Proving uniformity and independence by self-composition and coupling

Gilles Barthe[1], Thomas Espitau[2], Benjamin Grégoire[3]
Justin Hsu[4], and Pierre-Yves Strub[5]

[1] IMDEA Software Institute [2] Université Pierre-et-Marie-Curie
[3] Inria [4] University of Pennsylvania [5] École Polytechnique

Abstract

Proof by coupling is a powerful technique for proving probabilistic properties of two probabilistic processes, like stochastic dominance and rapid mixing of Markov chains. Couplings have also proved to be a useful abstraction for formal reasoning about relational properties of probabilistic programs, in particular for modeling reduction-based cryptographic proofs and for verifying differential privacy. In this paper, we demonstrate that probabilistic couplings can be also used for verifying *non-relational* probabilistic properties. Specifically, we show that the program logic pRHL—whose proofs correspond to proofs by coupling—can be used for proving uniformity and probabilistic independence. We formally verify our main examples using the EasyCrypt proof assistant.

1 Introduction

Uniformity and probabilistic independence are two of the most useful and commonly encountered properties when analyzing randomized computations. Uniform distributions are a central building block of randomized algorithms—arguably the simplest non-trivial distribution, the coin flip, is a uniform distribution over two values. Given access to uniform samples, there are known transformations for converting the samples to simulate more complex distributions, like the Gaussian or Laplace distributions. In the other direction, turning samples from various non-uniform distributions into uniform samples is an active area of research.

Probabilistic independence is no less central, enabling a property involving two random variables to be split into two properties involving a single random variable. For instance, we can decompose the probability of a conjunction as the product of the marginal probabilities. Independent random variables can also be analyzed with the help of more sophisticated tools, like concentration inequalities.

Given these and other applications, it is not surprising that researchers have investigated different methods of reasoning about uniformity and independence. For instance, Pearl and Paz [17] develop an axiomatic theory based on *graphoids* for modeling conditional independence in probability theory. However, proving uniformity and independence in program verification remains a challenging task. Most program logics for probabilistic programs do not treat these properties as first-class assertions, and provide reasoning principles that are cumbersome to use. Often, one must prove exact values for the probability of specific events.

For example, consider any logical formalism for proving properties of the form $\mathsf{Pr}_{[\![s]\!]_m}[E] = p$, which capture the fact that the event E has probability p in the distribution obtained by executing the randomized program s on some initial memory m; many proposed systems fall into this class [8, 12, 14, 16, 18, 19]. Suppose that we want to prove that the program variable x of finite type A is uniformly distributed w.r.t. the output distribution $[\![s]\!]_m$. The most direct way to show this property is to analyze the probability of each output: for every $a \in A$, $\mathsf{Pr}_{[\![s]\!]_m}[x = a] = \frac{1}{|A|}$.

For independence, the situation is similar. Assume that we want to prove that the two program variables x and y of respective types A and B are probabilistically independent w.r.t. the output distribution $[\![s]\!]_m$. This can be done by exhibiting functions f, g, h such that for every $a \in A$ and $b \in B$, we have: $\mathsf{Pr}_{[\![s]\!]_m}[x = a] = f(a)$, $\mathsf{Pr}_{[\![s]\!]_m}[y = b] = g(b)$, $\mathsf{Pr}_{[\![s]\!]_m}[x = a \wedge y = b] = h(a, b)$. Then, independence between x and y holds iff $h(a, b) = f(a) \cdot g(b)$ for every $a \in A$ and $b \in B$.

While these approaches can work, they can be laborious in practice. It may be awkward to directly compute the probability of $x = a$, and the functions f, g and h may be difficult to produce. We propose an alternative method based on probabilistic couplings for proving uniformity and independence. Probabilistic couplings are a classical method for proving sophisticated probabilistic properties (e.g., stochastic dominance, rapid mixing of Markov chains, and more [15, 20, 21]). More recently, couplings have been used to reason about relational properties of probabilistic programs, notably differential privacy [3, 4]. Here we show that uniformity and independence properties can also be verified using coupling, despite being non-relational properties. As a consequence, our verification method inherits the advantages of reasoning by couplings: compositional reasoning, and no need to reason directly about probabilistic events. More specifically, we demonstrate that uniformity and independence can be captured in the relational program logic pRHL [1].

Uniformity. Suppose we have a program s with a program variable x ranging over a finite set A, and we want to show that x is distributed uniformly over A after executing s. Rather than computing the probability of $\mathsf{Pr}_{[\![s]\!]_m}[x = a]$ for each $a \in A$, it suffices to show that the probabilities of any two outputs are equal:

$$\forall a_1, a_2 \in A.\ \mathsf{Pr}_{[\![s]\!]_m}[x = a_1] = \mathsf{Pr}_{[\![s]\!]_m}[x = a_2]\,.$$

Now, we can view uniformity as a relational property: if we consider two runs of s, then the probability of x being a_1 in the first run should be equal to the probability of x being a_2 in the second run. In pRHL, this property is described by the following judgment:

$$\forall a_1, a_2 \in A.\ \vDash s \sim s \ :\ \phi \Longrightarrow\ x\langle 1\rangle = a_1 \iff x\langle 2\rangle = a_2$$

where the assertion ϕ asserts that the initial states are equal.

Independence. Proving probabilistic independence is more involved, as it is unclear how to encode independence in pRHL. Nevertheless, we can prove independence in two different ways. Assume that we want to prove that the program variables x and y of respective finite types A and B are independent. First, if the distribution of $\langle x, y\rangle$ is uniformly distributed over $A \times B$, then x and y are independent (and themselves uniformly distributed). Indeed, assume that for all $a \in A$ and $b \in B$ we have $\mathsf{Pr}_{[\![s]\!]_m}[x = a \wedge y = b] = \frac{1}{|A|\cdot|B|}$. Then we have $\mathsf{Pr}_{[\![s]\!]_m}[x = a] = \sum_{b\in B} \mathsf{Pr}_{[\![s]\!]_m}[x = a \wedge y = b] = \frac{1}{|A|}$. A similar argument applies to the probability that $y = b$, from which independence follows. Thus, our first method of proving independence is by reduction to uniformity.

This approach is simple to use, but it only applies to proving independence of uniform random variables. A different approach is to express probabilistic independence as a property of a modified version of the program, without any requirement on uniformity. More specifically, independence of x and y can be derived from the equality between the probabilities of $x = a \wedge y = b$ and $x_1 = a \wedge y_2 = b$, where in the first case the probability is taken over the output of the original program s, and in the second case the probability is taken over the output of the program $s_1; s_2$, where s_1 and s_2 are renamings of s; we call $s_1; s_2$ a *self-composition* of s [2, 11]. The reason is not hard to see. Since the composed programs operate on disjoint memory, the final combined output distribution models two independent runs of the original program s. So, the probability $\mathsf{Pr}_{[\![s_1;s_2]\!]_{m_1 \uplus m_2}}[x_1 = a \land y_2 = b]$—$m_1 \uplus m_2$ is the disjoint union of two copies of m—is equal to the product of $\mathsf{Pr}_{[\![s_1]\!]_{m_1}}[x_1 = a]$ and $\mathsf{Pr}_{[\![s_2]\!]_{m_2}}[y_2 = b]$. Since s_1 and s_2 both model the original program s, this probability is in turn equal to $\mathsf{Pr}_{[\![s]\!]_m}[x = a]$ and $\mathsf{Pr}_{[\![s]\!]_m}[y = b]$ in the original program.

Our encoding casts independence as a relational property between a program s and its self-composition $s_1; s_2$, a property which can be directly expressed in the relational program logic pRHL [1]:

$$\forall a \in A, b \in B.\ \vDash s \sim s_1; s_2\ :\ \phi \Longrightarrow\ (x\langle 1\rangle = a \land y\langle 1\rangle = b)\ \Longleftrightarrow\ (x_1\langle 2\rangle = a \land y_2\langle 2\rangle = b)$$

where the precondition ϕ captures the initial conditions. We show that our approach extends to independence and conditional independence of sets of program variables.

Outline Sections 2 and 3 provide the relevant mathematical background and introduce the setting of our work. Section 4, 5 and 6 respectively address the case of uniformity, independence, and conditional independence. In each case we demonstrate our method using classic examples of randomized algorithms. We conclude the paper with a discussion of alternative frameworks for verifying these properties.

2 Mathematical background

For the sake of simplicity, we restrict ourselves to discrete (countable) sub-distributions.

Definition 1. *A* sub-distribution *over a set A is defined by a mass function $\mu : A \to \mathbb{R}^+$, which 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 the* weight *satisfies $|\mu| \triangleq \sum_{a\in A} \mu(a) \leq 1$. In particular, the* support *of the sub-distribution $\mathsf{supp}(\mu) \triangleq \{a \in A \mid \mu(a) \neq 0\}$ is discrete. When $|\mu|$ is equal to 1, we call μ a* distribution. *We let $\mathbb{D}(A)$ denote the set of sub-distributions over A. An event over A is a predicate over A. The probability of an event E w.r.t. a sub-distribution μ, written $\mathsf{Pr}_{x\sim\mu}[E]$, is defined as $\sum_{\{x\in A \mid E(x)\}} \mu(x)$.*

When working with sub-distributions over tuples, the probabilistic versions of the usual projections on tuples are called *marginals.* For distributions over pairs, we define the *first* and *second marginals* $\pi_1(\mu)$ and $\pi_2(\mu)$ of a distribution μ over $A \times B$ by $\pi_1(\mu)(a) \triangleq \sum_{b\in B} \mu(a, b)$ and $\pi_2(\mu)(b) \triangleq \sum_{a\in A} \mu(a, b)$. We are now ready to formally define coupling.

Definition 2. *Let A_1 and A_2 be two sets, and let $\Psi \subseteq A_1 \times A_2$. A Ψ-coupling for two sub-distributions μ_1, μ_2 resp. over A and B is a sub-distribution $\mu \in \mathbb{D}(A \times B)$ such that $\pi_1(\mu) = \mu_1$ and $\pi_2(\mu) = \mu_2$ and $\mathsf{supp}(\mu) \subseteq \Psi$. We write $\blacktriangleleft_\Psi \langle\mu_1\ \&\ \mu_2\rangle$ to denote the existence of a Ψ-coupling.*

In addition to the general definition, we shall also consider a special case of coupling: specifically, we say that (μ_1, μ_2) are *f-coupled* if $f : A_1 \to A_2$ is a bijection such that $\mu_1(x) = \mu_2(f(x))$ for every $x \in A_1$. In this case, we write $f \blacktriangleleft \langle\mu_1\ \&\ \mu_2\rangle$.

One useful consequence of couplings is that they can show that one event has smaller probability than another.

Lemma 3 (Fundamental lemma of coupling)**.** *Let E_1 and E_2 be predicates over A_1 and A_2, and let $\Psi \triangleq \{(x_1, x_2) \mid (x_1 \in E_1) \Rightarrow (x_2 \in E_2)\}$. If $\blacktriangleleft_{\Psi} \langle \mu_1 \,\&\, \mu_2 \rangle$, then $\Pr_{x_1 \sim \mu_1}[E_1] \leq \Pr_{x_2 \sim \mu_2}[E_2]$.*

One can immediately derive a variant of the lemma where $\iff$ and $=$ are used in place of $\Rightarrow$ and $\leq$ respectively. The following lemma provides a converse to the fundamental lemma of coupling in the special case where we are interested in proving the equality of two distributions.

Lemma 4. *For every $\mu_1, \mu_2 \in \mathbb{D}(A)$, the following are equivalent:*

- *$\mu_1 = \mu_2$;*
- *for every $a \in A$, $\Pr_{x \sim \mu_1}[x = a] = \Pr_{x \sim \mu_2}[x = a]$;*
- *for every $a \in A$, $\blacktriangleleft_{\Psi_a} \langle \mu_1 \,\&\, \mu_2 \rangle$ where $\Psi_a \triangleq \{(x_1, x_2) \mid x_1 = a \iff x_2 = a\}$;*
- *$\blacktriangleleft_{\Psi_A} \langle \mu_1 \,\&\, \mu_2 \rangle$ where $\Psi_A \triangleq \{(x_1, x_2) \mid x_1 = x_2\}$.*

We note that the third item (existence of liftings for pointwise equality) is often easier to establish than the last item (existence of lifting for equality), since one can choose the coupling for each possible value of a, rather than showing a single coupling for all values of a.

3 Setting

We will work with a simple probabilistic imperative language. Probabilistic assignments are of the form $x \xleftarrow{\$} g$, which assigns a value sampled according to the distribution g to the program variable x. The syntax of statements is defined by the grammar:

$$s ::= \textbf{skip} \mid \textbf{abort} \mid x \leftarrow e \mid x \xleftarrow{\$} g \mid s; s \mid \textbf{if } e \textbf{ then } s \textbf{ else } s \mid \textbf{while } e \textbf{ do } s$$

where x, e and g respectively range over (typed) variables in $\mathcal{X}$, expressions in $\mathcal{E}$ and distributions in $\mathcal{D}$. To ensure that the set of states is countable, we require that there are finitely many variables $\mathcal{X}$. As usual $\mathcal{E}$ is defined inductively from $\mathcal{X}$ and a set $\mathcal{F}$ of simply typed function symbols. In this paper, distributions used for sampling are either uniform distributions over a finite type A, or the Bernoulli distribution with parameter p, which we denote by $\mathbf{Bern}(p)$. We assume that expressions and statements are typed in the usual way.

We assume we are given a set-theoretical interpretation for every type and operator of the language. We define a state as a type-preserving mapping from variables to values, and we let **State** denote the set of states. The set of states is equipped with the usual functions for reading and writing a value; we use $m(x)$ to denote the value of x in m, and $m[x := v]$ to denote state update, in this case the state obtained from m by updating the value of x with v.

One can equip $\mathbb{D}(\textbf{State})$ with a monadic structure, using the Dirac distributions δ_x for the unit and *distribution expectation* $\mathbb{E}_{x \sim \mu}[M(x)]$ for the bind, where

$$\mathbb{E}_{x \sim \mu}[M(x)] : x \mapsto \sum_a \mu(a) \cdot M(a)(x).$$

The semantics of expressions and distribution expressions is parametrized by a state m, and is defined in the usual way where we require all distribution expressions to be interpreted as distributions (rather than sub-distributions).

Definition 5 (Semantics of statements)**.**

- *The semantics $[\![s]\!]_m$ of a statement s w.r.t. to some initial state m is a sub-distribution over states, and is defined by the clauses of Figure 1.*

$$\llbracket \mathbf{skip} \rrbracket_m = \delta_m \qquad \llbracket \mathbf{abort} \rrbracket_m = \mathbb{0}$$
$$\llbracket x \leftarrow e \rrbracket_m = \delta_{m[x:=\llbracket e \rrbracket_m]} \qquad \llbracket x \overset{\$}{\leftarrow} g \rrbracket_m = \mathbb{E}_{v \sim \llbracket g \rrbracket m}[\delta_{m[x:=v]}]$$
$$\llbracket s_1; s_2 \rrbracket_m = \mathbb{E}_{\xi \sim \llbracket s_1 \rrbracket_m}[\llbracket s_2 \rrbracket_\xi]$$
$$\llbracket \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2 \rrbracket_m = \text{if } \llbracket e \rrbracket_m \text{ then } \llbracket s_1 \rrbracket_m \text{ else } \llbracket s_2 \rrbracket_m$$
$$\llbracket \mathbf{while}\ b\ \mathbf{do}\ s \rrbracket_m = \lim_{n \to \infty} \llbracket (\mathbf{if}\ b\ \mathbf{then}\ s)^{[n]}; \mathbf{if}\ b\ \mathbf{then\ abort} \rrbracket_m$$

where $s^{[n]} \triangleq \overbrace{s; \dots; s}^{n \text{ times}}$.

Figure 1: Denotational semantics of programs

- *The (lifted) semantics $\llbracket s \rrbracket_\mu$ of a statement s w.r.t. to some initial sub-distribution μ over states is a sub-distribution over states, and is defined as $\llbracket s \rrbracket_\mu \triangleq \mathbb{E}_{m \sim \mu}[\llbracket s \rrbracket_m]$ $\mu \in \mathbb{D}(\mathbf{State})$.*

A basic and highly important property of probabilistic programs is termination. We say that a program s is *lossless* if for every initial memory m, $|\llbracket s \rrbracket_m| = 1$. By now, there are many sophisticated techniques for proving losslessness even for languages that allow both probabilistic sampling and non-determinism (including recent advances by Chatterjee et al. [9, 10], Ferrer Fioriti and Hermanns [13]). These techniques are capable of showing losslessness for all of our examples (in some cases with a high degree of automation), so throughout the paper, we assume that all programs are lossless. This assumption is used in the rules of pRHL and the characterizations of uniformity and independence.

3.1 Self-composition of programs

For every program s and $n \in \mathbb{N}$, we let $s^{\langle n \rangle}$ denote the n-fold self-composition of s, i.e. $s^{\langle n \rangle} \triangleq s_1; \dots, s_n$, where each $s_\imath$ is a copy of s where all variables are tagged with a superscript $\imath$. In order to state the main property of self-composition, we define the self-composition of a state; given a state m, we define its n-fold self-composition $m^{\langle n \rangle}$ as the state from $\mathcal{X}^{\langle n \rangle}$ to values, where $\mathcal{X}^{\langle n \rangle} \triangleq \{x^\imath \mid x \in \mathcal{X}, 1 \leq \imath \leq n\}$ such that for every x and $\imath$, $m^{\langle n \rangle}(x^\imath) \triangleq m(x)$. Given a state m from $\mathcal{X}^{\langle n \rangle}$, we denote by $m_\imath$ the $\imath$-th projection of m.

Proposition 6. *For every program s and state m*

$$\mathsf{Pr}_{\llbracket s^{\langle n \rangle} \rrbracket_{m^{\langle n \rangle}}} [\wedge_{1 \leq \imath \leq n} E^\imath_\imath] = \prod_{1 \leq \imath \leq n} \mathsf{Pr}_{\llbracket s \rrbracket_m} [E_\imath]$$

where the event $E^\imath$ is defined by $E^\imath(m'^{\langle n \rangle}) \triangleq E(\pi_\imath(m'))$ for every $\imath$ and $\pi_\imath$ is the projection from a self-composed state to its $\imath$-th component.

3.2 Probabilistic Relational Hoare Logic

Probabilistic Relational Hoare Logic (pRHL) is a program logic for reasoning about relational properties of probabilistic programs. Its judgments are of the form $\vDash s_1 \sim s_2 \ : \ \phi \implies \ \psi$, where s_1 and s_2 are commands and the pre-condition ϕ and the post-condition ψ are relational assertions, i.e. first-order formulae built over generalized expressions. The latter are similar to expressions, except that each variable is tagged with $\langle 1 \rangle$ or $\langle 2 \rangle$ to indicate the execution that it belongs to; we call the two executions *left* and *right*. Generalized expressions are interpreted w.r.t. a pair (m_1, m_2) of states, where the interpretation of the tagged variables $x\langle 1\rangle$ and $x\langle 2\rangle$ $m_1(x)$ and $m_2(x)$ respectively. We write $(m_1, m_2) \vDash \phi$ to denote that the interpretation of the assertion ϕ w.r.t. (m_1, m_2) is valid.

Definition 7. *A judgment* $\vDash s_1 \sim s_2 \ : \ \phi \Longrightarrow \ \psi$ *is valid iff for every states* m_1 *and* m_2*,* $(m_1, m_2) \vDash \phi$ *implies* $\blacktriangleleft_{\{(m_1', m_2') \mid (m_1', m_2') \vDash \psi\}} \langle [\![s_1]\!]_{m_1} \ \& \ [\![s_2]\!]_{m_2} \rangle$.

Figure 2 presents the main rules of the logic. This includes two-sided rules, which operate on both programs, and one-sided rules, which operate on a single program (left or right). Note that the rule [Struct] uses an auxiliary judgment that captures program equivalence and is defined in Figure 3. Note that structural rules include rules for loop splitting and swapping programs that operate on disjoint parts of the state, where $\mathsf{var}(s)$ denotes the set of variables read and written by the statement s. We refer to Barthe et al. [1, 6] for an explanation of the rules.

Throughout the paper, we often assert that the left and the right copies of a state are equal. This is captured by the relational assertion $\mathsf{EqMem} \triangleq \bigwedge_{x \in \mathcal{X}} x\langle 1\rangle = x\langle 2\rangle$. We also often assert cross-equality on n-fold composition of states $\mathsf{EqMem}^{\langle p\rangle, \langle q\rangle} \triangleq \bigwedge_{x \in \mathcal{X}, 1 \le \imath \le p, 1 \le \jmath \le q} x^{\imath}\langle 1\rangle = x^{\jmath}\langle 2\rangle$.

4 Uniformity

Reasoning about probabilistic programs often requires establishing that a set of program variables (each ranging over a finite type) is uniformly distributed:

Definition 8. *A set* $X = \{x_1, \dots, x_n\}$ *of program variables of finite types* $A_1, \dots, A_n$ *is* uniformly distributed *w.r.t. a distribution* $\mu \in \mathbb{D}(\mathbf{State})$ *iff for every* $(a_1, \dots, a_n) \in A_1 \times \dots \times A_n$*:*

$$\mathsf{Pr}_{\mu}\left[\bigwedge_{1 \le i \le n} x_i = a_i\right] = \prod_{1 \le i \le n} \frac{1}{|A_i|}$$

Note that the definition of uniformity (and in later sections of independence) naturally extends to sets of expressions, and so do our characterizations.

4.1 Characterization

The following proposition characterizes uniformity in terms of couplings.

Proposition 9 (Uniformity by coupling)**.** *Let* $X = \{x_1, \dots, x_n\}$ *be a set of variables of respective finite types* $A_1, \dots, A_n$*. For every program* s*, the following are equivalent:*

1. *for every state* m*,* X *is uniformly distributed w.r.t.* $[\![s]\!]_m$*;*
2. *for every two tuples* $(a_1, \dots, a_n), (a_1', \dots, a_n') \in A_1 \times \dots \times A_n$*,*

$$\vDash s \sim s \ : \ \mathsf{EqMem} \Longrightarrow \left(\bigwedge_{1 \le i \le n} x_i\langle 1\rangle = a_i\right) \Longleftrightarrow \left(\bigwedge_{1 \le i \le n} x_i\langle 2\rangle = a_i'\right)$$

Proof. Follows directly from properties of coupling. □

By expressing uniformity as a coupling property, we can use pRHL to prove uniformity. To demonstrate the technique, we consider classical examples randomized algorithms.

4.2 Simulating a fair coin

$$\textsc{Conseq}\ \frac{\begin{array}{c}\models s_1 \sim s_2 \ :\ \Phi \Longrightarrow \Psi \\ \Phi' \Longrightarrow \Phi \qquad \Psi \Longrightarrow \Psi'\end{array}}{\models s_1 \sim s_2 \ :\ \Phi' \Longrightarrow \Psi'} \qquad \textsc{Struct}\ \frac{\begin{array}{c}\models s_1 \sim s_2 \ :\ \Phi \Longrightarrow \Psi \\ \Phi \vdash s_1 \equiv s_1' \qquad \Phi \vdash s_2 \equiv s_2' \qquad \Phi \vdash s \equiv s'\end{array}}{\models s_1' \sim s_2' \ :\ \Phi \Longrightarrow \Psi}$$

$$\textsc{Case-L}\ \frac{\begin{array}{c}\models s_1 \sim s_2 \ :\ \Phi \wedge e\langle 1\rangle \Longrightarrow \Psi \\ \models s_1 \sim s_2 \ :\ \Phi \wedge \neg e\langle 1\rangle \Longrightarrow \Psi\end{array}}{\models s_1 \sim s_2 \ :\ \Phi \Longrightarrow \Psi} \qquad \textsc{Seq}\ \frac{\begin{array}{c}\models s_1 \sim s_2 \ :\ \Phi \Longrightarrow \Xi \\ \models s_1' \sim s_2' \ :\ \Xi \Longrightarrow \Psi\end{array}}{\models s_1; s_1' \sim s_2; s_2' \ :\ \Phi \Longrightarrow \Psi}$$

$$\textsc{Assg}\ \frac{\Phi \triangleq \Psi[e_1\langle 1\rangle/x_1\langle 1\rangle, e_2\langle 2\rangle/x_2\langle 2\rangle]}{\models x_1 \leftarrow e_1 \sim x_2 \leftarrow e_2 \ :\ \Phi \Longrightarrow \Psi}$$

$$\textsc{Rand}\ \frac{\begin{array}{c} f \blacktriangleleft \langle g_1 \ \&\ g_2\rangle \\ \Phi \triangleq \forall (v_1, v_2). f(v_1) = v_2 \Rightarrow \Psi[v_1/x_1\langle 1\rangle, v_2/x_2\langle 2\rangle]\end{array}}{\models x_1 \stackrel{\$}{\leftarrow} g_1 \sim x_2 \stackrel{\$}{\leftarrow} g_2 \ :\ \Phi \Longrightarrow \Psi}$$

$$\textsc{Cond}\ \frac{\begin{array}{c}\Phi \Longrightarrow e_1 = e_2 \\ \models s_1 \sim s_2 \ :\ \Phi \wedge e_1 \Longrightarrow \Psi s \qquad \models s_1' \sim s_2' \ :\ \Phi \wedge \neg e_1 \Longrightarrow \Psi s'\end{array}}{\models \textbf{if } e_1 \textbf{ then } s_1 \textbf{ else } s_1' \sim \textbf{if } e_2 \textbf{ then } s_2 \textbf{ else } s_2' \ :\ \Phi \Longrightarrow \Psi}$$

$$\textsc{While}\ \frac{\models s_1 \sim s_2 \ :\ \Psi \wedge e_1\langle 1\rangle \wedge e_2\langle 2\rangle \Longrightarrow \Psi \wedge e_1\langle 1\rangle = e_2\langle 2\rangle}{\models \textbf{while } e_1 \textbf{ do } s_1 \sim \textbf{while } e_2 \textbf{ do } s_2 \ :\ \Psi \wedge e_1\langle 1\rangle = e_2\langle 2\rangle \Longrightarrow \Psi \wedge \neg e_1\langle 1\rangle \wedge \neg e_2\langle 2\rangle}$$

$$\textsc{Assg-L}\ \frac{}{\models x_1 \leftarrow e_1 \sim \textbf{skip} \ :\ \Psi[e_1\langle 1\rangle/x_1\langle 1\rangle] \Longrightarrow \Psi}$$

$$\textsc{Rand-L}\ \frac{}{\models x_1 \stackrel{\$}{\leftarrow} g_1 \sim \textbf{skip} \ :\ \forall v_1 \in \mathsf{supp}(g_1), \Psi[v_1/x_1\langle 1\rangle] \Longrightarrow \Psi}$$

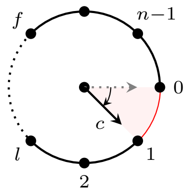
$$\textsc{Cond-L}\ \frac{\begin{array}{c}\models s_1 \sim s_2 \ :\ \Phi \wedge e_1\langle 1\rangle \Longrightarrow \Psi \\ \models s_1' \sim s_2 \ :\ \Phi \wedge \neg e_1\langle 1\rangle \Longrightarrow \Psi\end{array}}{\models \textbf{if } e_1 \textbf{ then } s_1 \textbf{ else } s_1' \sim s_2 \ :\ \Phi \Longrightarrow \Psi}$$

$$\textsc{While-L}\ \frac{\models s_1 \sim \textbf{skip} \ :\ \Psi \wedge e_1\langle 1\rangle \Longrightarrow \Psi}{\models \textbf{while } e_1 \textbf{ do } s_1 \sim \textbf{skip} \ :\ \Psi \Longrightarrow \Psi \wedge \neg e_1\langle 1\rangle}$$

Figure 2: Proof rules

$$\textsc{While-Split}\ \frac{}{\Phi \vdash \textbf{while } e \textbf{ do } s \equiv \textbf{while } e \wedge e' \textbf{ do } s; \textbf{while } e \textbf{ do } s} \qquad \textsc{Swap}\ \frac{\mathsf{var}(s_1) \cap \mathsf{var}(s_2) = \emptyset}{\Phi \vdash s_1; s_2 \equiv s_2; s_1}$$

Figure 3: Selected equivalence rules



```
$d \leftarrow 0;\ c \leftarrow 0;\ f \leftarrow 0;\ l \leftarrow 0;$
while $l + 1 \leq f$ do
    $d \xleftarrow{\$} \mathcal{U}_{\{-1,1\}};$
    if $c = l \wedge d = 1$ then $l \leftarrow l + 1;$
    if $c = f \wedge d = -1$ then $f \leftarrow f - 1;$
    $c \leftarrow c + d;$
$ret \leftarrow (l, l + 1)$
```

Figure 5: Cyclic random walk

```
$x \leftarrow 0;$
$y \leftarrow 0;$
while $x = y$ do
    $x \xleftarrow{\$}$ Bern$(p)$;
    $y \xleftarrow{\$}$ Bern$(p)$;
```

Figure 4: Bernoulli uniformizer

This example considers a process for simulating a fair coin using a biased coin. The idea is simple: 1) toss the coin twice; 2) if the two outcomes differ, return the value of the first coin; 3) if the two outcomes match, repeat from step 1. The algorithm does not require the bias of the coin to be known, as long as it is some constant bias and there is positive probability of returning 0 and 1. This process can be modelled by the program s from Figure 4, where $0 < p < 1$ is a real parameter modeling the probability of the biased coin to return 0 (tail). Our goal is to establish the trivial judgment $\{\top\}\ s\ \{\top\}$ and the following pRHL judgment:

$$\models s \sim s\ :\ \top \Longrightarrow\ x\langle 1\rangle\ \Longleftrightarrow\ \neg x\langle 2\rangle$$

By the fundamental lemma of the coupling, this implies that $\mathsf{Pr}_{[\![s]\!]_m}[x = 1] = \mathsf{Pr}_{[\![s]\!]_m}[x = 0]$, and hence that x is uniformly distributed upon termination. The proof proceeds by establishing the following invariant:

$$x\langle 2\rangle = \text{if } x\langle 1\rangle = y\langle 1\rangle \text{ then } y\langle 2\rangle \text{ else } \neg x\langle 1\rangle$$

Validity of the invariant entails that the desired postcondition holds on program exit, as the invariant and the negation of the loop guard hold. The invariant holds on entering the loop, so we only need to prove that it is preserved by the loop body. The proof proceeds as follows: first, we swap the two random assignments on the right, leading to the judgment:

$$\models (x \xleftarrow{\$} \mathbf{Bern}(p);\quad y \xleftarrow{\$} \mathbf{Bern}(p)) \sim (y \xleftarrow{\$} \mathbf{Bern}(p);\quad x \xleftarrow{\$} \mathbf{Bern}(p))\ :\ \phi' \Longrightarrow\ \phi$$

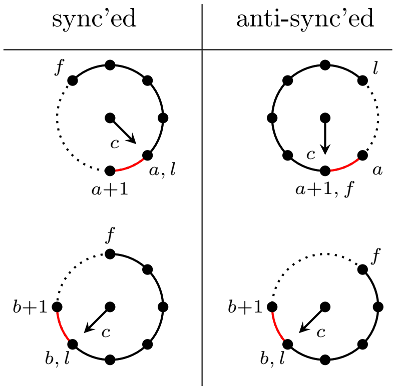
where ϕ denotes the loop invariant and ϕ' denotes its strengthening by the loop guard—we do not need the precondition, since the values are freshly sampled in the body. Next, we apply the [Rand] rule twice, with the identity bijection. This yields as precondition the tautology:

$$\forall v_1, v_2,\ v_2 = (\text{if } v_1 = v_2 \text{ then } v_1 \text{ else } \neg v_1).$$

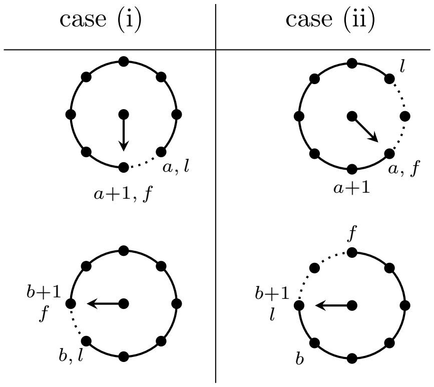
4.3 Cyclic random walk

Consider a random walk over a cyclic path composed of n nodes labeled $0, 1, \ldots, n-1$: starting from position 0, at each step, we flip a fair coin over $\{-1, 1\}$ and update the position accordingly to the result of the coin flip. To take into account that we are on a cyclic structure, all arithmetical operations are in the cyclic ring $\mathbb{Z}/n\mathbb{Z}$—i.e. are performed modulo n. At each iteration, when moving between two contiguous positions over the circle, we consider that the random walk visited the arc between the two nodes. We want to show that the last visited arc is uniformly distributed. Figure 5 (left) gives a graphical representation of the random walk, where c is the random walk position and the red arc is the last visited arc when c moved from 0 to 1.

This random walk can be seen as a simple version of sampling a uniformly random spanning tree on a graph. While Broder [7] analyzes the general problem, we can verify uniformity for the cyclic random walk with couplings.



The proof proceeds as follows. We first execute asynchronously the two random walks until they eventually synchronize respectively on the arcs $(a, a+1)$ and $(b, b+1)$. At that point, we are in one of the following cases: either the random walks synchronize on the same side of the arcs $(a, a+1)$ and $(b, b+1)$; either they synchronize on opposite sides. (These cases are examplified in the diagrams on the left, where the arc we want to synchronize on is in red). From that point, we execute the two processes resp. in lock-step (if they synchronized on the same side) or anti-lock-step (if they did not). At some point, both processes will visit the other side of the arcs $(a, a+1)$ and $(b, b+1)$, and since they execute in (anti)-lock-step, these events will occur synchronously. At that point, either the processes finished their walk and they resp. return the arcs $(a, a+1)$ and $(b, b+1)$ as their result (case (i) of the right diagram below); either they have other nodes to visit and they *will not* resp. return the arcs $(a, a+1)$ and $(b, b+1)$ (case (ii) of the same diagram).



We now detail the formal proof. Consider the program of Figure 5, where all arithmetical operations are done modulo n. This algorithm instruments the random walk with two points f and l representing the range $[f, l]$ (using clockwise ordering) of all the points that have been visited by the walk. When all nodes of the cycle have been visited (i.e. when $l + 1 = f$), the arc between l and $l + 1$ is the only arc that has not been visited by the walk. Let s be the program of Figure 5 and s' the loop body of the single loop of s. We want to show that the final arc *ret*—the only arc that has not been marked—is uniformly distributed among all arcs. Uniformity of *ret* is entailed by the following judgment:

$$\forall a, b \in \mathbb{Z}/_{n\mathbb{Z}}, \models s \sim s \ : \ \top \implies ret\langle 1\rangle = (a, a+1) \iff ret\langle 2\rangle = (b, b+1).$$

First, we make use of the loop splitting equivalence rules to transform the main loop into three pieces. We transform the loop in the left program into:

```
while $(|v| < n \wedge a, a+1 \notin [f, l])$ do $s'$;
while $(|v| < n \wedge \neg(a, a+1 \in [f, l]))$ do $s'$;
while $(|v| < n)$ do $s'$
```

where $[f, l]$ represent the range $f, f+1, \ldots, l$. We use a similar transformation on the right program, with b in place of a. To carry out the proof, we first use the one-sided loop rules ([While-L] and the corresponding version [While-R]) on the first loops of the left and right programs. This part of the proof correspond to the walks synchronization as described above. By a straightforward loop invariant, we can show that

$$\Phi \wedge P(a)\langle 1\rangle \wedge P(b)\langle 2\rangle$$

holds after the first loops, where $P(x) \triangleq (x \in [f,l] \oplus (x+1) \in [f,l])$ and $\Phi \triangleq \forall i \in \{1,2\}. (c \in [f,l])\langle i\rangle$ indicates that the current positions $(c\langle 1\rangle, c\langle 2\rangle)$ are contained in the range of visited arcs. Next, we show that after the two second loops the following relational invariant is satisfied:

$$(a, a+1 \in [f,l])\langle 1\rangle \wedge (b, b+1 \in [f,l])\langle 2\rangle \wedge (l\langle 1\rangle = a \iff l\langle 2\rangle = b)$$

After the second loop, there are two cases for the third loop. If $l\langle 1\rangle = a$, we have $l\langle 2\rangle = b$, $f\langle 1\rangle = a+1$ (since $a+1$ is visited in $\langle 1\rangle$) and $f\langle 2\rangle = b+1$. In that case, which correspond to the case (i) of the last diagram, the third loops both exit immediately and the random walks resp. return the arcs $(a, a+1)$ and $(b, b+1)$. Otherwise, we have $l\langle 1\rangle \neq a$ and $l\langle 2\rangle \neq b$, and we can show, using the rules [While-L] and [While-R], that $l\langle 1\rangle$ (resp. $l\langle 2\rangle$) will never be set to a (resp. b). In that case, which correspond to the case (ii) of the last diagram, we can show that the walks resp. return arcs distinct from $(a, a+1)$ and $(b, b+1)$.

We now focus on the the second loops, relating them with the two-sided variant of the [While] rule. The particular coupling we choose will depend on the the current positions in the two sides at the start of the second loops. If $a \in [f,l]\langle 1\rangle$ and $b \in [f,l]\langle 2\rangle$ then we have $l\langle 1\rangle = a$ and $l\langle 2\rangle = b$ (since $a+1 \notin [f,l]\langle 1\rangle$) and we couple the walks to make identical moves. In that case, the key part of the loop invariant is:

$$\bigwedge \left\{ \begin{array}{c} \Phi \wedge (a \in [f,l])\langle 1\rangle \wedge (b \in [f,l])\langle 2\rangle \\ c\langle 1\rangle - a = c\langle 2\rangle - b \\ l\langle 1\rangle = a \iff l\langle 2\rangle = b \end{array} \right\}$$

The first line enforces some structural invariant and the second line enforces that both walks make identical moves, relatively to a and b. The main difficulty is to show that both loops are synchronized. Note that there are two reasons the loop may exit. If l has been incremented, then the increment will be done on both side. Otherwise, if f has been decremented to $a+1$, then we have $c\langle 1\rangle = f\langle 1\rangle = a+2$, so $c\langle 1\rangle - a = c\langle 2\rangle - b = 2$ and $c\langle 2\rangle = b+2$ and the right loop will also decrement f to $b+1$. The case $a+1 \in [f,l]\langle 1\rangle$ and $b+1 \in [f,l]\langle 2\rangle$ is very similar, by reversing the roles of f and l. The remaining two cases, $a+1 \in [f,l]\langle 1\rangle$ and $b \in [f,l]\langle 2\rangle$ or $a \in [f,l]\langle 1\rangle$ and $b+1 \in [f,l]\langle 2\rangle$ is similar except that we force the walks to be execute in anti-lock-step. Using the rule [Case], we put together these four cases and we conclude by application of the rule for sequence.

4.4 Ballot Theorem

So far, we have shown how couplings can be used to prove that a set of program variables is uniformly distributed. Couplings can also be used for showing that two events have the same probability, as shown by the following example.

Example 10 (Ballot Theorem)**.** *Assume that voters must choose between two candidates A and B. The outcome of the vote is n_A votes for A and n_B votes for B, with $n_A > n_B$. Assuming that the order in which the votes are cast is uniformly random, the probability that A is always strictly ahead in partial counts is* $(n_A - n_B)/(n_A + n_b)$.

The process can be formalized by the program from Figure 6. Here we use the list l to store intermediate results. Using l_i to denote the i-th element of the list l, the Ballot Theorem is captured by the statement:

$$\forall n_A, n_B . n_A > n_B \implies \mathsf{Pr}_{[\![s]\!]_m}\left[\bigwedge_{1\leq i\leq n} l_i \neq 0 \;\middle|\; x_A = n_A \wedge x_B = n_B \right] = \frac{n_A - n_B}{n_A + n_B}.$$

```
$r \leftarrow 0; x_A \leftarrow 0; x_B \leftarrow 0; l \leftarrow \epsilon$;
while $|l| \leq n$ do
  $r \xleftarrow{\$} \{A, B\}$;
  if $r = A$ then ;
    $x_A \leftarrow x_A + 1$;
  else
    $x_B \leftarrow x_B + 1$;
  $l \leftarrow l :: (x_A - x_B)$
```

Figure 6: Ballot theorem

There exist many proofs of the Ballot's Theorem; we focus on the so-called Andre's reflection method. The crux of the method is a coupling proof of the following fact: "bad" sequences starting with a vote to to the loser are equiprobable with "bad" sequences starting with a vote to the winner, where a sequence of votes is "bad" if there is a tie at some point in the partial counts. Let $\phi \triangleq (\bigvee_{1\leq i\leq n} l_i = 0)$ and $\psi \triangleq x_A = n_A \wedge x_B = n_B$. The above facts are captured by the pRHL judgment (universally quantified over n_A and n_B such that $n_A > n_B$): $\vDash s \sim s \;:\; \top \Longrightarrow \; \xi$ where

$$\xi \triangleq (l_1 \cdot l_n > 0 \wedge \phi \wedge \psi)\langle 1\rangle \iff (l_1 \cdot l_n < 0 \wedge \phi \wedge \psi)\langle 2\rangle.$$

It follows from the properties of coupling that for every n_A and n_B such that $n_A > n_B$, $\mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n > 0 \wedge \phi \wedge l_n = k] = \mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n < 0 \wedge \phi \wedge l_n = k]$. In terms of conditional probabilities, we have $\mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n > 0 \wedge \phi \mid \psi] = \mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n < 0 \wedge \phi \mid \psi]$. Now observe that any sequence that starts with a vote to B (i.e. the loser) is necessarily bad. Therefore, $\mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n < 0 \wedge \phi \mid \psi] = \mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n < 0 \mid \psi]$. By the above and elementary properties of conditional independence:

$$\begin{aligned}\mathsf{Pr}_{[\![s]\!]_m}[\phi \mid \psi] &= \mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n > 0 \wedge \phi \mid \psi] + \mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n < 0 \wedge \phi \mid \psi] \\ &= 2 \cdot \mathsf{Pr}_{[\![s]\!]_m}[l_1 \cdot l_n < 0 \mid \psi].\end{aligned}$$

Note that the probability in the right-hand side of the last equation represents the probability that the first vote goes to the loser, conditional on ψ. This turns out to be exactly $\frac{n_B}{n_A+n_B}$, so we conclude that $\mathsf{Pr}_{[\![s]\!]_m}[\phi \mid \psi] = 2 \cdot \frac{n_B}{n_A+n_B}$ or equivalently $\mathsf{Pr}_{[\![s]\!]_m}[\neg\phi \mid \psi] = \frac{n_A - n_B}{n_A+n_B}$ as desired.

We now turn to the proof of the pRHL judgments. By symmetry it suffices to consider the first judgment. Using the rule of consequence and the elimination rule for universal quantification, it suffices to prove for every i:

$$\vDash s \sim s \;:\; \top \Longrightarrow \; l_1\langle 1\rangle \cdot l_n\langle 1\rangle > 0 \wedge l_i\langle 1\rangle = 0 \wedge \psi\langle 1\rangle \Rightarrow l_1\langle 2\rangle \cdot l_n\langle 2\rangle < 0 \wedge l_i\langle 2\rangle = 0 \wedge \psi\langle 2\rangle$$

We couple the samplings of x using the negation function until $|l| = i$, and then with the identity bijection. This establishes the following loop invariant, from which we can conclude:

$$(\forall j \leq i.\ l_j\langle 1\rangle = -l_j\langle 2\rangle) \wedge l_i\langle 1\rangle = l_i\langle 2\rangle = 0 \wedge (\forall j > i.\ l_j\langle 1\rangle = l_j\langle 2\rangle).$$

5 Independence

We now turn to characterizing probabilistic independence using couplings. We focus on probabilistic independence of program variables, a common task when reasoning about randomized computations. In our setting, the textbook definition of probabilistic independence can be cast as follows:

Definition 11. *A set $X = \{x_1, \ldots, x_n\}$ of program variables of types $A_1, \ldots, A_n$ is probabilistically independent w.r.t. a distribution $\mu \in \mathbb{D}(\mathbf{State})$ iff for every $(a_1, \ldots, a_n) \in\in A_1 \times \ldots \times A_n$:*

$$\mathsf{Pr}_{\mu}\left[\bigwedge_{1 \leq i \leq n} x_i = a_i\right] = \prod_{1 \leq i \leq n} \mathsf{Pr}_{\mu}\left[x_i = a_i\right].$$

5.1 Characterization

Our first characterization of independence is based on the observation that uniformity entails independence.

Fact 12 (Independence from uniformity)**.** *From every state m, if X is uniform w.r.t. $[\![s]\!]_m$ then X is independent w.r.t. $[\![s]\!]_m$.*

This observation enables proving independence by coupling, in the special case where variables are uniform and independent. For the general case, we will use an alternative characterization based on self-composition.

Proposition 13 (Independence by coupling)**.** *The following are equivalent:*

1. *for every state m, X is independent w.r.t. $[\![s]\!]_m$;*
2. *the following judgment, between a single copy of the program on the one hand and a n-fold copy on the other hand, is derivable for every $(a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$:*

$$\vDash s \sim s^{\langle n \rangle} \;:\; \mathsf{EqMem}^{\langle 1 \rangle, \langle n \rangle} \Longrightarrow \bigwedge_{1 \leq i \leq n} x_i\langle 1 \rangle = a_i \iff \bigwedge_{1 \leq i \leq n} x_i^i\langle 2 \rangle = a_i$$

Proof. The validity of the universally quantified pRHL judgment is equivalent to the following statement: for every $a_1, \ldots, a_n$ and n-fold copy $m^{\langle n \rangle}$ of some initial state m,

$$\mathsf{Pr}_{[\![s]\!]_m}\left[\bigwedge_{1 \leq i \leq n} x_i = a_i\right] = \mathsf{Pr}_{[\![s^{\langle n \rangle}]\!]_{m^{\langle n \rangle}}}\left[\bigwedge_{1 \leq i \leq n} x_i^i = a_i\right] = \prod_{1 \leq i \leq n} \mathsf{Pr}_{[\![s]\!]_m}\left[x_i = a_i\right].$$

The last equality comes from the property of n-fold self-composition 6. □

5.2 Pairwise independence of bits

Our first example is a well-known algorithm for generating 2^n pairwise independent bits. The algorithm first samples n independent bits $b_1 \ldots b_n$, and then defines for every subset $X \subseteq \{1, \ldots, n\}$ the bit $z_X = \bigoplus_{i \in X} b_i$. We can prove pairwise independence of the computed bits, i.e. for every $X \neq Y$, z_X and z_Y are independent. Since there are 2^n subsets of $\{1, \ldots, n\}$, this gives us 2^n pairwise independent bits constructed from n independants bits. The algorithm is encoded by the program s in Figure 7, where bits maps $\{0, \ldots, 2^n - 1\}$ to a subset in $\mathcal{P}(\{1, \ldots, n\})$ of positions that are 1 in the binary representation, and **for** $i\ = a$ **to** b **do** s is usual syntactic sugar for **while** loop with an incrementing counter i.

```
for $i\ = 1$ to $n$ do
  $b_i \xleftarrow{\$} \{0, 1\}$;
for $j\ = 0$ to $2^n - 1$ do
  $z_j \leftarrow \bigoplus_{k \in \mathsf{bits}(j)} b_k$;
```

Figure 7: Pairwise independence

By our characterization based on self-composition, pairwise independence of z_j and $z_{j'}$ for every $j \neq j'$ is equivalent to the (universally quantified) pRHL judgment:

$$\models s \sim s_1; s_2 \ : \ \top \Longrightarrow \ z_j\langle 1\rangle = a \wedge z_{j'}\langle 1\rangle = a' \iff z^1_j\langle 2\rangle = a \wedge z^2_{j'}\langle 2\rangle = a'$$

Since $j \neq j'$, the two sets $\mathsf{bits}(j)$ and $\mathsf{bits}(j')$ must differ in at least one element. Let k_0 be the smallest element in which they differ. Without loss of generality, we can assume that $k_0 \notin \mathsf{bits}(j)$ and $k_0 \in \mathsf{bits}(j')$. The crux of the proof is to establish the following judgment:

$$\models s_l \sim s_r \ : \ \top \Longrightarrow \ z\langle 1\rangle = a \wedge z'\langle 1\rangle = a' \iff z\langle 2\rangle = a \wedge z''\langle 2\rangle = a'$$

where $z = \bigoplus_{k\in\mathsf{bits}(j)} b_k$, $z' = \bigoplus_{k\in\mathsf{bits}(j')} b_k$ and $z'' = \bigoplus_{k\in\mathsf{bits}(j')} b'_k$ and

$$s_l \ \triangleq \ \textbf{for } i \in [1 \dots n] \setminus k_0 \textbf{ do } b_i \xleftarrow{\$} \{0,1\}; \ b_{k_0} \xleftarrow{\$} \{0,1\}$$

$$s_r \ \triangleq \ \textbf{for } i \in [1 \dots n] \setminus k_0 \textbf{ do } (b_i \xleftarrow{\$} \{0,1\}; b'_i \xleftarrow{\$} \{0,1\}); \ b_{k_0} \xleftarrow{\$} \{0,1\}; \ b'_{k_0} \xleftarrow{\$} \{0,1\}$$

This is proved coupling the variables of the two programs in an appropriate way. We couple the random samplings as follows:

- for every $k \neq k_0$, we couple $b_k\langle 1\rangle$ and $b_k\langle 2\rangle$ using the identity sampling;
- we use the Rnd-R rule for $b'_k\langle 2\rangle$ for every $k \neq k_0$;
- we couple $b_{k_0}\langle 1\rangle$ and $b'_{k_0}\langle 2\rangle$ with the bijection which ensures:

$$b_{k_0}\langle 1\rangle \oplus \left(\bigoplus_{k\in\mathsf{bits}(j')\setminus\{k_0\}} b_k\langle 1\rangle \right) = b'_{k_0}\langle 2\rangle \oplus \left(\bigoplus_{k\in\mathsf{bits}(j')\setminus\{k_0\}} b'_k\langle 2\rangle \right)$$

Putting everything together, we obtain a proof obligation whose validity follows from the algebraic properties of $\oplus$.

5.3 k-wise independence

```
for $i = 1$ to $n$ do
  $a_i \xleftarrow{\$} \mathbb{Z}/p\mathbb{Z}$;
for $m = 0$ to $n - 1$ do
  $x_m \leftarrow 0$;
  for $j = 0$ to $k - 1$ do
    $x_m \leftarrow a_j \cdot m^j$;
```

Figure 8: k-wise independence

The previous example can be generalized to achieve k-wise independence for general k. Suppose we wish to generate n random variables that are k-wise independent. We will work in $\mathbb{Z}/p\mathbb{Z}$, the field of integers modulo a prime p, such that $k \leq p$. Let $a_0, \dots, a_{k-1}$ be drawn uniformly at random from $\mathbb{Z}/p\mathbb{Z}$ and define the family of random variables for every $m \in \{1, \dots, n\}$:

$$x_m = \sum_{j=0}^{k-1} a_j \cdot m^j ,$$

where we take $0^0 = 1$ by convention. The corresponding code is given in Figure 8. Then, we can show that any collection of k distinct variables $\{x_i\}_i$ is independent.

For simplicity, we will show that the first k elements $x_0, \ldots, x_{k-1}$ are uniform, and hence independent. Let $v_0, \ldots, v_{k-1} \in \mathbb{Z}/_{n\mathbb{Z}}$ be arbitrary elements of the field. Then the probability that $(x_0, \ldots, x_{k-1}) = (v_0, \ldots, v_{k-1})$ is equal to p^{-k}. Indeed, the equations

$$\begin{cases} v_0 &= a_0 \\ \vdots & \vdots \\ v_{k-1} &= \sum_{j=0}^{k-1} a_j \cdot (k-1)^j \end{cases}$$

define a system of linear equations with variables $a_0, \ldots, a_{k-1}$. By basic linear algebra the system of equations has a unique solution for the variables $a_0, \ldots, a_{k-1}$,[1] which we denote $(v_0^*, \ldots, v_{k-1}^*)$. Now consider the pRHL judgment that establishes uniformity:

$$\vDash s \sim s \ : \ \top \implies \ x_0\langle 1\rangle = v_0 \wedge \cdots \wedge x_{k-1}\langle 1\rangle = v_{k-1} \iff x_0\langle 2\rangle = w_0 \wedge \cdots \wedge x_{k-1}\langle 2\rangle = w_{k-1}$$

By applying (relational) weakest precondition on the deterministic fragments of the program, the judgment is reduced to

$$\vDash s \sim s \ : \ \top \implies \ a_0\langle 1\rangle = v_0^* \wedge \cdots \wedge a_{k-1}\langle 1\rangle = v_{k-1}^* \iff a_0\langle 2\rangle = w_0^* \wedge \cdots \wedge a_{k-1}\langle 2\rangle = w_{k-1}^*$$

We then repeatedly apply the rule for random sampling, with the permutation on $\mathbb{Z}/p\mathbb{Z}$ that exchanges (v_i^*, w_i^*).

6 Conditional independence

We also consider conditional independence. Recall that the conditional probability $\mathsf{Pr}_{x\sim\mu}\left[A \mid B\right]$ is defined when $\mathsf{Pr}_{x\sim\mu}\left[B\right] \neq 0$ and satisfies $\mathsf{Pr}_{x\sim\mu}\left[A \mid B\right] \triangleq \frac{\mathsf{Pr}_{x\sim\mu}[A\wedge B]}{\mathsf{Pr}_{x\sim\mu}[B]}$.

Definition 14. *Let $X = \{x_1, \ldots, x_n\}$ be a set of program variables of types $A_1, \ldots, A_n$ and let E be an event. We say that X probabilistically independent conditionally on E w.r.t. a distribution $\mu \in \mathbb{D}(\mathbf{State})$ iff for every $(a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$:*

$$\mathsf{Pr}_\mu\left[\bigwedge_{1\le i\le n} x_i = a_i \middle| E\right] = \prod_{1\le i\le n} \mathsf{Pr}_\mu\left[x_i = a_i \mid E\right].$$

In the above definition, we are implicitly assuming that $\mathsf{Pr}_\mu\left[E\right] \neq 0$.

The following lemma unfolds the definition of conditional independence and is useful for the characterization of the next section.

[1] Let the Vandermonde matrix $V(1, \ldots, k-1) = \left(i^{j-1}\right)_{i,j}$ be

$$V(1, \ldots, k-1) \cdot (a_0, \ldots, a_{k-1})^T = (v_0, \ldots, v_{k-1})^T.$$

The system of equations has a unique solution if and only if the matrix $V(1, \ldots, k-1)$ is invertible in the space of matrices over $\mathbb{Z}/_{n\mathbb{Z}}$, which happens if and only if its determinant is non-zero mod n. Expanding,

$$\det(V(1, \ldots, k-1) = \prod_{i\neq j}(i-j) = \prod_{i=2}^{k-1} i!$$

Note that p does not divide the determinant by Gauss' lemma, since n can't divide any of the terms $i!$ for any $i < n$. Therefore, the system of equations has a unique solution.

Lemma 15. *A set of variables X is conditionally independent on an event E w.r.t. μ iff for every $a_1 \in A_1, \ldots, a_n \in A_n$:*

$$\mathsf{Pr}_{\mu}\left[\bigwedge_{1 \le i \le n} x_i = a_i \wedge E\right] \cdot (\mathsf{Pr}_{\mu}[E])^{n-1} = \prod_{1 \le i \le n} \mathsf{Pr}_{\mu}[x_i = a_i \wedge E]$$

6.1 Characterization

The characterization of independence based on self-composition can be extended as follows.

Proposition 16 (Conditional independence by coupling)**.** *The following are equivalent:*

1. *for every state m, X is independent conditionally on E w.r.t. $[\![s]\!]_m$;*
2. $\vDash s^{\langle n \rangle} \sim s^{\langle n \rangle} \ : \ \mathsf{EqMem}^{\langle n \rangle, \langle n \rangle} \implies (\phi_1 \wedge \mathbb{E}\langle 1 \rangle) \iff (\phi_2 \wedge \mathbb{E}\langle 2 \rangle)$ *where* $\mathbb{E} \triangleq \bigwedge_{1 \le i \le n} E^i$, $\phi_1 \triangleq \bigwedge_{1 \le i \le n} (x_i^1\langle 1 \rangle = a_i)$ *and* $\phi_2 \triangleq \bigwedge_{1 \le i \le n} (x_i^i\langle 2 \rangle = a_i)$.

Proof. The proof is similar to the case of independence. □

6.2 Example: conditional independence

```
$x \xleftarrow{\$} \mu$;
$y \xleftarrow{\$} \mu'$;
$z \xleftarrow{\$} \mu''$;
$w \leftarrow f(x, y)$;
$w' \leftarrow g(y, z)$;
```

Figure 9: Conditional indep.

We consider a simple example often used to illustrate Bayesian networks models. Let y, w, and w' be random variables. y is sampled from a fixed distribution μ. Both w and w' depend on y, along with independent sources of randomness, respectively x and z. While w and w' are not independent—they share dependence on y—if we *condition* on a particular value of y, then w and w' are independent.

The code of the corresponding program s is given in Figure 9. We want to show that w and w' are independent conditioned on $y = c$ for every c. Using our characterization based on self-composition, it amounts to proving the following (universally quantified) pRHL judgment:

$$\vDash s^{\langle 2 \rangle} \sim s^{\langle 2 \rangle} \ : \ \mathsf{EqMem}^{\langle 2 \rangle, \langle 2 \rangle} \implies \phi\langle 1 \rangle \iff \psi\langle 2 \rangle$$

where $\begin{cases} \phi \triangleq w^1 = a \wedge {w'}^1 = b \wedge y^1 = c \wedge y^2 = c \\ \psi \triangleq w^1 = a \wedge {w'}^2 = b \wedge y^1 = c \wedge y^2 = c. \end{cases}$

The proof proceeds by moving the samplings of z^1 and z^2 in both programs to the front of the program, and then swapping samplings in the left program only (we can use the rule [Swap] to reorder the instructions). Then, we couple $z^1\langle 1 \rangle$ to be equal to $z^2\langle 2 \rangle$, and $z^2\langle 1 \rangle$ to be equal to $z^1\langle 2 \rangle$. We apply the identity coupling to all other random samplings.

7 Conclusion

We have proposed a new lightweight method based on probabilistic couplings for proving uniformity and independence properties of probabilistic programs. An interesting direction for future work would be to combine our method with recent work for proving accuracy bounds for deferentially private computations [5]. The combined techniques could verify more precise accuracy bounds, by using independence and concentration bounds.

References

[1] G. Barthe, B. Grégoire, and S. Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Savannah, Georgia*, pages 90–101, New York, 2009. URL http://certicrypt.gforge.inria.fr/2013.Journal.pdf.

[2] G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21(06):1207–1252, 2011. URL http://www-sop.inria.fr/lemme/Tamara.Rezk/publication/Barthe-DArgenio-Rezk-Journal.pdf.

[3] G. Barthe, T. Espitau, B. Grégoire, J. Hsu, L. Stefanesco, and P.-Y. Strub. Relational reasoning via probabilistic coupling. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Suva, Fiji*, volume 9450 of *Lecture Notes in Computer Science*, pages 387–401. Springer-Verlag, 2015. URL http://arxiv.org/abs/1509.03476.

[4] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. Proving differential privacy via probabilistic couplings. In *IEEE Symposium on Logic in Computer Science (LICS), New York, New York*, pages 749–758, 2016.

[5] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. A program logic for union bounds. In *International Colloquium on Automata, Languages and Programming (ICALP), Rome, Italy*, volume 55 of *Leibniz International Proceedings in Informatics*, pages 107:1–107:15. Schloss Dagstuhl–Leibniz Center for Informatics, 2016. doi: 10.4230/LIPIcs.ICALP.2016.107. URL http://dx.doi.org/10.4230/LIPIcs.ICALP.2016.107.

[6] G. Barthe, B. Grégoire, J. Hsu, and P.-Y. Strub. Coupling proofs are probabilistic product programs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*, 2017. URL http://arxiv.org/abs/1607.03455.

[7] A. Z. Broder. Generating random spanning trees. In *ACM SIGACT Symposium on Theory of Computing (STOC), Seattle, Washington*, pages 442–447, 1989. doi: 10.1109/SFCS.1989.63516. URL http://dx.doi.org/10.1109/SFCS.1989.63516.

[8] R. Chadha, L. Cruz-Filipe, P. Mateus, and A. Sernadas. Reasoning about probabilistic sequential programs. *Theoretical Computer Science*, 379(1–2):142–165, 2007.

[9] K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. 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*, volume abs/1510.08517, 2016. URL http://arxiv.org/abs/1510.08517.

[10] K. Chatterjee, P. Novotný, and D. Zikelic. Stochastic invariants for probabilistic termination. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*, volume abs/1611.01063, 2017. URL http://arxiv.org/abs/1611.01063.

[11] Á. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In *Security in Pervasive Computing (SPC), Boppard, Germany*, volume 3450 of *Lecture Notes in Computer Science*, pages 193–209. Springer-Verlag, 2005.

[12] J. den Hartog. *Probabilistic extensions of semantical models.* PhD thesis, Vrije Universiteit Amsterdam, 2002.

[13] L. M. Ferrer Fioriti and H. Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. volume 50, pages 489–501, 2015.

[14] D. Kozen. A probabilistic PDL. *J. Comput. Syst. Sci.*, 30(2):162–178, 1985.

[15] T. Lindvall. *Lectures on the coupling method.* Courier Corporation, 2002.

[16] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.

[17] J. Pearl and A. Paz. Graphoids: Graph-based logic for reasoning about relevance relations or when would x tell you more about y if you already know z? In *ECAI*, pages 357–363, 1986.

[18] L. H. Ramshaw. *Formalizing the Analysis of Algorithms.* PhD thesis, Computer Science, 1979.

[19] R. Rand and S. Zdancewic. VPHL: A Verified Partial-Correctness Logic for Probabilistic Programs. In *Conference on the Mathematical Foundations of Programming Semantics (MFPS), Nijmegen, The Netherlands*, 2015.

[20] H. Thorisson. *Coupling, Stationarity, and Regeneration.* Springer-Verlag, 2000.

[21] C. Villani. *Optimal transport: old and new.* Springer-Verlag, 2008.

A Further example: rejection sampling

```
$b \leftarrow 0$;
while $\neg b$ do
  $x \xleftarrow{\$} \mathcal{U}_A$;
  $b \leftarrow P\ x$;
```

Figure 10: Rejection sampling

This example is a classic randomized algorithm: given a uniform distribution $\mathcal{U}_A$ over some finite type A, and some non-empty predicate P, its goal is to output a uniformly sampled value that satisfies P. This is formalized by the program from Figure 10. Using a straightforward extension of our characterization 9, it suffices to prove that the program outputs a value that satisifes P (which can be done by reasoning directly on the semantics of the program or using a logic for sure events) and the (universally quantified) pRHL judgment:

$$\forall a_1, a_2 \in A.\ \vDash s \sim s\ :\ P\ a_1 \wedge P\ a_2 \Longrightarrow\ x\langle 1\rangle = a_1 \iff x\langle 2\rangle = a_2$$

The proof of the pRHL judgment is particularly simple; one simply needs to choose as loop invariant $b\langle 1\rangle = b\langle 2\rangle \wedge x\langle 1\rangle = a_1 \iff x\langle 2\rangle = a_2$ and use in the rule for random sampling the permutation π_{a_1,a_2} such that $\pi_{a_1,a_2}(a_1) = a_2$, $\pi_{a_1,a_2}(a_2) = a_1$, and $\pi_{a_1,a_2}(x) = x$ for every $x \notin \{a_1, a_2\}$.