation with no result. Probabilistic assignments $x \xleftarrow{\$} g$ assign a value sampled from a distribution g to a program variable x. The syntax of statements is defined by the grammar:

$$\begin{aligned} s ::= {} & \mathbf{skip} \mid \mathbf{abort} \mid x \leftarrow e \mid x \xleftarrow{\$} g \mid s; s \\ & \mid \mathbf{if}\ e\ \mathbf{then}\ s\ \mathbf{else}\ s \mid \mathbf{while}\ e\ \mathbf{do}\ s \mid x \leftarrow \mathcal{I}(e) \mid x \leftarrow \mathcal{A}(e) \end{aligned}$$

where x, e, and g range over typed variables in $\mathcal{X}$, expressions in $\mathcal{E}$ and distribution expressions in $\mathcal{D}$ respectively. The set $\mathcal{E}$ of well-typed expressions is defined inductively from $\mathcal{X}$ and a set $\mathcal{F}$ of function symbols, while the set $\mathcal{D}$ of well-typed distribution expressions is defined by combining a set of distribution symbols $\mathcal{S}$ with expressions in $\mathcal{E}$. Programs may call a set $\mathcal{I}$ of internal procedures as well as a set $\mathcal{A}$ of external procedures. We assume that we have code for internal procedures but not for external procedures—we only know indirect information, like which internal procedures they may call. Borrowing a convention from cryptography, we call internal procedures *oracles* and external procedures *adversaries*.

$$
\begin{aligned}
[\![\mathbf{skip}]\!]_m &= \delta_m \\
[\![\mathbf{abort}]\!]_m &= \mathbf{0} \\
[\![x \leftarrow e]\!]_m &= \delta_{m[x:=[\![e]\!]_m]} \\
[\![x \xleftarrow{\$} g]\!]_m &= \mathbb{E}_{v \sim [\![g]\!]_m}[\delta_{m[x:=v]}] \\
[\![s_1; s_2]\!]_m &= \mathbb{E}_{m' \sim [\![s_1]\!]_m}[[\![s_2]\!]_{m'}] \\
[\![\mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2]\!]_m &= \text{if } [\![e]\!]_m \text{ then } [\![s_1]\!]_m \text{ else } [\![s_2]\!]_m \\
[\![\mathbf{while}\ e\ \mathbf{do}\ s]\!]_m &= \lim_{n\to\infty} [\![(\mathbf{if}\ e\ \mathbf{then}\ s)^n; \mathbf{if}\ e\ \mathbf{then}\ \mathbf{abort}]\!]_m \\
[\![x \leftarrow \mathcal{I}(e)]\!]_m &= [\![f_{\mathbf{arg}} \leftarrow e; f_{\mathbf{body}}; x \leftarrow f_{\mathbf{res}}]\!]_m \\
[\![x \leftarrow \mathcal{A}(e)]\!]_m &= [\![a_{\mathbf{arg}} \leftarrow e; a_{\mathbf{body}}; x \leftarrow a_{\mathbf{res}}]\!]_m
\end{aligned}
$$

$$
[\![s]\!]_\mu = \mathbb{E}_{m \sim \mu}[[\![s]\!]_m]
$$

Fig. 1. Denotational semantics of programs

Semantics. The denotational semantics of programs is adapted from the seminal work of [27] and interprets programs as sub-distribution transformers. We view states as type-preserving mappings from variables to values; we write **State** for the set of states and **SDist**(**State**) for the set of probabilistic states. For each procedure name $f \in \mathcal{I} \cup \mathcal{A}$, we assume a set $\mathcal{X}_f^{\mathfrak{L}} \subseteq \mathcal{X}$ of *local variables* s.t. $\mathcal{X}_f^{\mathfrak{L}}$ are pairwise disjoint. The other variables $\mathcal{X} \setminus \bigcup_f \mathcal{X}_f^{\mathfrak{L}}$ are *global variables.*

To define the interpretation of expressions and distribution expressions, we let $[\![e]\!]_m$ denote the interpretation of expression e with respect to state m, and $[\![e]\!]_\mu$ denote the interpretation of expression e with respect to an initial sub-distribution μ over states defined by the clause $[\![e]\!]_\mu \triangleq \mathbb{E}_{m\sim\mu}[[\![e]\!]_m]$. Likewise, we define the semantics of commands in two stages: first interpreted in a single input memory, then interpreted in an input sub-distribution over memories.

Definition 7. *The semantics of commands are given in Fig. 1.*

- *The semantics $[\![s]\!]_m$ of a statement s in initial state m is a sub-distribution over states.*
- *The (lifted) semantics $[\![s]\!]_\mu$ of a statement s in initial sub-distribution μ over states is a sub-distribution over states.*

We briefly comment on loops. The semantics of a loop $\mathbf{while}\, e\, \mathbf{do}\, c$ is defined as the limit of its lower approximations, where the n-th *lower approximation* of $[\![\mathbf{while}\ e\ \mathbf{do}\ c]\!]_\mu$ is $[\![(\mathbf{if}\ e\ \mathbf{then}\ s)^n; \mathbf{if}\ e\ \mathbf{then}\ \mathbf{abort}]\!]_\mu$, where $\mathbf{if}\ e\ \mathbf{then}\ s$ is shorthand for $\mathbf{if}\, e\, \mathbf{then}\, s\, \mathbf{else}\, \mathbf{skip}$ and c^n is the n-fold composition $c; \cdots; c$. Since the sequence is increasing, the limit is well-defined by Lemma 3. In contrast, the n-th *approximation* of $[\![\mathbf{while}\, e\, \mathbf{do}\, c]\!]_\mu$ defined by $[\![(\mathbf{if}\ e\ \mathbf{then}\ s)^n]\!]_\mu$ may not converge, since they are not necessarily increasing. However, in the special case

where the output distribution has weight 1, the n-th lower approximations and the n-th approximations have the same limit.

Lemma 4. *If the sub-distribution* $[\![\mathbf{while}\, e\, \mathbf{do}\, c]\!]_\mu$ *has weight* 1*, then the limit of* $[\![(\mathbf{if}\, e\, \mathbf{then}\, s)^n]\!]_\mu$ *is defined and*

$$\lim_{n\to\infty} [\![(\mathbf{if}\, e\, \mathbf{then}\, s)^n; \mathbf{if}\, e\, \mathbf{then}\, \mathbf{abort}]\!]_\mu = \lim_{n\to\infty} [\![(\mathbf{if}\, e\, \mathbf{then}\, s)^n]\!]_\mu.$$

This follows by Lemma 1, since lower approximations are below approximations so the limit of their weights (and the weight of their limit) is 1. It will be useful to identify programs that terminate with probability 1.

Definition 8 (Lossless). *A statement* s *is* lossless *if for every sub-distribution* μ*,* $|[\![s]\!]_\mu| = |\mu|$*, where* $|\mu|$ *is the total probability of* μ*. Programs that are not lossless are called* lossy*.*

Informally, a program is lossless if all probabilistic assignments sample from full distributions rather than sub-distributions, there are no **abort** instructions, and the program is almost surely terminating, i.e. infinite traces have probability zero. Note that if we restrict the language to sample from full distributions, then losslessness coincides with almost sure termination.

Another important class of loops are loops with a uniform upper bound on the number of iterations. Formally, we say that a loop $\mathbf{while}\, e\, \mathbf{do}\, s$ is *certainly terminating* if there exists k such that for every sub-distribution μ, we have $|[\![\mathbf{while}\, e\, \mathbf{do}\, s]\!]_\mu| = |[\![(\mathbf{if}\, e\, \mathbf{then}\, s)^k]\!]_\mu|$. Note that certain termination of a loop does not entail losslessness—the output distribution of the loop may not have weight 1, for instance, if the loop samples from a sub-distribution or if the loop aborts with positive probability.

Semantics of Procedure Calls and Adversaries. The semantics of internal procedure calls is straightforward. Associated to each procedure name $f \in \mathcal{I}$, we assume a designated input variable $f_{\mathbf{arg}} \in \mathcal{X}_f^{\mathfrak{L}}$, a piece of code $f_{\mathbf{body}}$ that executes the function call, and a result expression $f_{\mathbf{res}}$. A function call $x \leftarrow \mathcal{I}(e)$ is then equivalent to $f_{\mathbf{arg}} \leftarrow e; f_{\mathbf{body}}; x \leftarrow f_{\mathbf{res}}$. Procedures are subject to well-formedness criteria: procedures should only use local variables in their scope and after initializing them, and should not perform recursive calls.

External procedure calls, also known as adversary calls, are a bit more involved. Each name $a \in \mathcal{A}$ is parametrized by a set $a_{\mathbf{ocl}} \subseteq \mathcal{I}$ of internal procedures which the adversary may call, a designated input variable $a_{\mathbf{arg}} \in \mathcal{X}_a^{\mathfrak{L}}$, a (unspecified) piece of code $a_{\mathbf{body}}$ that executes the function call, and a result expression $a_{\mathbf{res}}$. We assume that adversarial code can only access its local variables in $\mathcal{X}_a^{\mathfrak{L}}$ and can only make calls to procedures in $a_{\mathbf{ocl}}$. It is possible to impose more restrictions on adversaries—say, that they are lossless—but for simplicity we do not impose additional assumptions on adversaries here.

4 Proof System

In this section we introduce a program logic for proving properties of probabilistic programs. The logic is abstract—assertions are arbitrary predicates on sub-distributions—but the meta-theoretic properties are clearest in this setting. In the following section, we will give a concrete version suitable for practical use.

Assertions and Closedness Conditions. We use predicates on state distribution.

Definition 9 (Assertions). *The set* Assn *of assertions is defined as* $\mathcal{P}(\mathbf{SDist}(\mathbf{State}))$. *We write* $\eta(\mu)$ *for* $\mu \in \eta$.

Usual set operations are lifted to assertions using their logical counterparts, e.g., $\eta \wedge \eta' \triangleq \eta \cap \eta'$ and $\neg\eta \triangleq \overline{\eta}$. Our program logic uses a few additional constructions. Given a predicate ϕ over states, we define

$$\Box\phi(\mu) \triangleq \forall m.\, m \in \mathrm{supp}(\mu) \implies \phi(m)$$

where $\mathrm{supp}(\mu)$ is the set of all states with non-zero probability under μ. Intuitively, ϕ holds deterministically on all states that we may sample from the distribution. To reason about branching commands, given two assertions η_1 and η_2, we let

$$(\eta_1 \oplus \eta_2)(\mu) \triangleq \exists \mu_1, \mu_2.\, \mu = \mu_1 + \mu_2 \wedge \eta_1(\mu_1) \wedge \eta_2(\mu_2).$$

This assertion means that the sub-distribution is the sum of two sub-distributions such that η_1 holds on the first piece and η_2 holds on the second piece.

Given an assertion η and an event $E \subseteq \mathbf{State}$, we let $\eta_{|E}(\mu) \triangleq \eta(\mu_{|E})$. This assertion holds exactly when η is true on the portion of the sub-distribution satisfying E. Finally, given an assertion η and a function F from $\mathbf{SDist}(\mathbf{State})$ to $\mathbf{SDist}(\mathbf{State})$, we define $\eta[F] \triangleq \lambda\mu.\, \eta(F(\mu))$. Intuitively, $\eta[F]$ is true in a sub-distribution μ exactly when η holds on $F(\mu)$.

Now, we can define the closedness properties of assertions. These properties will be critical to our rules for **while** loops.

Definition 10 (Closedness properties). *A family of assertions* $(\eta_n)_{n\in\mathbb{N}^\infty}$ *is:*

- u-closed *if for every increasing sequence of sub-distributions* $(\mu_n)_{n\in\mathbb{N}}$ *such that* $\eta_n(\mu_n)$ *for all* $n \in \mathbb{N}$ *then* $\eta_\infty(\lim_{n\to\infty} \mu_n)$*;*
- t-closed *if for every converging sequence of sub-distributions* $(\mu_n)_{n\in\mathbb{N}}$ *such that* $\eta_n(\mu_n)$ *for all* $n \in \mathbb{N}$ *then* $\eta_\infty(\lim_{n\to\infty} \mu_n)$*;*
- d-closed *if it is* t*-closed and downward closed, that is for every sub-distributions* $\mu \leq \mu'$, $\eta_\infty(\mu')$ *implies* $\eta_\infty(\mu)$.

When $(\eta_n)_n$ *is constant and equal to* η*, we say that* η *is* u*-/*t*-/*d*-closed.*

Note that t-closedness implies u-closedness, but the converse does not hold. Moreover, u-closed, t-closed and d-closed assertions are closed under arbitrary

intersections and finite unions, or in logical terms under finite boolean combinations, universal quantification over arbitrary sets and existential quantification over finite sets.

Finally, we introduce the necessary machinery for the frame rule. The set $\mathrm{mod}(s)$ of *modified* variables of a statement s consists of all the variables on the left of a deterministic or probabilistic assignment. In this setting, we say that an assertion η is *separated* from a set of variables X, written $\mathsf{separated}(\eta, X)$, if $\eta(\mu_1) \iff \eta(\mu_2)$ for any distributions μ_1, μ_2 s.t. $|\mu_1| = |\mu_2|$ and $\mu_{1|\overline{X}} = \mu_{2|\overline{X}}$ where for a set of variables X, the restricted sub-distribution $\mu_{|X}$ is

$$\mu_{|X} : m \in \mathbf{State}_{|X} \mapsto \Pr_{m' \sim \mu}[m = m'_{|X}]$$

where $\mathbf{State}_{|X}$ and $m_{|X}$ restrict $\mathbf{State}$ and m to the variables in X.

Intuitively, an assertion is separated from a set of variables X if every two sub-distributions that agree on the variables outside X either both satisfy the assertion, or both refute the assertion.

Judgments and Proof Rules. Judgments are of the form $\{\eta\}\ s\ \{\eta'\}$, where the assertions η and η' are drawn from Assn.

Definition 11. *A judgment* $\{\eta\}\ s\ \{\eta'\}$ *is* valid*, written* $\models \{\eta\}\ s\ \{\eta'\}$*, if* $\eta'([\![s]\!]_\mu)$ *for every interpretation of adversarial procedures and every probabilistic state* μ *such that* $\eta(\mu)$*.*

Figure 2 describes the structural and basic rules of the proof system. Validity of judgments is preserved under standard structural rules, like the rule of consequence [Conseq]. As usual, the rule of consequence allows to weaken the post-condition and to strengthen the post-condition; in our system, this rule serves as the interface between the program logic and mathematical theorems from probability theory. The [Exists] rule is helpful to deal with existentially quantified pre-conditions.

The rules for **skip**, assignments, random samplings and sequences are all straightforward. The rule for **abort** requires $\square\bot$ to hold after execution; this assertion uniquely characterizes the resulting null sub-distribution. The rules for assignments and random samplings are semantical.

The rule [Cond] for conditionals requires that the post-condition must be of the form $\eta_1 \oplus \eta_2$; this reflects the semantics of conditionals, which splits the initial probabilistic state depending on the guard, runs both branches, and recombines the resulting two probabilistic states.

The next two rules ([Split] and [Frame]) are useful for local reasoning. The [Split] rule reflects the additivity of the semantics and combines the pre- and post-conditions using the $\oplus$ operator. The [Frame] rule asserts that lossless statements preserve assertions that are not influenced by modified variables.

The rule [Call] for internal procedures is as expected, replacing the procedure call f with its definition.

Figure 3 presents the rules for loops. We consider four rules specialized to the termination behavior. The [While] rule is the most general rule, as it deals with

$$\frac{\eta_0 \Rightarrow \eta_1 \qquad \{\eta_1\}\ s\ \{\eta_2\} \qquad \eta_2 \Rightarrow \eta_3}{\{\eta_0\}\ s\ \{\eta_3\}}\ [\textsc{Conseq}] \qquad \frac{\forall x : T.\, \{\eta\}\ s\ \{\eta'\}}{\{\exists x : T.\, \eta\}\ s\ \{\eta'\}}\ [\textsc{Exists}]$$

$$\frac{}{\{\eta\}\ \mathbf{abort}\ \{\Box\bot\}}\ [\textsc{Abort}] \qquad \frac{\eta' \triangleq \eta[[\![x \leftarrow e]\!]]}{\{\eta'\}\ x \leftarrow e\ \{\eta\}}\ [\textsc{Assgn}] \qquad \frac{}{\{\eta\}\ \mathbf{skip}\ \{\eta\}}\ [\textsc{Skip}]$$

$$\frac{\eta' \triangleq \eta[[\![x \xleftarrow{\$} g]\!]]}{\{\eta'\}\ x \xleftarrow{\$} g\ \{\eta\}}\ [\textsc{Sample}] \qquad \frac{\{\eta_0\}\ s_1\ \{\eta_1\} \qquad \{\eta_1\}\ s_2\ \{\eta_2\}}{\{\eta_0\}\ s_1; s_2\ \{\eta_2\}}\ [\textsc{Seq}]$$

$$\frac{\{\eta_1 \wedge \Box e\}\ s_1\ \{\eta_1'\} \qquad \{\eta_2 \wedge \Box\neg e\}\ s_2\ \{\eta_2'\}}{\{(\eta_1 \wedge \Box e) \oplus (\eta_2 \wedge \Box\neg e)\}\ \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \{\eta_1' \oplus \eta_2'\}}\ [\textsc{Cond}]$$

$$\frac{\{\eta_1\}\ s\ \{\eta_1'\} \qquad \{\eta_2\}\ s\ \{\eta_2'\}}{\{\eta_1 \oplus \eta_2\}\ s\ \{\eta_1' \oplus \eta_2'\}}\ [\textsc{Split}]$$

$$\frac{\mathsf{separated}(\eta, \mathrm{mod}(s)) \qquad s \text{ is lossless}}{\{\eta\}\ s\ \{\eta\}}\ [\textsc{Frame}]$$

$$\frac{\{\eta\}\ f_{\mathbf{arg}} \leftarrow e; f_{\mathbf{body}}\ \{\eta'[[\![x \leftarrow f_{\mathbf{res}}]\!]]\}}{\{\eta\}\ x \leftarrow f(e)\ \{\eta'\}}\ [\textsc{Call}]$$

Fig. 2. Structural and basic rules

arbitrary loops. For simplicity, we explain the rule in the special case where the family of assertions is constant, i.e. we have $\eta_n = \eta$ and $\eta'_n = \eta'$. Informally, the η is the loop invariant and η' is an auxiliary assertion used to prove the invariant. We require that η is u-closed, since the semantics of a loop is defined as the limit of its lower approximations. Moreover, the first premise ensures that starting from η, one guarded iteration of the loop establishes η'; the second premise ensures that restricting to $\neg e$ a probabilistic state μ' satisfying η' yields a probabilistic state μ satisfying η. It is possible to give an alternative formulation where the second premise is substituted by the logical constraint ${\eta'}_{|\neg e} \implies \eta$. As usual, the post-condition of the loop is the conjunction of the invariant with the negation of the guard (more precisely in our setting, that the guard has probability 0).

The [While-AST] rule deals with lossless loops. For simplicity, we explain the rule in the special case where the family of assertions is constant, i.e. we have $\eta_n = \eta$. In this case, we know that lower approximations and approximations have the same limit, so we can directly prove an invariant that holds after one guarded iteration of the loop. On the other hand, we must now require that the η satisfies the stronger property of t-closedness.

The [While-D] rule handles arbitrary loops with a d-closed invariant; intuitively, restricting a sub-distribution that satisfies a downwards closed assertion η yields a sub-distribution which also satisfies η.

The [While-CT] rule deals with certainly terminating loops. In this case, there is no requirement on the assertions.

We briefly compare the rules from a verification perspective. If the assertion is d-closed, then the rule [While-D] is easier to use, since there is no need to prove any termination requirement. Alternatively, if we can prove certain termination of the loop, then the rule [While-CT] is the best to use since it does not impose any condition on assertions. When the loop is lossless, there is no need to introduce an auxiliary assertion η', which simplifies the proof goal. Note however that it might still be beneficial to use the [While] rule, even for lossless loops, because of the weaker requirement that the invariant is u-closed rather than t-closed.

Finally, Fig. 4 gives the adversary rule for general adversaries. It is highly similar to the general rule [While-D] for loops since the adversary may make an arbitrary sequence of calls to the oracles in $a_{\mathbf{ocl}}$ and may not be lossless. Intuitively, η plays the role of the invariant: it must be d-closed and it must be preserved by every oracle call with arbitrary arguments. If this holds, then η is also preserved by the adversary call. Some framing conditions are required, similar to the ones of the [Frame] rule: the invariant must not be influenced by the state writable by the external procedures.

It is possible to give other variants of the adversary rule with more general invariants by restricting the adversary, e.g., requiring losslessness or bounding the number of calls the external procedure can make to oracles, leading to rules akin to the almost surely terminating and certainly terminating loop rules, respectively.

$$\frac{\begin{array}{c}\mathsf{uclosed}((\eta'_n)_{n\in\mathbb{N}^\infty})\\ \forall n.\,\{\eta_n\}\ \mathbf{if}\ e\ \mathbf{then}\ s\ \{\eta_{n+1}\} \qquad \forall n.\,\{\eta_n\}\ \mathbf{if}\ e\ \mathbf{then\ abort}\ \{\eta'_n\}\end{array}}{\{\eta_0\}\ \mathbf{while}\ e\ \mathbf{do}\ s\ \{\eta'_\infty \wedge \Box\neg e\}}\ [\textsc{While}]$$

$$\frac{\begin{array}{c}\mathsf{tclosed}((\eta_n)_{n\in\mathbb{N}^\infty}) \qquad \forall n.\,\{\eta_n\}\ \mathbf{if}\ e\ \mathbf{then}\ s\ \{\eta_{n+1}\}\\ \forall \mu.\,\eta_0(\mu) \implies |[\![(\mathbf{while}\ e\ \mathbf{do}\ s)]\!]_\mu| = 1\end{array}}{\{\eta_0\}\ \mathbf{while}\ e\ \mathbf{do}\ s\ \{\eta_\infty \wedge \Box\neg e\}}\ [\textsc{While-AST}]$$

$$\frac{\mathsf{dclosed}((\eta_n)_{n\in\mathbb{N}^\infty}) \qquad \forall n.\,\{\eta_n\}\ \mathbf{if}\ e\ \mathbf{then}\ s\ \{\eta_{n+1}\}}{\{\eta_0\}\ \mathbf{while}\ e\ \mathbf{do}\ s\ \{\eta_\infty \wedge \Box\neg e\}}\ [\textsc{While-D}]$$

$$\frac{\begin{array}{c}\forall n.\,\{\eta_n\}\ \mathbf{if}\ e\ \mathbf{then}\ s\ \{\eta_{n+1}\}\\ \forall \mu.\,\eta_0(\mu) \implies [\![(\mathbf{if}\ e\ \mathbf{then}\ s)^k]\!]_\mu = [\![(\mathbf{while}\ e\ \mathbf{do}\ s)]\!]_\mu\end{array}}{\{\eta_0\}\ \mathbf{while}\ e\ \mathbf{do}\ s\ \{\eta_k \wedge \Box\neg e\}}\ [\textsc{While-CT}]$$

Fig. 3. Rules for loops

$$\frac{\begin{array}{c}\forall n \in \mathbb{N}^\infty.\ \mathsf{separated}(\eta_n, \{x, \mathfrak{s}\}) \qquad \mathsf{dclosed}((\eta_n)_{n\in\mathbb{N}^\infty})\\ \forall f \in a_{\mathbf{ocl}}, x \in \mathcal{X}_a^{\mathfrak{L}}, e \in \mathcal{E}, n \in \mathbb{N}.\,\{\eta_n\}\ x \leftarrow f(e)\ \{\eta_{n+1}\}\end{array}}{\{\eta_0\}\ x \leftarrow a(e)\ \{\eta_\infty\}}\ [\textsc{Adv}]$$

Fig. 4. Rules for adversaries

Soundness and Relative Completeness. Our proof system is sound and relatively complete with respect to the semantics; these proofs have also been formalized in the Coq proof assistant.

Theorem 1 (Soundness). *Every judgment* $\{\eta\}\ s\ \{\eta'\}$ *provable using the rules of our logic is valid.*

Completeness of the logic follows from the next lemma, whose proof makes an essential use of the [While] rule. In the sequel, we use $\mathbf{1}_\mu$ to denote the characteristic function of a probabilistic state μ, an assertion stating that the current state is equal to μ.

Lemma 5. *For every probabilistic state* μ*, the following judgment is provable using the rule of the logic:*

$$\{\mathbf{1}_\mu\}\ s\ \{\mathbf{1}_{[\![s]\!]_\mu}\}.$$

Proof. By induction on the structure of s.

- $s = \mathbf{abort}$, $s = \mathbf{skip}$, $x \leftarrow e$ and $s = x \xleftarrow{\$} g$ are trivial;
- $s = s_1; s_2$, we have to prove

  $$\{\mathbf{1}_\mu\}\ s_1; s_2\ \{\mathbf{1}_{[\![s_2]\!]_{[\![s_1]\!]_\mu}}\}.$$

  We apply the [Seq] rule with $\eta_1 = \mathbf{1}_{[\![s_1]\!]_\mu}$ premises can be directly proved using the induction hypothesis;
- $s = \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2$, we have to prove

  $$\{\mathbf{1}_\mu\}\ \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \{(\mathbf{1}_{[\![s_1]\!]_{\mu_{|e}}} \oplus \mathbf{1}_{[\![s_2]\!]_{\mu_{|\neg e}}})\}.$$

  We apply the [Conseq] rule to be able to apply the the [Cond] rule with $\eta_1 = \mathbf{1}_{[\![s_1]\!]_{\mu_{|e}}}$ and $\eta_2 = \mathbf{1}_{[\![s_2]\!]_{\mu_{|\neg e}}}$ Both premises can be proved by an application of the [Conseq] rule followed by the application of the induction hypothesis.
- $s = \mathbf{while}\ e\ \mathbf{do}\ s$, we have to prove

  $$\{\mathbf{1}_\mu\}\ \mathbf{while}\ e\ \mathbf{do}\ s\ \{\mathbf{1}_{\lim_{n\to\infty}\ [\![(\mathbf{if}\ e\ \mathbf{then}\ s)^n;\mathbf{if}\ e\ \mathbf{then\ abort}]\!]_\mu}\}.$$

  We first apply the [While] rule with $\eta'_n = \mathbf{1}_{[\![(\mathbf{if}\ e\ \mathbf{then}\ s)^n]\!]_\mu}$ and

  $$\eta_n = \mathbf{1}_{[\![(\mathbf{if}\ e\ \mathbf{then}\ s)^n;\mathbf{if}\ e\ \mathbf{then\ abort}]\!]_\mu}.$$

  For the first premise we apply the same process as for the conditional case: we apply the [Conseq] and [Cond] rules and we conclude using the induction hypothesis (and the [Skip] rule). For the second premise we follow the same process but we conclude using the [Abort] rule instead of the induction hypothesis. Finally we conclude since $\mathsf{uclosed}((\eta_n)_{n\in\mathbb{N}^\infty})$. □

The abstract logic is also relatively complete. This property will be less important for our purposes, but it serves as a basic sanity check.

Theorem 2 (Relative completeness). *Every valid judgment is derivable.*

Proof. Consider a valid judgment $\{\eta\}\ s\ \{\eta'\}$. Let μ be a probabilistic state such that $\eta(\mu)$. By the above proposition, $\{\mathbf{1}_\mu\}\ s\ \{\mathbf{1}_{[\![s]\!]_\mu}\}$. Using the validity of the judgment and [Conseq], we have $\{\mathbf{1}_\mu \wedge \eta(\mu)\}\ s\ \{\eta'\}$. Using the [Exists] and [Conseq] rules, we conclude $\{\eta\}\ s\ \{\eta'\}$ as required. □

The side-conditions in the loop rules (e.g., uclosed/tclosed/dclosed and the weight conditions) are difficult to prove, since they are semantic properties. Next, we present a concrete version of the logic with give easy-to-check, syntactic sufficient conditions.

5 A Concrete Program Logic

To give a more practical version of the logic, we begin by setting a concrete syntax for assertions

Assertions. We use a two-level assertion language, presented in Fig. 5. A *probabilistic assertion* η is a formula built from comparison of probabilistic expressions, using first-order quantifiers and connectives, and the special connective $\oplus$. A *probabilistic expression* p can be a logical variable v, an operator applied to probabilistic expressions $o(\boldsymbol{p})$ (constants are 0-ary operators), or the expectation $\mathbb{E}[\tilde{e}]$ of a state expression $\tilde{e}$. A *state expression* $\tilde{e}$ is either a program variable x, the characteristic function $\mathbf{1}_\phi$ of a state assertion ϕ, an operator applied to state expressions $o(\tilde{\boldsymbol{e}})$, or the expectation $\mathbb{E}_{v\sim g}[\tilde{e}]$ of state expression $\tilde{e}$ in a given distribution g. Finally, a *state assertion* ϕ is a first-order formula over program variables. Note that the set of operators is left unspecified but we assume that all the expressions in $\mathcal{E}$ and $\mathcal{D}$ can be encoded by operators.

$$\tilde{e} ::= x \mid v \mid \mathbf{1}_\phi \mid \mathbb{E}_{v\sim g}[\tilde{e}] \mid o(\tilde{\boldsymbol{e}}) \qquad \text{(S-expr.)}$$
$$\phi ::= \tilde{e} \bowtie \tilde{e} \mid FO(\phi) \qquad \text{(S-assn.)}$$
$$p ::= v \mid o(\boldsymbol{p}) \mid \mathbb{E}[\tilde{e}] \qquad \text{(P-expr.)}$$
$$\eta ::= p \bowtie p \mid \eta \oplus \eta \mid FO(\eta) \qquad \text{(P-assn.)}$$
$$\bowtie \in \{=, <, \leq\} \qquad o \in Ops \qquad \text{(Ops.)}$$

Fig. 5. Assertion syntax

The interpretation of the concrete syntax is as expected. The interpretation of probabilistic assertions is relative to a valuation ρ which maps logical variables to values, and is an element of Assn. The definition of the interpretation is straightforward; the only interesting case is $[\![\mathbb{E}[\tilde{e}]]\!]^\rho_\mu$ which is defined by $\mathbb{E}_{m\sim\mu}[[\![\tilde{e}]\!]^\rho_m]$, where $[\![\tilde{e}]\!]^\rho_m$ is the interpretation of the state expression $\tilde{e}$ in the memory m and valuation ρ. The interpretation of state expressions is a mapping from memories to values, which can be lifted to a mapping from distributions over memories to distributions over values. The definition of the interpretation is straightforward; the most interesting case is for expectation $[\![\mathbb{E}_{v\sim g}[\tilde{e}]]\!]^\rho_m \triangleq \mathbb{E}_{w\sim[\![g]\!]^\rho_m}[[\![\tilde{e}]\!]^{\rho[v:=w]}_m]$. We present the full interpretations in the supplemental materials.

Many standard concepts from probability theory have a natural representation in our syntax. For example:

- the probability that ϕ holds in some probabilistic state is represented by the probabilistic expression $\Pr[\phi] \triangleq \mathbb{E}[\mathbf{1}_\phi]$;
- probabilistic independence of state expressions $\tilde{e}_1, \ldots, \tilde{e}_n$ is modeled by the probabilistic assertion $\#\{\tilde{e}_1, \ldots, \tilde{e}_n\}$, defined by the clause[11]

$$\forall v_1 \ldots v_n,\ \ \Pr[\top]^{n-1} \Pr[\bigwedge_{i=1\ldots n} \tilde{e}_i = v_i] = \prod_{i=1\ldots n} \Pr[\tilde{e}_i = v_i];$$

- the fact that a distribution is proper is modeled by the probabilistic assertion $\mathcal{L} \triangleq \Pr[\top] = 1$;
- a state expression $\tilde{e}$ distributed according to a law g is modeled by the probabilistic assertion

$$\tilde{e} \sim g \triangleq \forall w,\ \ \Pr[\tilde{e} = w] = \mathbb{E}[\mathbb{E}_{v \sim g}[\mathbf{1}_{v=w}]].$$

  The inner expectation computes the probability that v drawn from g is equal to a fixed w; the outer expectation weights the inner probability by the probability of each value of w.

We can easily define $\Box$ operator from the previous section in our new syntax: $\Box\phi \triangleq \Pr[\neg\phi] = 0$.

Syntactic Proof Rules. Now that we have a concrete syntax for assertions, we can give syntactic versions of many of the existing proof rules. Such proof rules are often easier to use since they avoid reasoning about the semantics of commands and assertions. We tackle the non-looping rules first, beginning with the following syntactic rules for assignment and sampling:

$$\frac{}{\{\eta[x := e]\}\ x \leftarrow e\ \{\eta\}}\ [\textsc{Assgn}] \qquad \frac{}{\{\mathcal{P}^g_x(\eta)\}\ x \xleftarrow{\$} g\ \{\eta\}}\ [\textsc{Sample}]$$

The rule for assignment is the usual rule from Hoare logic, replacing the program variable x by its corresponding expression e in the pre-condition. The replacement $\eta[x := e]$ is done recursively on the probabilistic assertion η; for instance for expectations, it is defined by $\mathbb{E}[\tilde{e}][x := e] \triangleq \mathbb{E}[\tilde{e}[x := e]]$, where $\tilde{e}[x := e]$ is the syntactic substitution.

The rule for sampling uses probabilistic substitution operator $\mathcal{P}^g_x(\eta)$, which replaces all occurrences of x in η by a new integration variable t and records that t is drawn from g; the operator is defined in Fig. 6.

Next, we turn to the loop rule. The side-conditions from Fig. 3 are purely semantic, while in practice it is more convenient to use a sufficient condition in the Hoare logic. We give sufficient conditions for ensuring certain and almost-sure termination in Fig. 7; $\tilde{e}$ is an integer-valued expression. The first side-condition $\mathcal{C}_{\text{CTerm}}$ shows certain termination given a strictly decreasing *variant* $\tilde{e}$

[11] The term $\Pr[\top]^{n-1}$ is necessary since we work with sub-distributions.

$$
\begin{array}{ll}
\mathcal{P}_x^g(v) & \triangleq v \\
\mathcal{P}_x^g(\mathbb{E}[\tilde{e}]) & \triangleq \mathbb{E}[\mathbb{E}_{t\sim g}[\tilde{e}[x := t]]] \\
\mathcal{P}_x^g(o(\boldsymbol{\eta})) & \triangleq o(\mathcal{P}_x^g(\eta_1), \ldots, \mathcal{P}_x^g(\eta_n)) \\
\mathcal{P}_x^g(\eta_1 \bowtie \eta_2) & \triangleq \mathcal{P}_x^g(\eta_1) \bowtie \mathcal{P}_x^g(\eta_2)
\end{array}
$$

for $o \in \mathbf{Ops}, \bowtie \in \{\wedge, \vee, \Rightarrow\}$.

Fig. 6. Syntactic op. $\mathcal{P}$ (main cases)

$$
\begin{array}{rl}
\mathcal{C}_{\mathrm{CTerm}} \triangleq & \{\mathcal{L} \wedge \Box(\tilde{e} = k \wedge 0 < k \wedge b)\}\ s\ \{\mathcal{L} \wedge \Box(\tilde{e} < k)\} \\
& \models \eta \Rightarrow (\exists \dot{y}.\ \Box \tilde{e} \leq \dot{y}) \wedge \Box(\tilde{e} = 0 \Rightarrow \neg b) \\
\mathcal{C}_{\mathrm{ASTerm}} \triangleq & \{\mathcal{L} \wedge \Box(\tilde{e} = k \wedge 0 < k \leq K \wedge b)\}\ s\ \{\mathcal{L} \wedge \Box(0 \leq \tilde{e} \leq K) \wedge \Pr[\tilde{e} < k] \geq \epsilon\} \\
& \models \eta \Rightarrow \Box(0 \leq \tilde{e} \leq K \wedge \tilde{e} = 0 \Rightarrow \neg b) \\
& \models \mathsf{tclosed}(\eta)
\end{array}
$$

Fig. 7. Side-conditions for loop rules

that is bounded below, similar to how a decreasing variant shows termination for deterministic programs. The second side-condition $\mathcal{C}_{\mathrm{ASTerm}}$ shows almost-sure termination given a probabilistic variant $\tilde{e}$, which must be bounded both above and below. While $\tilde{e}$ may increase with some probability, it must decrease with strictly positive probability. This condition was previously considered by [17] for probabilistic transition systems and also used in expectation-based approaches [33,20]. Our framework can also support more refined conditions (e.g., based on super-martingales [9,31]), but the condition $\mathcal{C}_{\mathrm{ASTerm}}$ already suffices for most randomized algorithms.

While t-closedness is a semantic condition (cf. Definition 10), there are simple syntactic conditions to guarantee it. For instance, assertions that carry a non-strict comparison $\bowtie \in \{\leq, \geq, =\}$ between two bounded probabilistic expressions are t-closed; the assertion stating probabilistic independence of a set of expressions is t-closed.

Precondition Calculus. With a concrete syntax for assertions, we are also able to incorporate syntactic reasoning principles. One classic tool is Morgan and McIver's *greatest pre-expectation*, which we take as inspiration for a pre-condition calculus for the loop-free fragment of Ellora. Given an assertion η and a loop-free statement s, we mechanically construct an assertion η^* that is the pre-condition of s that implies η as a post-condition. The basic idea is to replace each expectation expression p inside η by an expression p^* that has the same denotation before running s as p after running s. This process yields an assertion η^* that, interpreted before running s, is logically equivalent to η interpreted after running s.

The computation rules for pre-conditions are defined in Fig. 8. For a probability assertion η, its pre-condition $\mathrm{pc}(s, \eta)$ corresponds to η where the expectation ex-

pressions of the form $\mathbb{E}[\tilde{e}]$ are replaced by their corresponding *pre-term*, $\mathrm{pe}(s, \mathbb{E}[\tilde{e}])$. Pre-terms correspond loosely to Morgan and McIver's *pre-expectations*—we will make this correspondence more precise in the next section. The main interesting cases for computing pre-terms are for random sampling and conditionals. For random sampling the result is $\mathcal{P}_x^g(\mathbb{E}[\tilde{e}])$, which corresponds to the [Sample] rule. For conditionals, the expectation expression is split into a part where e is true and a part where e is not true. We restrict the expectation to a part satisfying e with the operator $\mathbb{E}[\tilde{e}]_{|e} \triangleq \mathbb{E}[\tilde{e} \cdot \mathbf{1}_e]$. This corresponds to the expected value of $\tilde{e}$ on the portion of the distribution where e is true. Then, we can build the pre-condition calculus into Ellora.

$$\begin{aligned}
\mathrm{pe}(s_1; s_2, \mathbb{E}[\tilde{e}]) &\triangleq \mathrm{pe}(s_1, \mathrm{pe}(s_2, \mathbb{E}[\tilde{e}])) \\
\mathrm{pe}(x \leftarrow e, \mathbb{E}[\tilde{e}]) &\triangleq \mathbb{E}[\tilde{e}][x := e] \\
\mathrm{pe}(x \xleftarrow{\$} g, \mathbb{E}[\tilde{e}]) &\triangleq \mathcal{P}_x^g(\mathbb{E}[\tilde{e}]) \\
\mathrm{pe}(\textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2, \mathbb{E}[\tilde{e}]) &\triangleq \mathrm{pe}(s_1, \mathbb{E}[\tilde{e}])_{|e} + \mathrm{pe}(s_2, \mathbb{E}[\tilde{e}])_{|\neg e} \\
\\
\mathrm{pc}(s, p_1 \bowtie p_2) &\triangleq \mathrm{pe}(s, p_1) \bowtie \mathrm{pe}(s, p_2)
\end{aligned}$$

Fig. 8. Precondition calculus (selected)

Theorem 1. *Let s be a non-looping command. Then, the following rule is derivable in the concrete version of* Ellora*:*

$$\frac{}{\{pc(s, \eta)\}\ s\ \{\eta\}}\ [\mathrm{PC}]$$

6 Case Studies: Embedding Lightweight Logics

While Ellora is suitable for general-purpose reasoning about probabilistic programs, in practice humans typically use more special-purpose proof techniques—often targeting just a single, specific kind of property, like probabilistic independence—when proving probabilistic assertions. When these techniques apply, they can be a convenient and powerful tool.

To capture this intuitive style of reasoning, researchers have considered lightweight program logics where the assertions and proof rules are tailored to a specific proof technique. We demonstrate how to integrate these tools in an assertion-based logic by introducing and embedding a new logic for reasoning about independence and distribution laws, useful properties when analyzing randomized algorithms. We crucially rely on the rich assertions in Ellora—it is not clear how to extend expectation-based approaches to support similar, lightweight reasoning. Then, we show to embed the union bound logic [4] for proving accuracy bounds.

6.1 Law and Independence Logic

We begin by describing the law and independence logic IL, a proof system with intuitive rules that are easy to apply and amenable to automation. For simplicity, we only consider programs which sample from the binomial distribution, and have deterministic control flow—for lack of space, we also omit procedure calls.

Definition 12 (Assertions). IL *assertions have the grammar:*

$$\xi := \mathsf{det}(e) \mid \#\, E \mid e \sim \mathrm{B}(e,p) \mid \top \mid \bot \mid \xi \wedge \xi$$

where $e \in \mathcal{E}$*,* $E \subseteq \mathcal{E}$*, and* $p \in [0,1]$*.*

The assertion $\mathsf{det}(e)$ states that e is deterministic in the current distribution, i.e., there is at most one element in the support of its interpretation. The assertion $\#\, E$ states that the expressions in E are independent, as formalized in the previous section. The assertion $e \sim \mathrm{B}(m,p)$ states that e is distributed according to a binomial distribution with parameter m (where m can be an expression) and constant probability p, i.e. the probability that $e = k$ is equal to the probability that exactly k independent coin flips return heads using a biased coin that returns heads with probability p.

Assertions can be seen as an instance of a logical abstract domain, where the order between assertions is given by implication based on a small number of axioms. Examples of such axioms include independence of singletons, irreflexivity of independence, anti-monotonicity of independence, an axiom for the sum of binomial distributions, and rules for deterministic expressions:

$$\#\{x\} \qquad \#\{x,x\} \iff \mathsf{det}(x) \qquad \#(E \cup E') \implies \#\, E$$

$$e \sim \mathrm{B}(m,p) \wedge e' \sim \mathrm{B}(m',p) \wedge \#\{e,e'\} \implies e + e' \sim \mathrm{B}(m+m',p)$$

$$\bigwedge_{1 \le i \le n} \mathsf{det}(e_i) \implies \mathsf{det}(f(e_1,\ldots,e_n))$$

Definition 13. *Judgments of the logic are of the form* $\{\xi\}\ s\ \{\xi'\}$*, where* ξ *and* ξ' *are* IL*-assertions. A judgment is* valid *if it is derivable from the rules of Fig. 9; structural rules and rule for sequential composition are similar to those from § 4 and omitted.*

The rule [IL-Assgn] for deterministic assignments is as in § 4. The rule [IL-Sample] for random assignments yields as post-condition that the variable x and a set of expressions E are independent assuming that E is independent before the sampling, and moreover that x follows the law of the distribution that it is sampled from. The rule [IL-Cond] for conditionals requires that the guard is deterministic, and that each of the branches satisfies the specification; if the guard

is not deterministic, there are simple examples where the rule is not sound.[12] The rule [IL-While] for loops requires that the loop is certainly terminating with a deterministic guard. Note that the requirement of certain termination could be avoided by restricting the structural rules such that a statement s has deterministic control flow whenever $\{\xi\}\ s\ \{\xi'\}$ is derivable.

We now turn to the embedding. The embedding of IL assertions into general assertions is immediate, except for $\mathsf{det}(e)$ which is translated as $\Box e \vee \Box\neg e$. We let $\overline{\xi}$ denote the translation of ξ.

Theorem 2 (Embedding and soundness of IL logic)**.** *If $\{\xi\}\ s\ \{\xi'\}$ is derivable in the* IL *logic, then $\{\overline{\xi}\}\ s\ \{\overline{\xi'}\}$ is derivable in (the syntactic variant of)* Ellora*. As a consequence, every derivable judgment $\{\xi\}\ s\ \{\xi'\}$ is valid.*

Proof sketch. By induction on the derivation. The interesting cases are conditionals and loops. For conditionals, the soundness follows from the soundness of the rule:

$$\frac{\{\eta\}\ s_1\ \{\eta'\} \qquad \{\eta\}\ s_2\ \{\eta'\} \qquad \Box e \vee \Box\neg e}{\{\eta\}\ \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \{\eta'\}}$$

To prove the soundness of this rule, we proceed by case analysis on $\Box e \vee \Box\neg e$. We treat the case $\Box e$; the other case is similar. In this case, η is equivalent to $\eta_1 \wedge \Box e \oplus \eta_2 \wedge \Box\neg e$, where $\eta_1 = \eta$ and $\eta_2 = \bot$. Let $\eta_1' = \eta'$ and $\eta_2 = \Box\bot$; again, $\eta_1' \oplus \eta_2'$ is logically equivalent to η'. The soundness of the rule thus follows from the soundness of the [Cond] and [Conseq] rules. For loops, there exists a natural number n such that $\mathbf{while}\, b\, \mathbf{do}\, s$ is semantically equivalent to $(\mathbf{if}\, b\, \mathbf{then}\, s)^n$. By assumption $\{\xi\}\ s\ \{\xi\}$ holds, and thus by induction hypothesis $\{\overline{\xi}\}\ s\ \{\overline{\xi}\}$. We also have $\xi \implies \mathsf{det}(b)$, and hence $\{\overline{\xi}\}\ \mathbf{if}\, b\, \mathbf{then}\, s\ \{\overline{\xi}\}$. We conclude by [Seq]. $\square$

To illustrate our system IL, consider the statement s in Fig. 10 which flips a fair coin N times and counts the number of heads. Using the logic, we prove that $\mathsf{c} \sim \mathrm{B}(N \cdot (N+1)/2, 1/2)$ is a post-condition for s. We take the invariant:

$$\mathsf{c} \sim \mathrm{B}\left(\mathsf{j}(\mathsf{j}+1)/2, 1/2\right)$$

The invariant holds initially, as $0 \sim B(0, 1/2)$. For the inductive case, we show:

$$\{\mathsf{c} \sim \mathrm{B}\left(0, 1/2\right)\}\ s_0\ \{\mathsf{c} \sim \mathrm{B}\left((\mathsf{j}+1)(\mathsf{j}+2)/2, 1/2\right)\}$$

where s_0 represents the loop body, i.e. $\mathsf{x} \xleftarrow{\$} \mathrm{B}\left(\mathsf{j}, 1/2\right); \mathsf{c} \leftarrow \mathsf{c} + \mathsf{x}$. First, we apply the rule for sequence taking as intermediate assertion

$$\mathsf{c} \sim \mathrm{B}\left(\mathsf{j}(\mathsf{j}+1)/2, 1/2\right) \wedge \mathsf{x} \sim \mathrm{B}\left(\mathsf{j}, 1/2\right) \wedge \#\{\mathsf{x}, \mathsf{c}\}$$

[12] Consider the following program where $\mathbf{Bern}(p)$ is the Bernoulli distribution with parameter p:

$$b \xleftarrow{\$} \mathbf{Bern}(p); \mathbf{if}\ b\ \mathbf{then}\ x_1 \xleftarrow{\$} \mathbf{Bern}(p_1); x_2 \xleftarrow{\$} \mathbf{Bern}(p_2)$$
$$\mathbf{else}\ x_1 \xleftarrow{\$} \mathbf{Bern}(p_1'); x_2 \xleftarrow{\$} \mathbf{Bern}(p_2')$$

Each branch establishes $\#\{x_1, x_2\}$, but this is not a valid post-condition for the conditional. There are similar examples using the binomial distribution.

$$\frac{}{\{\xi[x := e]\}\ x \leftarrow e\ \{\xi\}}\ [\text{IL-Assgn}]$$

$$\frac{\{x\} \cap \mathrm{FV}(E) \cap \mathrm{FV}(e) = \emptyset}{\{\#\, E\}\ x \xleftarrow{\$} \mathrm{B}(e,p)\ \{\#(E \cup \{x\}) \wedge x \sim \mathrm{B}(e,p)\}}\ [\text{IL-Sample}]$$

$$\frac{\{\xi\}\ s_1\ \{\xi'\} \qquad \{\xi'\}\ s_2\ \{\xi''\}}{\{\xi\}\ s_1; s_2\ \{\xi''\}}\ [\text{IL-Seq}]$$

$$\frac{\{\xi\}\ s_1\ \{\xi'\} \qquad \{\xi\}\ s_2\ \{\xi'\} \qquad \xi \implies \mathsf{det}(b)}{\{\xi\}\ \mathbf{if}\ b\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \{\xi'\}}\ [\text{IL-Cond}]$$

$$\frac{\{\xi\}\ s\ \{\xi\} \qquad \xi \implies \mathsf{det}(b) \qquad \mathcal{C}_{\mathrm{CTerm}}}{\{\xi\}\ \mathbf{while}\ b\ \mathbf{do}\ s\ \{\xi\}}\ [\text{IL-While}]$$

Fig. 9. IL proof rules (selected)

```
proc sum () =
  var c:int, x:int;
  c ← 0;
  for j ← 1 to N do
    x ←$ B(j,1/2);
    c ← c + x;
  return c
```

Fig. 10. Sum of bin.

The first premise follows from the rule for random assignment and structural rules. The second premise follows from the rule for deterministic assignment and the rule of consequence, applying axioms about sums of binomial distributions.

We briefly comment on several limitations of IL. First, IL is restricted to programs with deterministic control flow, but this restriction could be partially relaxed by enriching IL with assertions for conditional independence. Such assertions are already expressible in the logic of Ellora; adding conditional independence would significantly broaden the scope of the IL proof system and open the possibility to rely on axiomatizations of conditional independence (e.g., based on graphoids [36]). Second, the logic only supports sampling from binomial distributions. It is possible to enrich the language of assertions with clauses $c \sim g$ where g can model other distributions, like the uniform distribution or the Laplace distribution. The main design challenge is finding a core set of useful facts about these distributions. Enriching the logic and automating the analysis are interesting avenues for further work.

6.2 Embedding the Union Bound Logic

The program logic aHL [4] was recently introduced for estimating accuracy of randomized computations. One main application of aHL is proving accuracy of randomized algorithms, both in the offline and online settings—i.e. with adversary calls. aHL is based on the union bound, a basic tool from probability theory, and has judgments of the form $\models_\beta \{\Phi\}\ s\ \{\Psi\}$, where s is a statement, Φ and Ψ are first-order formulae over program variables, and β is a probability, i.e. $\beta \in [0,1]$. A judgment $\models_\beta \{\Phi\}\ s\ \{\Psi\}$ is valid if for every memory m such that $\Phi(m)$, the probability of $\neg\Psi$ in $[\![s]\!]_m$ is upper bounded by β, i.e. $\Pr_{[\![s]\!]_m}[\neg\Psi] \leq \beta$.

Figure 11 presents some key rules of aHL, including a rule for sampling from the Laplace distribution $\mathcal{L}_\epsilon$ centered around e. The predicate $\mathcal{C}_{\mathrm{CTerm}}(k)$ indicates that the loop terminates in at most k steps on any memory that satisfies the pre-condition. Moreover, β is a function of ϵ.

$$\frac{}{\models_\beta \{\top\}\ x \xleftarrow{\$} \mathcal{L}_\epsilon(e)\ \{|x-e| \leq \frac{1}{\epsilon}\log\frac{1}{\beta}\}}\ [\text{aHL-Sample}]$$

$$\frac{\models_{\beta_1} \{\Phi\}\ s_1\ \{\Theta\} \qquad \models_{\beta_2} \{\Theta\}\ s_2\ \{\Psi\}}{\models_{\beta_1+\beta_2} \{\Phi\}\ s_1; s_2\ \{\Psi\}}\ [\text{aHL-Seq}]$$

$$\frac{\models_\beta \{\Phi\}\ c\ \{\Phi\} \qquad \mathcal{C}_{\mathrm{CTerm}}(k)}{\models_{k\cdot\beta} \{\Phi\}\ \textbf{while}\ e\ \textbf{do}\ c\ \{\Phi \wedge \neg e\}}\ [\text{aHL-While}]$$

Fig. 11. aHL proof rules (selected)

aHL has a simple embedding into Ellora.

Theorem 3 (Embedding of aHL)**.** *If* $\models_\beta \{\Phi\}\ s\ \{\Psi\}$ *is derivable in* aHL*, then* $\{\Box\Phi\}\ s\ \{\mathbb{E}[\mathbf{1}_{\neg\Psi}] \leq \beta\}$ *is derivable in* Ellora.

7 Case Studies: Verifying Randomized Algorithms

In this section, we will demonstrate Ellora on a selection of examples; we present further examples in the supplemental material. Together, they exhibit a wide variety of different proof techniques and reasoning principles which are available in the Ellora's implementation.

Hypercube Routing. will begin with the *hypercube routing* algorithm [41,42]. Consider a network topology (the *hypercube*) where each node is labeled by a bitstring of length D and two nodes are connected by an edge if and only if the two corresponding labels differ in exactly one bit position.

In the network, there is initially one packet at each node, and each packet has a unique destination. The algorithm implements a routing strategy based on *bit fixing*: if the current position has bitstring i, and the target node has bitstring j, we compare the bits in i and j from left to right, moving along the edge that corrects the first differing bit. Valiant's algorithm uses randomization to guarantee that the total number of steps grows *logarithmically* in the number of packets. In the first phase, each packet i select an intermediate destination $\rho(i)$ uniformly at random, and use bit fixing to reach $\rho(i)$. In the second phase, each packet use bit fixing to go from $\rho(i)$ to the destination j. We will focus on the first phase since the reasoning for the second phase is nearly identical. We can model the strategy with the code in Figure 18, using some syntactic sugar for the **for** loops.[13] We assume that initially, the position of the packet i is at node

```
proc route (D T : int) :
  var ρ, pos, usedBy : node map;
  var nextE : edge;
  pos ← Map.init id 2^D; ρ ←Map.empty;
  for i ← 1 to 2^D do
    ρ[i] ←$ [1, 2^D]
    for t ← 1 to T do
      usedBy ← Map.empty;
      for i ← 1 to 2^D do
        if pos[i] ≠ ρ[i] then
          nextE ← getEdge pos[i] ρ[i];
          if usedBy[nextE] = ⊥ then
            // Mark edge used
            usedBy[nextE] ← i;
            // Move packet
            pos[i] ← dest nextE
  return (pos, ρ)
```

Fig. 12. Hypercube Routing

i (see `Map.init`). Then, we initialize the random intermediate destinations ρ. The remaining loop encodes the evaluation of the routing strategy iterated T time. The variable `usedBy` is a map that logs if an edge is already used by a packet, it is empty at the beginning of each iteration. For each packet, we try to move it across one edge along the path to its intermediate destination. The function

[13] Recall that the number of node in a hypercube of dimension D is 2^D so each node can be identified by a number in $[1, 2^D]$.

`getEdge` returns the next edge to follow, following the bit-fixing scheme. If the packet can progress (its edge is not used), then its current position is updated and the edge is marked as used.

We show that if the number of timesteps T is $4D + 1$, then all packets reach their intermediate destination in at most T steps, except with a small probability 2^{-2D} of failure. That is, the number of timesteps grows linearly in D, logarithmic in the number of packets. This is formalized in our system as:

$$\{T = 4D + 1\}\ \mathsf{route}\ \{\Pr[\exists i.\ \mathsf{pos}[i] \neq \rho[i]] \leq 2^{-2D}]\}$$

```
proc coupon (N : int) :
  var int cp[N], t[N];
  var int X ←0;
  for p ← 1 to N do
    ct ← 0;
    cur ←$ [1, N];
    while cp[cur] = 1 do
      ct ← ct + 1;
      cur ←$ [1, N];
    t[p]   ← ct;
    cp[cur] ← 1;
    X ← X + t[p];
  return X
```

Fig. 13. Coupon collector

Modeling Infinite Processes. Our second example is the *coupon collector* process. The algorithm draws a uniformly random coupon (we have N coupon) on each day, terminating when it has drawn at least one of each kind of coupon. The code of the algorithm is displayed in Fig. 13; the array `cp` records of the coupons seen so far, `t` holds the number of steps taken before seeing a new coupon, and X tracks of the total number of steps. Our goal is to bound the average number of iterations. This is formalized in our logic as:

$$\{\mathcal{L}\}\ \mathsf{coupon}\ \left\{\mathbb{E}[X] = \sum_{i\in[1,N]} \left(\frac{N}{N-i+1}\right)\right\}.$$

```
proc pwInd (N : int) :
  var bool X[2^N], B[N];
  for i ← 1 to N do
    B[i] ←$ Ber(1/2);
  for j ← 1 to 2^N do
    X[j] ← 0;
    for k ← 1 to N do
      if k ∈ bits(j) then
        X[j] ← X[j] ⊕ B[k]
  return X
```

Fig. 14. Pairwise Independence

Limited Randomness. Pairwise independence says that if we see the result of X_i, we do not gain information about all other variables X_k. However, if we see the result of *two* variables X_i, X_j, we may gain information about X_k. There are many constructions in the algorithms literature that grow a small number

of independent bits into more pairwise independent bits. Figure 14 gives one procedure, where $\oplus$ is exclusive-or, and `bits(j)` is the set of positions set to 1 in the binary expansion of `j`. The proof uses the following fact, which we fully verify: for a uniformly distributed Boolean random variable Y, and a random variable Z of any type,

$$Y \# Z \Rightarrow Y \oplus f(Z) \# g(Z) \tag{1}$$

for any two Boolean functions f, g. Then, note that $\mathtt{X}[i] = \bigoplus_{\{j \in \mathtt{bits}(i)\}} \mathtt{B}[j]$ where the big XOR operator ranges over the indices j where the bit representation of i has bit j set. For any two $i, k \in [1, \ldots, 2^{\mathtt{N}}]$ distinct, there is a bit position in $[1, \ldots, \mathtt{N}]$ where i and k differ; call this position r and suppose it is set in i but not in k. By rewriting,

$$\mathtt{X}[i] = \mathtt{B}[r] \oplus \bigoplus_{\{j \in \mathtt{bits}(i) \setminus r\}} \mathtt{B}[j] \quad \text{and} \quad \mathtt{X}[k] = \bigoplus_{\{j \in \mathtt{bits}(k) \setminus r\}} \mathtt{B}[j].$$

Since $\mathtt{B}[j]$ are all independent, $\mathtt{X}[i] \# \mathtt{X}[k]$ follows from Eq. (1) taking Z to be the distribution on tuples $\langle \mathtt{B}[1], \ldots, \mathtt{B}[\mathtt{N}] \rangle$ excluding $\mathtt{B}[r]$. This verifies pairwise independence:

$$\{\mathcal{L}\}\ \mathtt{pwInd(N)}\ \{\mathcal{L} \wedge \forall i, k \in [2^{\mathtt{N}}].\ i \neq k \Rightarrow \mathtt{X}[i] \# \mathtt{X}[k]\}.$$

Adversarial Programs. Pseudorandom functions (PRF) and pseudorandom permutations (PRP) are two idealized primitives that play a central role in the design of symmetric-key systems. Although the most natural assumption to make about a blockcipher is that it behaves as a pseudorandom permutation, most commonly the security of such a system is analyzed by replacing the blockcipher with a perfectly random function. The PRP/PRF Switching Lemma [22,6] fills the gap: given a bound for the security of a blockcipher as a pseudorandom function, it gives a bound for its security as a pseudorandom permutation.

Lemma 4 (PRP/PRF switching lemma)**.** *Let A be an adversary with blackbox access to an oracle O implementing either a random permutation on $\{0,1\}^l$ or a random function from $\{0,1\}^l$ to $\{0,1\}^l$. Then the probability that the adversary A distinguishes between the two oracles in at most q calls is bounded by*

$$|\Pr_{PRP}[b \wedge |H| \leq q] - \Pr_{PRF}[b \wedge |H| \leq q]| \leq \frac{q(q-1)}{2^{l+1}},$$

where H is a map storing each adversary call and $|H|$ is its size.

Proving this lemma can be done using the Fundamental Lemma of Game-Playing, and bounding the probability of *bad* in the program from Fig. 15. We focus on the latter. Here we apply the [Adv] rule of Ellora with the invariant $\forall k, \Pr[\mathsf{bad} \wedge |H| \leq k] \leq \frac{k(k-1)}{2^{l+1}}$ where $|H|$ is the size of the map H, i.e. the number of adversary call. Intuitively, the invariant says that at each call to the oracle the probability that `bad` has been set before and that the number of adversary call is less than k is bounded by a polynomial in k.

The invariant is d-closed and true before the adversary call, since at that point $\Pr[\mathsf{bad}] = 0$. Then we need to prove that the oracle preserves the invariant, which can be done easily using the precondition calculus ([PC] rule).

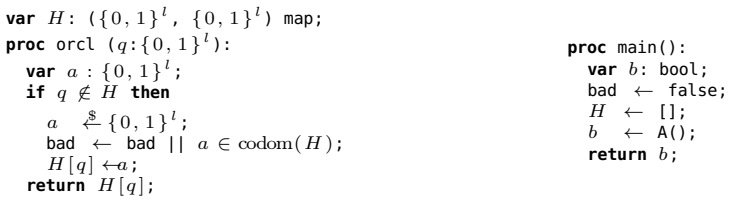


Fig. 15. PRP/PRF game

8 Implementation and Mechanization

We have built a prototype implementation of Ellora within EasyCrypt [5,2], a theorem prover originally designed for verifying cryptographic protocols. EasyCrypt provides a convenient environment for constructing proofs in various Hoare logics, supporting interactive, tactic-based proofs for manipulating assertions and allowing users to invoke external tools, like SMT-solvers, to discharge proof obligations. EasyCrypt provides a mature set of libraries for both data structures (sets, maps, lists, arrays, etc.) and mathematical theorems (algebra, real analysis, etc.), which we extended with theorems from probability theory.

Example	LC	FPLC
`hypercube`	100	1140
`coupon`	27	184
`vertex-cover`	30	61
`pairwise-indep`	30	231
`private-sums`	22	80
`poly-id-test`	22	32
`random-walk`	16	42
`dice-sampling`	10	64
`matrix-prod-test`	20	75

Table 1. Benchmarks

We used the implementation for verifying many examples from the literature, including all the programs presented in § 7 as well as some additional examples (such as polynomial identity test, private running sums, properties about random walks, etc.). The verified proofs bear a strong resemblance to the existing, paper proofs. Independently of this work, Ellora has been used to formalize the main theorem about a randomized gossip-based protocol for distributed systems [26, Theorem 2.1]. Some libraries developed in the scope of Ellora have been incorporated into the main branch of EasyCrypt, including a general library on probabilistic independence.

A New Library for Probabilistic Independence. In order to support assertions of the concrete program logic, we enhanced the standard libraries of EasyCrypt, notably the ones dealing with big operators and sub-distributions. Like all EasyCrypt libraries, they are written in a foundational style, i.e. they are defined instead of axiomatized. A large part of our libraries are proved formally from first principles. However, some results, such as concentration bounds, are currently declared as axioms.

Our formalization of probabilistic independence deserves special mention. We formalized two different (but logically equivalent) notions of independence. The first is in terms of products of probabilities, and is based on heterogenous lists.

Since Ellora (like EasyCrypt) has no support for heterogeneous lists, we use a smart encoding based on second-order predicates. The second definition is more abstract, in terms of product and marginal distributions. While the first definition is easier to use when reasoning about randomized algorithms, the second definition is more suited for proving mathematical facts. We prove the two definitions equivalent, and formalize a collection of related theorems.

Mechanized Meta-Theory. The proofs of soundness and relative completeness of the abstract logic, without adversary calls, and the syntactical termination arguments have been mechanized in the Coq proof assistant. The development is available in supplemental material.

9 Related Work

More on Assertion-Based Techniques. The earliest assertion-based system is due to Ramshaw [37], who proposes a program logic where assertions can be formulas involving *frequencies*, essentially probabilities on sub-distributions. Ramshaw's logic allows assertions to be combined with operators like $\oplus$, similar to our approach. [18] presents a Hoare-style logic with general assertions on the distribution, allowing expected values and probabilities. However, his **while** rule is based on a semantic condition on the guarded loop body, which is less desirable for verification because it requires reasoning about the semantics of programs. [8] give decidability results for a probabilistic Hoare logic without **while** loops. We are not aware of any existing system that supports assertions about general expected values; existing works also restrict to Boolean distributions. [38] formalize a Hoare logic for probabilistic programs but unlike our work, their assertions are interpreted on *distributions* rather than sub-distributions. For conditionals, their semantics rescales the distribution of states that enter each branch. However, their assertion language is limited and they impose strong restrictions on loops.

Other Approaches. Researchers have proposed many other approaches to verify probabilistic program. For instance, verification of Markov transition systems goes back to at least [17,40]; our condition for ensuring almost-sure termination in loops is directly inspired by their work. Automated methods include model checking (see e.g., [1,25,29]) and abstract interpretation (see e.g., [32,12]). Techniques for reasoning about higher-order (functional) probabilistic languages are an active subject of research (see e.g., [7,13,14]). For analyzing probabilistic loops, in particular, there are tools for reasoning about running time. There are also automated systems for synthesizing invariants [11,3]. [9,10] use a martingale method to compute the expected time of the coupon collector process for $N = 5$—fixing N lets them focus on a program where the outer **while** loop is fully unrolled. Martingales are also used by [15] for analyzing probabilistic termination. Finally, there are approaches involving symbolic execution; [39] use a mix of static and dynamic analysis to check probabilistic programs from the approximate computing literature.

10 Conclusion and Perspectives

We introduced an expressive program logic for probabilistic programs, and showed that assertion-based systems are suited for practical verification of probabilistic programs. Owing to their richer assertions, program logics are a more suitable foundation for specialized reasoning principles than expectation-based systems. As evidence, our program logic can be smoothly extended with custom reasoning for probabilistic independence and union bounds. Future work includes proving better accuracy bounds for differentially private algorithms, and exploring further integration of Ellora into EasyCrypt.

Acknowledgments. We thank the reviewers for their helpful comments. This work benefited from discussions with Dexter Kozen, Annabelle McIver, and Carroll Morgan. This work was partially supported by ERC Grant #679127, and NSF grants 1513694 and 1718220.

References

1. Baier, C.: Probabilistic model checking. In: Dependable Software Systems Engineering, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 45, pp. 1–23. IOS Press (2016), http://dx.doi.org/10.3233/978-1-61499-627-9-1
2. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.: Easycrypt: A tutorial. In: Foundations of Security Analysis and Design (FOSAD). Lecture Notes in Computer Science, vol. 8604, pp. 146–166. Springer-Verlag (2013)
3. Barthe, G., Espitau, T., Ferrer Fioriti, L.M., Hsu, J.: Synthesizing probabilistic invariants via Doob's decomposition. In: International Conference on Computer Aided Verification (CAV), Toronto, Ontario (2016), https://arxiv.org/abs/1605.02765
4. Barthe, G., Gaboardi, M., Grégoire, B., Hsu, J., Strub, P.Y.: A program logic for union bounds. In: International Colloquium on Automata, Languages and Programming (ICALP), Rome, Italy (2016), http://arxiv.org/abs/1602.05681
5. Barthe, G., Grégoire, B., Heraud, S., Béguelin, S.Z.: Computer-aided security proofs for the working cryptographer. In: IACR International Cryptology Conference (CRYPTO), Santa Barbara, California. Lecture Notes in Computer Science, vol. 6841, pp. 71–90. Springer-Verlag (2011)
6. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: IACR International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Saint Petersburg, Russia. pp. 409–426 (2006), http://dx.doi.org/10.1007/11761679_25
7. Bizjak, A., Birkedal, L.: Step-indexed logical relations for probability. In: International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), London, England. pp. 279–294. Springer-Verlag (2015), https://arxiv.org/abs/1501.02623
8. Chadha, R., Cruz-Filipe, L., Mateus, P., Sernadas, A.: Reasoning about probabilistic sequential programs. Theoretical Computer Science 379(1–2), 142–165 (2007)

9. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: International Conference on Computer Aided Verification (CAV), Saint Petersburg, Russia. Lecture Notes in Computer Science, vol. 8044, pp. 511–526. Springer-Verlag (2013)
10. Chakarov, A., Sankaranarayanan, S.: Expectation invariants as fixed points of probabilistic programs. In: International Symposium on Static Analysis (SAS), Perpignan, France. Lecture Notes in Computer Science, vol. 8723, pp. 85–100. Springer-Verlag (2014)
11. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Saint Petersburg, Florida. pp. 327–342 (2016), http://doi.acm.org/10.1145/2837614.2837639
12. Cousot, P., Monerau, M.: Probabilistic abstract interpretation. In: European Symposium on Programming (ESOP), Tallinn, Estonia. Lecture Notes in Computer Science, vol. 7211, pp. 169–193. Springer-Verlag (2012)
13. Crubillé, R., Dal Lago, U.: On probabilistic applicative bisimulation and call-by-value λ-calculi. In: European Symposium on Programming (ESOP), Grenoble, France. Lecture Notes in Computer Science, vol. 8410, pp. 209–228. Springer-Verlag (2014), https://arxiv.org/abs/1501.02623
14. Dal Lago, U., Sangiorgi, D., Alberti, M.: On coinductive equivalences for higher-order probabilistic functional programs. In: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Diego, California. pp. 297–308 (2014), https://arxiv.org/abs/1311.1722
15. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Mumbai, India. pp. 489–501 (2015)
16. Gretz, F., Katoen, J.P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Performance Evaluation 73, 110–132 (2014)
17. Hart, S., Sharir, M., Pnueli, A.: Termination of probabilistic concurrent programs. ACM Transactions on Programming Languages and Systems 5(3), 356–380 (1983)
18. den Hartog, J.: Probabilistic extensions of semantical models. Ph.D. thesis, Vrije Universiteit Amsterdam (2002)
19. Hurd, J.: Formal verification of probabilistic algorithms. Tech. Rep. UCAM-CL-TR-566, University of Cambridge, Computer Laboratory (2003)
20. Hurd, J.: Verification of the Miller-Rabin probabilistic primality test. Journal of Logic and Algebraic Programming 56(1–2), 3–21 (2003), http://dx.doi.org/10.1016/S1567-8326(02)00065-6
21. Hurd, J., McIver, A., Morgan, C.: Probabilistic guarded commands mechanized in HOL. Theoretical Computer Science 346(1), 96–112 (2005)
22. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: ACM SIGACT Symposium on Theory of Computing (STOC), Seattle, Washington. pp. 44–61 (1989), http://doi.acm.org/10.1145/73007.73012
23. Kaminski, B.L., Katoen, J.P., Matheja, C.: Inferring covariances for probabilistic programs. In: International Conference on Quantitative Evaluation of Systems (QEST). pp. 191–206. Springer-Verlag (2016)
24. Kaminski, B., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In: European Symposium on Programming (ESOP), Eindhoven, The Netherlands (Jan 2016)

25. Katoen, J.P.: The probabilistic model-checking landscape. In: IEEE Symposium on Logic in Computer Science (LICS), New York, New York (2016)
26. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. pp. 482–491 (2003), http://dx.doi.org/10.1109/SFCS.2003.1238221
27. Kozen, D.: Semantics of probabilistic programs. vol. 22, pp. 328–350. Elsevier (1981), https://www.sciencedirect.com/science/article/pii/0022000081900362
28. Kozen, D.: A probabilistic PDL. Journal of Computer and System Sciences 30(2), 162–178 (1985)
29. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: International Conference on Computer Aided Verification (CAV), Snowbird, Utah. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer-Verlag (2011)
30. McIver, A., Morgan, C.: Abstraction, Refinement, and Proof for Probabilistic Systems. Monographs in Computer Science, Springer-Verlag (2005)
31. McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.P.: A new rule for almost-certain termination. Proceedings of the ACM on Programming Languages 1(POPL) (2018), https://arxiv.org/abs/1612.01091, appeared at ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Los Angeles, California
32. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: International Symposium on Static Analysis (SAS), Santa Barbara, California. Lecture Notes in Computer Science, vol. 1824, pp. 322–339. Springer-Verlag (2000)
33. Morgan, C.: Proof rules for probabilistic loops. In: BCS–FACS Conference on Refinement, Bath, England (1996)
34. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems 18(3), 325–353 (1996)
35. Olmedo, F., Kaminski, B.L., Katoen, J.P., Matheja, C.: Reasoning about recursive probabilistic programs. In: IEEE Symposium on Logic in Computer Science (LICS), New York, New York. pp. 672–681 (2016)
36. Pearl, J., Paz, A.: Graphoids: Graph-based logic for reasoning about relevance relations. In: ECAI. pp. 357–363 (1986)
37. Ramshaw, L.H.: Formalizing the Analysis of Algorithms. Ph.D. thesis, Computer Science (1979)
38. Rand, R., Zdancewic, S.: VPHL: a verified partial-correctness logic for probabilistic programs. In: Conference on the Mathematical Foundations of Programming Semantics (MFPS), Nijmegen, The Netherlands (2015)
39. Sampson, A., Panchekha, P., Mytkowicz, T., McKinley, K.S., Grossman, D., Ceze, L.: Expressing and verifying probabilistic assertions. In: ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Edinburgh, Scotland. p. 14 (2014)
40. Sharir, M., Pnueli, A., Hart, S.: Verification of probabilistic programs. SIAM Journal on Computing 13(2), 292–314 (1984)
41. Valiant, L.G.: A scheme for fast parallel communication. SIAM Journal on Computing 11(2), 350–361 (1982)
42. Valiant, L.G., Brebner, G.J.: Universal schemes for parallel communication. In: ACM SIGACT Symposium on Theory of Computing (STOC), Milwaukee, Wisconsin. pp. 263–277 (1981), http://doi.acm.org/10.1145/800076.802479

A Soundness of Ellora

Before presenting the proof of soundness, we will introduce two technical lemmas needed for the loop rules. Intuitively, d- and t-closed assertions are preserved in the limit of general and lossless loops, respectively.

Proposition 5.

1. *If η is d-closed and is s.t.*

$$\forall \mu.\, \mu \models \eta \implies [\![\mathbf{if}\ b\ \mathbf{then}\ s]\!]_\mu \models \eta,$$

   then $\forall \mu.\, \mu \models \eta \implies [\![\mathbf{while}\ b\ \mathbf{do}\ s]\!]_\mu \models \eta$.
2. *If η is t-closed and is s.t.*

$$\forall \mu.\, \mu \models \eta \implies [\![\mathbf{if}\ b\ \mathbf{then}\ s]\!]_\mu \models \eta,$$

   then $\forall \mu.\, \mu \models \eta \implies [\![\mathbf{while}\ b\ \mathbf{do}\ s]\!]_\mu \models \eta$, *provided that* $\mathbf{while}\ b\ \mathbf{do}\ s$ *is lossless.*

Proof. We only treat the first case; the second is similar. Let η be a d-closed assertion s.t. for any sub-distribution μ, if $\eta(\mu)$, then $[\![\mathbf{if}\ b\ \mathbf{then}\ s]\!]_\mu \models \eta$. We prove by induction on n that for any sub-distribution μ such that $\mu \models \eta$, we have $[\![(\mathbf{if} b \mathbf{then} s)^n]\!]_\mu \models \eta$. By downward closedness of η, we have $[\![(\mathbf{if} b \mathbf{then} s)^n_{|\neg b}]\!]_\mu \models \eta$. We conclude by t-closenedness of η. □

Lemma 6. *Let η be an assertion and s be a command. Then $\{\eta[[\![s]\!]]\}\ s\ \{\eta\}$.*

Proof. Let $\mu \models \eta[[\![s]\!]]$. By definition of $\eta[[\![s]\!]]$, this amounts to have $[\![s]\!]_\mu \models \eta$, which exactly gives the expected result. □

Lemma 7. *Let μ be a sub-distribution and s be a command. Then, $|[\![s]\!]_\mu| \leq |\mu|$.*

Proof. We have

$$|[\![s]\!]_\mu| = |\mathbb{E}_{m\sim\mu}[[\![s]\!]_m]| = \sum_m \mu(m) \cdot \overbrace{|[\![s]\!]_m|}^{\leq 1}$$
$$\leq \sum_m \mu(m) = |\mu| \square$$

Lemma 8. *Let μ be a sub-distribution s.t. $\mu \triangleq \sum_{i\in I} \lambda_i \mu_i$ where I is a finite set, all the λ_i's are in $[0,1]$ and all the μ_i's are sub-distributions. Then, for any command s,*

$$[\![s]\!]_\mu = \sum_{i\in I} \lambda_i \cdot [\![s]\!]_{\mu_i}$$

Proof. Immediate consequence of the linearity of $\mathbb{E}$. □

Lemma 9. *Let s be a lossless command and μ be a sub-distribution. Then* $\mu_{|\overline{\mathrm{mod}(s)}} = [\![s]\!]_\mu{}_{|\overline{\mathrm{mod}(s)}}$.

Proof. The proof is done by a direct induction of s. □

Lemma 10. *The rules of* Ellora *are sound.*

Proof. We use the notation of the rules.

- [Skip] — immediate since $[\![\mathbf{skip}]\!]_\mu = \mu$.
- [Abort] — immediate by definition of $\Box\eta$ and since

$$\mathrm{supp}([\![\mathbf{abort}]\!]_\mu) = \mathrm{supp}(\not\vdash) = \emptyset.$$

- [Assgn] & [Sample] — immediate consequence of Lemma 6.
- [Seq] — let $\mu \models \eta_1$. Then, by the first premise, $[\![s_1]\!]_\mu \models \eta_2$. Hence, by the second premise, $[\![s_2]\!]_{[\![s_1]\!]_\mu} \models \eta_3$.
- [Conseq] — if $\mu \models \eta_0$, then $\mu \models \eta_1$ by the first premise. Hence, $[\![s]\!]_\mu \models \eta_2$ by the second premise and $[\![s]\!]_\mu \models \eta_3$ by the third premise.
- [Split] — let $\mu \models \eta_1 \oplus \eta_2$. Then, there exists μ_1, μ_2 s.t. $\mu = \mu_1 + \mu_2$ and $\mu_i \models \eta_i$ for $i \in \{1, 2\}$. By the two premises of the rule, we have $[\![s]\!]_{\mu_i} \models \eta'_i$ for $i \in \{1, 2\}$. Now, by Lemma 8, we have $[\![s]\!]_\mu = [\![s]\!]_{\mu_1} + [\![s]\!]_{\mu_2}$. By taking resp. $[\![s]\!]_{\mu_1}$ and $[\![s]\!]_{\mu_2}$ for the witnesses of $\eta'_1 \oplus \eta'_2$, we obtain that $[\![s]\!]_\mu \models \eta'_1 \oplus \eta'_2$.
- [Cond] — we first prove that

$$\{\eta_1 \wedge \Box e\}\ \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \{\eta'_1\}. \tag{2}$$

  Let $\mu \models \eta_1 \wedge \Box e$. Then, for $m \in \mathrm{supp}(\mu)$, we know that $[\![e]\!]_m = \top$. It follows that

$$\begin{aligned}[\![\mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2]\!]_\mu &= \mathbb{E}_{m\sim\mu}[\underbrace{[\![\mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2]\!]_m}_{=[\![s_1]\!]_m}] \\ &= [\![s_1]\!]_\mu.\end{aligned}$$

  However, by the first premise, $[\![s_1]\!]_\mu \models \eta'_1$. Hence,

$$[\![\mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2]\!]_\mu \models \eta'_1,$$

  concluding the proof of Eq. (2). By a similar reasoning, we have:

$$\{\eta_2 \wedge \Box\neg e\}\ \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \{\eta'_2\}. \tag{3}$$

  We conclude the proof of soundness of [Cond] from the one of [Split] applied to Eq. (2) and Eq. (3).
- [Call] — immediate consequence of [Seq] and [Assgn].
- [Frame] — let $\mu \models \eta$. Let $\mu \models \eta$. To show that $[\![s]\!]_\mu \models \eta$, from the definition of $\mathsf{separated}(\eta, \mathrm{mod}(s))$, it is sufficient to show that $|\mu| = |[\![s]\!]_\mu|$ and $\mu_{|\overline{\mathrm{mod}(s)}} = [\![s]\!]_\mu{}_{|\overline{\mathrm{mod}(s)}}$. The first one is a consequence of the losslessness of s, the second one is a direct application of Lemma 9.

– [WHILE] — from the first premise, by induction on n and using [SEQ], we know that for any n, the following holds:

$$\{\eta\}\ (\textbf{if}\ b\ \textbf{then}\ s)^n\ \{\eta\}. \tag{4}$$

Now, let $\mu \models \eta$. First, form the definition of $[\![\textbf{while}\ b\ \textbf{do}\ s]\!]_\mu$, we know that

$$[\![\textbf{while}\ b\ \textbf{do}\ s]\!]_\mu \models \Box\neg b.$$

It remains to prove $[\![\textbf{while}\ b\ \textbf{do}\ s]\!]_\mu \models \eta$. In the case of certain termination, we know the existence of a k s.t.

$$[\![\textbf{while}\ b\ \textbf{do}\ s]\!] = [\![(\textbf{if}\ b\ \textbf{then}\ s\ \textbf{else}\ )^k]\!]$$

and we conclude by Eq. (4). For the cases of almost surely termination and almost termination, we conclude for Eq. (4) and Proposition 5 of the paper.

– [ADV] — the proof is done by induction on the body of the external procedure which is of the form $s_1; c_1^?; \cdots; s_n; c_n^?$ where the s_i's do not contain calls and the $c_i^?$'s are potential calls to the oracles. From the [FRAME] rule, we know that the s_i's preserve the invariant η — noting that the losslessness of the adversary implies the losslessness of the s_i's. Likewise, from the last premise of the [ADV] rule, we know that calls to the oracles also preserve the invariant. Hence, by multiple application of the [SEQ] rule, we obtain that the adversary body maintains η. □

B Semantics of Assertions

The semantics of assertions is given Fig. 16.

C Precondition Calculus

Figure 17 contains the full definition of the precondition calculus.

D Soundness of the Syntactic Rules

D.1 Certain Termination

Proposition 11 (Soundness of rule [WHILE-$\mathcal{C}_{\text{CTERM}}$])**.** *Let μ be a sub-distributions such that $\mathcal{C}_{CTerm}$ is valid. Then,*

$$[\![\textbf{while}\ b\ \textbf{do}\ s]\!]\mu \models \eta \wedge \Box\neg b.$$

Proof. Given μ satisfying $\mathcal{C}_{\text{CTerm}}$, we first claim that there exists a decreasing function $f : \mathbb{N} \to \mathbb{N}$ such that $\Box(\tilde{e} \leq f(n) \vee \neg b)$ holds at each iteration of the

$$
\begin{array}{ll}
[\![v]\!]^{\rho}_{m} & \triangleq \rho(v) \\
[\![\mathbf{1}_{\phi}]\!]^{\rho}_{m} & \triangleq \mathbf{1}_{[\![\phi]\!]^{\rho}_{m}} \\
[\![\mathbb{E}_{v\sim g}[\tilde{e}]]\!]^{\rho}_{m} & \triangleq \mathbb{E}_{w\sim[\![g]\!]^{\rho}_{m}}[[\![\tilde{e}]\!]^{\rho[v:=w]}_{m}] \\
[\![o(\tilde{e})]\!]^{\rho}_{m} & \triangleq o([\![\tilde{e}]\!]^{\rho}_{m})
\end{array}
$$

$$
\begin{array}{ll}
[\![\tilde{e}_1 \bowtie \tilde{e}_2]\!]^{\rho}_{m} & \triangleq [\![\tilde{e}_1]\!]^{\rho}_{m} \bowtie [\![\tilde{e}_2]\!]^{\rho}_{m} \\
[\![FO(\phi)]\!]^{\rho}_{m} & \triangleq FO([\![\phi]\!]^{\rho}_{m})
\end{array}
$$

$$
\begin{array}{ll}
[\![\mathbb{E}[\tilde{e}]]\!]^{\rho}_{\mu} & \triangleq \mathbb{E}_{m\sim\mu}[[\![\tilde{e}]\!]^{\rho}_{m}] \\
[\![o(p)]\!]^{\rho}_{\mu} & \triangleq o([\![p]\!]^{\rho}_{\mu})
\end{array}
$$

$$
\begin{array}{ll}
[\![p_1 \bowtie p_2]\!]^{\rho}_{\mu} & \triangleq [\![p_1]\!]^{\rho}_{\mu} \bowtie [\![p_2]\!]^{\rho}_{\mu} \\
[\![\eta_1 \oplus \eta_2]\!]^{\rho}_{\mu} & \triangleq \exists \mu_1, \mu_2.\, \mu = \mu_1 + \mu_2 \wedge [\![\eta_1]\!]^{\rho}_{\mu_1} \wedge [\![\eta_2]\!]^{\rho}_{\mu_2} \\
[\![FO(\eta)]\!]^{\rho}_{\mu} & \triangleq FO([\![\eta]\!]^{\rho}_{\mu})
\end{array}
$$

Fig. 16. Semantics of assertions

$$
\begin{array}{ll}
\mathrm{pe}(s, o(p)) & \triangleq o(\mathrm{pe}(s,p)) \quad \text{where } o \in \mathbf{Ops} \\
\mathrm{pe}(\mathbf{skip}, \mathbb{E}[\tilde{e}]) & \triangleq \mathbb{E}[\tilde{e}] \\
\mathrm{pe}(s_1; s_2, \mathbb{E}[\tilde{e}]) & \triangleq \mathrm{pe}(s_1, \mathrm{pe}(s_2, \mathbb{E}[\tilde{e}])) \\
\mathrm{pe}(x \leftarrow e, \mathbb{E}[\tilde{e}]) & \triangleq \mathbb{E}[\tilde{e}][x := e] \\
\mathrm{pe}(x \xleftarrow{\$} g, \mathbb{E}[\tilde{e}]) & \triangleq \mathcal{P}^{g}_{x}(\mathbb{E}[\tilde{e}]) \\
\mathrm{pe}(\mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2, \mathbb{E}[\tilde{e}]) & \triangleq \mathrm{pe}(s_1, \mathbb{E}[\tilde{e}])_{|e} + \mathrm{pe}(s_2, \mathbb{E}[\tilde{e}])_{|\neg e} \\
\mathrm{pe}(\mathbf{abort}, \mathbb{E}[\tilde{e}]) & \triangleq 0 \\
\\
\mathrm{pc}(s, p_1 \bowtie p_2) & \triangleq \mathrm{pe}(s, p_1) \bowtie \mathrm{pe}(s, p_2) \\
\mathrm{pc}(s, FO(\eta)) & \triangleq FO(\mathrm{pc}(s, \eta))
\end{array}
$$

Fig. 17. Precondition calculus

loop. Indeed, we remark that the statement $\exists k\, \Box(\tilde{e} \leq k)$ holds at first iteration by the precondition hypothesis. Let:

$$f(n) = \begin{cases} k - n \text{ if } n \leq k \\ 0 \qquad \text{otherwise.} \end{cases}$$

Then unrolling the loop and using the semantical [Seq] rule ensure by induction the claimed domination. The termination of the loop arises then naturally from the exit condition $\Box(\tilde{e} = 0 \Rightarrow \neg b)$. □

D.2 Almost-Sure Termination

The main challenge for proving the soundness of the is proving termination; from there, we can conclude by rule [While-$\mathcal{C}_{\text{AST}_{\text{ERM}}}$]. Our arguments use basic notations and theorems from the theory of Markov processes.

Proposition 12 (Soundness of rule [While-$\mathcal{C}_{\text{AST}_{\text{ERM}}}$])**.** *Let η be a t-closed assertion. For any sub-distributions μ, such that $\mathcal{C}_{ASTerm}$ is valid. Then*

$$[\![\textbf{while } b \textbf{ do } s]\!]\mu \models \eta.$$

Proof. As indicated, by soundness of the semantic while rule for almost-sure termination, and the premises, it suffices to prove almost-sure termination. The sketch of the proof is the following:

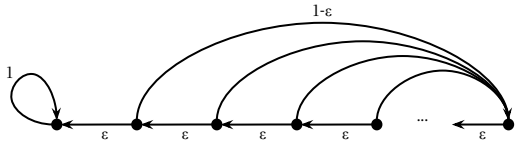
1. We follow the behavior of the variant by seeing it as a random variable on the space of state.
2. We first introduce a Markov chain that reaches a particular state (the state zero) with probability 1.
3. We then stochasticaly dominates the variant by the latter chain.
4. In particular, this shows that the probability of the variant to eventually reach 0 is 1, ensuring almost-sure termination.

Step one: Variant as a Random variable.
We consider the integer-valued variant $\tilde{e}$ as a random variable over the space of states. Let the family $(\tilde{e}_i)_i$ represents the value taken by $\tilde{e}$ after the i-th iteration of the loop. Then we have, using the semantical [Seq] rule and the

- The sequence is uniformly bounded by K.
- The probability of decreasing is bounded below by ϵ:

$$\forall i \in \mathbb{N}.\ \Pr[\tilde{e}_i > \tilde{e}_{i+1} \mid \tilde{e}_i, \ldots, \tilde{e}_0] = \epsilon_i \geq \epsilon > 0.$$

Step two: Modelization with a Markov chain.
First, we can assume that $K > 0$ since if $K = 0$, the loop terminates immediately. Consider the following finite Markov chain $(X_i)_i$, over the state space $S = \{0, \ldots, K\}$, following the transitions:



This chain models the following behavior: with probability ϵ the value decrease by one, while with probability $1-\epsilon$ it jumps to K. Since zero is the only connected component of the underlying graph, the probability of terminating in the state zero is 1 by a standard result on Markov chains.

Step three: Stochastic domination.
By construction, there exists a natural coupling between $(\tilde{X}_i)$ and $(\tilde{e}_i)_i$, since the probability of the event $\tilde{e}_i$ *decrease* is greater than the probability of the event $\tilde{X}_i$ decrease. Since we impose that $X_0 = \tilde{e}_0$ (initial position), a simple induction over i ensure that we have, for this coupling: $\Pr[\tilde{e}_i > 0] \leq \Pr[X_i > 0]$, so $\lim_{i\to\infty} \Pr[\tilde{e}_i = 0] \geq 1 - \lim_{i\to\infty} \Pr[X_i > 0] = 1$ as desired.

□

E Further Examples

E.1 Randomization for Approximation: Vertex Cover

We begin with a classical application of randomization: *approximation algorithms* for computationally hard problems. For problems that take long time to solve in the worst case, we can sometimes devise efficient algorithms that find a solution that is "nearly" as good as the true solution.

Our first example illustrates a famous approximation algorithm for the *vertex cover* problem. The input is a graph described by vertices V and edges E. The goal is to output a vertex cover: a subset $C \subseteq V$ such that each edge has at least one endpoint in C, and such that C is as small as possible.

It is known that this problem is NP-complete, but there is simple randomized algorithm that returns a vertex cover that is at on average at most twice the size of the optimal vertex cover. The algorithm proceeds by maintaining a current cover (initially empty) and considering each edge in order. If neither endpoint is in current cover, the algorithm adds one of the two endpoints uniformly at random. The Ellora program is:

```
proc VC (E : set<edge>) :
var   set<node> C ← ∅;
for (e1,e2) in E do
  if (e1 ∉ C) ∧ (e2 ∉ C) then
    b ←$ {0,1};
    C ← (b ? e1 : e2) ∪ C;
  fi
end
```

Here, we represent edges as a finite set of pairs of nodes. We loop through the edges, adding one point of each uncovered edge to the cover $\mathtt{C}$ uniformly at random. The operator $\mathtt{b}\ ?\ \mathtt{e}_1 : \mathtt{e}_2$ returns $\mathtt{e}_1$ if $\mathtt{b}$ is true, and $\mathtt{e}_2$ if not.

To prove the approximation guarantee, we first assume that we have a set of nodes $\mathtt{C}^*$. We only assume that $\mathtt{C}^*$ is a valid vertex cover; i.e., each edge has at least one endpoint in $\mathtt{C}^*$. Then, we use the following loop invariant:

$$\mathbb{E}[\mathtt{size}(\mathtt{C} \setminus \mathtt{C}^*)] \leq \mathbb{E}[\mathtt{size}(\mathtt{C} \cap \mathtt{C}^*)]. \tag{5}$$

Given the loop invariant, we can prove the conclusion by letting $\mathtt{C}^*$ be the cover OPT of minimal size, and reasoning about intersections and differences of sets.

Clearly the invariant is initially true. To see why the invariant is preserved, let $\mathtt{e}$ be the current edge, with both endpoints out of $\mathtt{C}$. Since $\mathtt{C}^*$ is a vertex cover, it has at least one endpoint of $\mathtt{e}$. Since our algorithm includes an endpoint of $\mathtt{e}$ uniformly at random, the probability we choose a vertex not in $\mathtt{C}^*$ is at most $1/2$, so the expectation on the left in Eq. (5) increases by at most $1/2$. If $\mathtt{e}$ is not covered in $\mathtt{C}$ but is covered by $\mathtt{C}^*$, there is at least a $1/2$ probability that we increase the intersection $\mathtt{C} \cap \mathtt{C}^*$, so the right side in Eq. (5) increases by at least $1/2$. Thus, the invariant is preserved, and we can prove

$$\{\mathrm{isVC}(\mathtt{C}^*, \mathtt{E})\}\ \mathtt{VC(E)}\ \{\mathbb{E}[\mathtt{size}(\mathtt{C} \setminus \mathtt{C}^*)] \leq \mathbb{E}[\mathtt{size}(\mathtt{C} \cap \mathtt{C}^*)]\}$$

and by reasoning on intersection and difference of sets,

$$\{\mathrm{isVC}(\mathtt{C}^*, \mathtt{E})\}\ \mathtt{VC(E)}\ \{\mathbb{E}[\mathtt{size}(\mathtt{C})] \leq 2 \cdot \mathbb{E}[\mathtt{size}(\mathtt{C}^*)]\}.$$

E.2 Random Walks: Termination and Reachability

A canonical example of a random process is a *random walk*. There are many variations, but the basic scenario describes a numeric position changing over time, where the position depends on the position at the previous timestep, influenced by random noise drawn from some distribution.

To demonstrate how to verify interesting facts about random processes, we will model a one-dimensional random walk on the natural numbers. We start at position 0, and repeatedly update our position according to the following rules. From 0, we always make a step to 1. From non-zero positions, we flip a fair coin that is biased to come up heads with probability $p > 0$. If heads, we increase our position by 1; if tails, we decrease by 1.

We will prove two facts about this random walk. First, for any natural number T, the probability of eventually reaching T is 1. Second, when we reach T, we must first pass through all intermediate points $1, \ldots, T-1$. In Ellora, we can express the random walk with the following code.

```
var bool visited[T];
var int  pos = 0;
visited[0] ← true;
while pos ≠ T do:
  c  ←$ Ber(p);
```

```
  pos ← pos + ((pos = 0) ∨ c) ? 1 : -1;
  visited[pos] ← true;
end
```

In order to verify this example, we will use the probabilistic while rule [While-AST$_{\text{ERM}}$]. First we establish almost-sure termination by finding an appropriate termination measure: the distance between our current position, and `T`. Indeed, this measure is bounded by `T`, and has a non-negative (at least 1/2) probability of decreasing each iteration. Therefore, the loop terminates almost-surely, and thus our random walk eventually reaches any point `T` with probability 1.

To prove our second assertion—that we visit every point from 0 to `T`—we use the following loop invariant for the while command:

$$\square(\forall i.\ 0 \leq i \leq \texttt{pos} \Rightarrow \texttt{visited[i]} = \texttt{true}).$$

In other words, if we have reached position `pos`, then we must have already reached every position in $[0, \texttt{pos}]$. Since this invariant is t-closed, we may apply rule [While-AST$_{\text{ERM}}$] and the invariant holds at the end of the loop. With the assertion $\texttt{pos} = \texttt{T}$ at termination, we have enough to prove that each position is visited:

$$\{\mathcal{L}\}\ s\ \{\mathcal{L} \wedge \square(\forall i \in [\texttt{T}].\ \texttt{visited[i]} = \texttt{true})\}.$$

The losslessness post-condition indicates that the walk terminates almost-surely.

E.3 Amplification: Polynomial Identity Testing

A second use of randomness is in running independent trials of the same algorithm. This technique, known as *probability amplification*, runs a randomized algorithm several times in order to reduce the error probability. Roughly speaking, a single trial may have high error with some probability, but by repeating the trial it is unlikely that all of the trials yield poor results. By combining the results appropriately—e.g., with a majority vote for algorithms with binary outputs, or by selecting the best answer when we can check the quality of the solution—we can produce an output that is more accurate than a single run of the original algorithm.

An example in this vein is probabilistic *polynomial identity testing*. Given two multivariate polynomials $P(x_1, \ldots, x_n)$ and $Q(x_1, \ldots, x_n)$ over the finite field $\mathbb{F}_q$ of q elements, we want to check whether $P = Q$, or equivalently, whether the polynomial $P - Q$ is zero or not. We will take n uniformly random samples $(v_i)_i$ from $\mathbb{F}_q$ and check whether $(P - Q)(v_1, \ldots, v_n) = 0$. We repeat the trial q times, rejecting if we see a sample where the difference is non-zero. In our system, this corresponds to the following program:

```
var bool res = true;
for i = 1 to q do:
  v  ←$ Unif(F_q^n);
  res ← res ∧ ((P - Q)(v) = 0);
end
```

The proof uses an instance of the Schwartz-Zippel lemma due to Øystein Ore for finite fields, which upper bounds the probability of randomly picking a root of a polynomial over a finite field.

Lemma 13. *Let P a non-zero polynomial function over $\mathbb{F}_q$. If we sample the variables $v_1, \ldots, v_n$ uniformly at random from $\mathbb{F}_q$, then*

$$\Pr[P(v_1, \ldots, v_n) = 0] \leq 1 - 1/q.$$

We encode this lemma as an axiom in our system:

$$P \neq 0 \wedge v \sim \mathbf{Unif}(\mathbb{F}_q^n) \Rightarrow \Pr[P(v) = 0] \leq 1 - 1/q.$$

With this fact, we can prove the following loop invariant:

$$\mathtt{P} \neq \mathtt{Q} \Rightarrow \Pr[\mathtt{res} = \mathtt{true}] \leq (1 - 1/\mathtt{q})^i,$$

finally proving that

$$\{\mathtt{P} \neq \mathtt{Q}\}\ s\ \{\Pr[\mathtt{res} = \mathtt{true}] \leq (1 - 1/\mathtt{q})^{\mathtt{q}} \leq 1/e\}.$$

We have also verified Freivald's algorithm, an amplification-based algorithm for checking matrix multiplication.

E.4 Concentration Bounds: Private Running Sums

Now, we turn to examples involving independence of random variables. Our first example is drawn from the *differential privacy* literature. Given a list of N integers, we add noise from a two-sided geometric distribution to each entry in order to protect privacy, and we calculate the *partial sums* of the noisy values: $x_1, x_1 + x_2, x_1 + x_2 + x_3$, and so on. We wish to measure how far the noisy sums deviate from the true sums.

In Ellora, we can express this algorithm as follows:

```
var int X[N], noise[N], out[N];
var int acc = 0;
for i = 1 to N do
  noise[i] ←$ twogeom(ϵ);
  out[i]   ←  acc + X[i] + noise[i];
  acc      ←  out[i];
end
```

The parameter ϵ to the noise distribution `twogeom` is a numeric parameter controlling the strength of the privacy guarantee, by changing the magnitude of the noise.

Our loop invariant tracks three pieces of information: (i) `noise[i]` is distributed according to `twogeom`(ϵ); (ii) the array `noise` remains independent; and (iii) `out[i]` stores the noisy running sum:

$$\forall q \in [\mathtt{i}].\ \mathtt{out[q]} - \sum_{p \in [q]} \mathtt{X}[p] = \sum_{p \in [q]} \mathtt{noise}[p].$$

To bound the error introduced by the noise, we need to bound $\left|\sum_{p\in[q]} \mathtt{noise}[p]\right|$. Since we know that the elements in noise are all independent, we can apply a concentration bound to bound the probability of a large error in the p-th running sum, concluding:

$$\{\mathcal{L}\}\ s\ \left\{\forall p \in [\mathtt{N}].\ \Pr\left[\left|\sum_{i\in[p]} \mathtt{X}_i\right| \geq T\right] \leq Q(\epsilon, T)/\sqrt{\mathtt{N}}\right\}$$

for a particular function Q derived from the Berry-Esseen theorem.

E.5 Hypercube Routing in More Details

We will begin with the *hypercube routing* algorithm [41,42]. To set the stage, consider a network where each node is labeled by a bitstring of length D, and two nodes are connected by an edge if and only if the two corresponding labels differ in exactly one bit position. This network topology is known as a *hypercube*, a D-dimensional version of the standard cube; a simple example with $D = 3$ is in Fig. 18. In the network, there is initially one packet at each node, and each

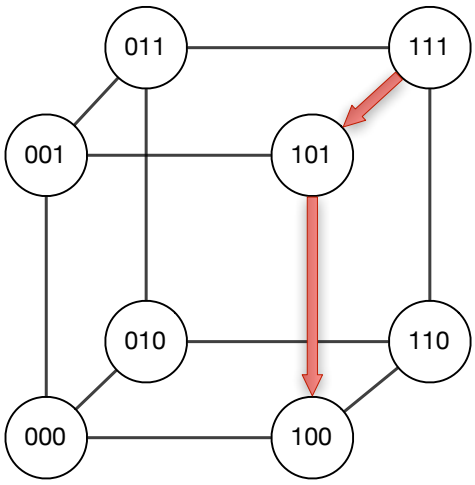


Fig. 18. Hypercube path from 111 to 100 ($D = 3$)

packet has a unique destination. Our goal is to design a routing strategy that will move the packets from node to node, following the edges, until all packets reach their destination. Furthermore, the routing should be *oblivious*: to avoid coordination overhead, each packet must select a path without considering the behavior of the other packets.

To model the flow of packets, each packet's current position is a node in the graph. Time proceeds in a series of steps, and at each step at most one packet can traverse any single edge. If several packets try to use the same edge

simultaneously, one packet will be selected to move (for any selection strategy that selects *some* packet). The other packets wait at their current position; these packets are *delayed* and make no progress this step.

The routing strategy is based on *bit fixing*: if the current position has bitstring i, and the target node has bitstring j, we compare the bits in i and j from left to right, moving along the edge that corrects the first differing bit. For instance, if we are at 111 and we wish to reach 100, we will move along the edge corresponding to the second position: from 111 to 101, and then along the edge corresponding to the third position: from 101 to 100. See Fig. 18 for a picture.

While this strategy is simple and oblivious, there are permutations π that require a total number of steps growing linearly in the number of packets to route all packets. Valiant proposes a simple modification, so that the total number of steps grows *logarithmically* in the number of packets. In the first phase, each packet i will first select an intermediate destination $\rho(i)$ uniformly at random from all nodes, and use bit fixing to reach $\rho(i)$. In the second phase, each packet will use bit fixing to go from $\rho(i)$ to the destination $\pi(i)$. We will focus on the first phase, since the reasoning for the second phase is nearly identical. We can model the strategy with the following code, using some syntactic sugar for the **for** loops; recall that the number of node in a hypercube of dimension D is 2^D so each node can be identified by a number in $[1, 2^D]$.

```
proc route (D T : int) :
  var ρ, pos, usedBy : node map;
  var nextE : edge;
  pos ← Map.init id 2^D; ρ ←Map.empty;
  for i ← 1 to 2^D do ρ[i] ←$ [1, 2^D]
  for t ← 1 to T do
    usedBy ← Map.empty;
    for i ← 1 to 2^D do
      if pos[i] ≠ ρ[i] then
        nextE ← getEdge pos[i] ρ[i];
        if usedBy[nextE] = ⊥ then
          usedBy[nextE] ← i; // Mark edge used
          pos[i] ← dest nextE // Move packet
  return (pos, ρ)
```

We assume that initially the position of the packet i is at node i (see `Map.init`). Then, we initialize the random intermediate destinations ρ. The remaining loop encodes the evaluation of the routing strategy iterated T time. The variable `usedBy` is a map that logs if an edge is already used by a packet, it is empty at the beginning of each iteration. For each packet, we try to move it across one edge along the path to its intermediate destination. The function `getEdge` returns the next edge to follow, following the bit-fixing scheme. If the packet can progress (its edge is not used), then its current position is updated and the edge is marked as used.

Our goal is to show that if the number of timesteps T is $4D + 1$, then all packets reach their intermediate destination in at most T steps, except with a small probability 2^{-2D} of failure. That is, the number of timesteps grows linearly in D, logarithmic in the number of packets. At a high level, the analysis involves three steps.

The first step is to consider a single packet traveling from i to $\rho(i)$, and to calculate the average number of other packets that share at least one edge with i's path P. Let $H^\rho(i,j)$ be 1 if the paths of packets i and j share at least one edge and 0 otherwise. Then, the load $R^\rho(i)$ on i's path is the sum of $H^\rho(i,j)$ over all the other packets j. By using the fact that each intermediate destination in ρ is uniformly distributed, and by analyzing the number of packets beside i's that use each edge on i's path, we can upper bound the expected value of $R^\rho(i)$ by $D/2$. In the second step, we move from a bound on the expectation of $R^\rho(i)$ to a *high-probability bound*: we want to show that $R^\rho(i) < 3D$ holds except with a small probability of failure. The key tool is the *Chernoff bound*, which gives high probability bounds of sums of independent samples. The libraries in Ellora include a mechanized proof of a generalized Chernoff bound, with the following statement (leaving off some of the parameters):

```
lemma Chernoff n(d:mem distr)(X:int → mem → bool):
  let X m = Σ_{i∈[1,n]} X i m in
  E[X|d] ≤ μ ∧ indep [0,n] (X i) d
  ⇒ Pr[X > (1+δ)μ|d] ≤ (e^δ/(1+δ)^{1+δ})^μ
```

Returning to the proof, we bounded the expectation of R^ρ in the previous step. To apply the Chernoff bound, we need to show that for any packet i, the expressions $H^\rho(i,j)$ are independent for all j. This is not exactly true, since $H^\rho(i,j_1)$ and $H^\rho(i,j_2)$ both depend on the (random) destination $\rho(i)$ of packet i. However, it suffices to show that these variables are independent if we fix the value of $\rho(i)$; then we can apply the Chernoff bound to upper bound R^ρ with high probability.

Finally, we can bound the total delay of any packet. This portion of the proof rests on an intricate loop invariant assigning an imaginary coin for each delay step to some packet that crosses P. By showing that each packet holds at most one coin, we can conclude that the i's delay is at most the number $R^\rho(i)$ of crossing packets. With our high probability bound from the previous step, we can show that $T = 4D + 1$ timesteps is sufficient to route all packets i to $\rho(i)$, except with some small probability:

$$\{T = 4D + 1\}\ \mathsf{route}\ \{\Pr[\exists i.\ \mathsf{pos}[i] \neq \rho[i]] \leq 2^{-2D}]\}$$

E.6 Coupon Collector in More Details

Our second example is the *coupon collector* process. The algorithm draws a uniformly random coupon (we have N coupon) on each day, terminating when it has drawn at least one of each kind of coupon. The code of the algorithm is as follows.

```
proc coupon (N : int) :
  var int cp[N], t[N];
  var int X ←0;
  for p ← 1 to N do
    ct ← 0;
    cur ←$ [1, N];
    while cp[cur] = 1 do
      ct ← ct + 1;
```

```
    cur ←$ [1, N];
  t[p] ←ct;
  cp[cur] ← 1;
  X ← X + t[p];
return X
```

The array cp keeps track of the coupons seen so far; initially $\mathsf{cp}[i] = 0$. We divide the loop into a sequence of phases (the outer loop) where in each phase we repeatedly sample coupons and wait until we see a new coupon (the inner loop). We keep track of the number of steps we spend in each phase p in $\mathtt{t}[p]$, and the total number of steps in X.

Our goal is to bound the average number of iterations. The code involves two nested loops, so we have two loop invariants. The inner loop has a probabilistic guard: at every iteration, there is a finite probability that the loop terminates (i.e., if we draw a new coupon), but there is no finite bound on the number of iterations we need to run. Since our desired loop invariant is not downwards closed, we must apply the rule for almost sure termination. We use a variant that is 1 if we have not seen a new coupon, and 0 if we have seen a new coupon. Note that each iteration, we have a strictly positive probability $\rho(p)$ of seeing a new coupon and decreasing the variant. Furthermore, the variant is bounded by 1, and the loop exits when the variant reaches 0. So, the side-condition $\mathcal{C}_{\mathrm{ASTerm}}$ holds. For the inner loop, we prove that forall c the invariant η_i is preserved:

$$\bigvee \begin{cases} (\Box(\mathsf{cp[cur]} = 1 \Rightarrow c \leq \mathsf{ct}) \wedge \Pr[\mathsf{cp[cur]} = 0 \wedge c = \mathsf{ct}] = (1-\rho(p))^c\rho(p)) \\ \exists k \in [1, c).\ \begin{cases} \Box(\mathsf{cp[cur]} = 1 \Rightarrow \mathsf{ct} = k) \wedge (\mathsf{cp[cur]} = 0 \Rightarrow \mathsf{ct} \leq k) \\ \Pr[\mathsf{cp[cur]} = 1 \wedge \mathsf{ct} = k] = (1-\rho(p))^k). \end{cases} \end{cases}$$

Note that this is a t-closed formula; there is an existential in the second disjunction, but it has finite domain (for fixed c).

For intuition, for every natural number c there are two cases: Either we have already unrolled more than c iterations, or not. The first disjunction corresponds to the first case, since loops where the guard is true all have $\mathsf{ct} \geq c$, and the probability of stopping at c iterations is $(1-\rho(p))^c\rho(p)$—we see c old coupons, and then a new one. Otherwise, we have the second disjunction. The integer k represents the current number of unfoldings of the loop. If the loop is continuing then $k = \mathsf{ct}$. If the loop is terminated, it terminated before the current iteration: $\mathsf{ct} < k$. Furthermore, the probability of continuing at iteration k is $(1-\rho(p))^k$. At the end of the loop we have $\Box\mathsf{cp[cur]} = 0$. So, by the first conjunct and some manipulations,

$$\forall c \in \mathbb{N}.\ \Pr[c = \mathsf{ct}] = (1-\rho(p))^c\rho(p)$$

holds when the inner loop exits, precisely describing the distribution of iterations ct as $\mathbf{Geom}(\rho(p))$ by definition.

The outer loop is easier to handle, since the loop has a fixed bound N on the number of iterations so we can use the rule for certain termination. For the loop

invariant, we take:

$$\eta_{out} \triangleq \begin{cases} \forall i \in [p-1].\ t[i] \sim \mathsf{Geom}(\rho(i)) \wedge \Box\Big(X = \sum_{i\in[p-1]} t[i]\Big) \\ \wedge\ \Box\Big(\sum_{i\in[1,N]} \mathsf{cp}[i] = p-1\Big) \\ \wedge\ \forall i \in [1,N].\ \Box(\mathsf{cp}[i] \in \{0,1\}). \end{cases}$$

The first conjunct states that the previous waiting times follow a geometric distribution with parameter $\rho(i)$; this assertion follows from the previous reasoning on the inner loop. The second conjunct asserts that X holds the total waiting time so far. The final two conjuncts state that there are at most $p-1$ flags set in cp. Thus,

$$\{\mathcal{L}\}\ \mathsf{coupon}\ \begin{Bmatrix} \forall i \in [1,N].\ t[i] \sim \mathsf{Geom}(\rho(i)) \\ \wedge\ \Box X = \sum_{i\in[1,N]} t[i] \end{Bmatrix}$$

at the end of the outer loop. By applying linearity of expectations and a fact about the expectation of the geometric distribution, we can bound the expected running time:

$$\{\mathcal{L}\}\ \mathsf{coupon}\ \left\{\mathbb{E}[X] = \sum_{i\in[1,N]} \left(\tfrac{N}{N-i+1}\right)\right\}.$$