5

Probabilistic Couplings from Program Logics

Gilles Barthe
MPI For Security and Privacy, Bochum & IMDEA Software Institute, Madrid
Justin Hsu
University of Wisconsin–Madison

Abstract: *Proof by coupling* is a powerful technique for proving properties about pairs of probabilistic processes. Originally developed by probability theorists, this proof technique has recently found surprising applications in formal verification, enabling clean proofs of probabilistic relational properties. We show that the probabilistic program logic pRHL is a formal logic for proofs by coupling, with logical proof rules capturing reasoning steps in traditional coupling proofs. This connection gives a new method to formally verify probabilistic properties.

5.1 Introduction

Formal verification of probabilistic programs is an active area of research which aims to reason about safety and liveness properties of probabilistic computations. Many important properties for probabilistic programs are naturally expressed in terms of two program executions; for this reason, such properties are called *relational*. While there exist established approaches to verify relational properties of deterministic programs, reasoning about relational properties of probabilistic programs is more challenging. In this chapter we explore a powerful method called *proof by coupling*. This technique—originally developed in probability theory for analyzing Markov Chains—is surprisingly useful for establishing a broad range of relational properties, including:

- **probabilistic equivalence** (also **differential privacy**): two programs produce distributions that are equivalent or suitably close from an observer's point of view. For instance, *differential privacy* requires that two similar inputs—say, the private database and a hypothetical version with one individual's data omitted—yield closely related output distributions;

[a] From *Foundations of Probabilistic Programming*, edited by Gilles Barthe, Joost-Pieter Katoen and Alexandra Silva published 2020 by Cambridge University Press.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

- **stochastic domination**: one probabilistic program is more likely than another to produce large outputs;
- **convergence** (also **mixing**): the output distributions of two probabilistic loops approach each other as the loops execute more iterations;
- **truthfulness** (also **Nash equilibrium**): an agent's average utility is larger when reporting an honest value instead of deviating to a misleading value.

At first glance, relational properties appear to be even harder to establish than standard, non-relational properties—instead of analyzing a single probabilistic computation, we now need to deal with two. (Indeed, any property of a single program can be viewed as a relational property between the target program and the trivial, do-nothing program.) However, relational properties often relate two highly similar programs, even comparing the same program on two different inputs. In these cases, we can leverage a powerful abstraction and an associated proof technique from probability theory—*probabilistic coupling* and *proof by coupling*.

The fundamental observation is that probabilistic relational properties compare computations in two different worlds, assuming no particular correlation between random samples. Accordingly, we may freely assume any correlation we like for the purposes of the proof—a relational property holds (or does not hold) regardless of which one we pick. For instance, if two programs generate identical output distributions, this holds whether they share coin flips or take independent samples; relational properties do not require that the two programs use separate randomness. By carefully arranging the correlation, we can reason about two executions as if they were linked in some convenient way.

To take advantage of this freedom, we need some way to design specific correlations between program executions. In principle, this can be a highly challenging task. The two runs may take samples from different distributions, and it is unclear exactly how they can or should share randomness. When the two programs have similar shapes, however, we can link two computations in a step-by-step fashion. First, correlations between intermediate samples can be described by *probabilistic couplings*, joint distributions over pairs. For example, a valid coupling of two fair coin flips could specify that the draws take opposite values; the correlated distribution would produce "(heads, tails)" and "(tails, heads)" with equal probability. A coupling formalizes what it means to share randomness: a single source of randomness simulates draws from two distributions. Since randomness can be shared in different ways, two distributions typically support a variety of distinct couplings.

A *proof by coupling*, then, describes two correlated executions by piecing together couplings for corresponding pairs of sampling instructions. In the course of a proof, we can imagine stepping through the two programs in parallel, selecting couplings along the way. For instance, if we apply the opposite coupling to link a coin flip in one

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

program with a coin flip in the other, we may assume the samples remain opposite when analyzing the rest of the programs. By flowing these relations forward from two initial inputs, a proof by coupling can focus on just pairs of similar executions as it builds up to a coupling between two output distributions. This is the main product of the proof: features of the final coupling imply properties about the output distributions, and hence relational properties about the original programs.

Working in tandem, couplings and proofs by couplings can significantly simplify probabilistic reasoning in several ways.

- **Reduce to one source of randomness.** By analyzing two runs as if they shared a single source of randomness, we can reason about two programs as if they were one.
- **Abstract away probabilities.** Proofs by coupling isolate probabilistic reasoning from the non-probabilistic parts of the proof, which are more straightforward. We only need to think about probabilistic aspects when we select couplings at the sampling instructions; throughout the rest of the programs, we can reason purely in terms of deterministic relations between the two runs.
- **Enable compositional, structured reasoning.** By focusing on each step of an algorithm individually and then smoothly combining the results, the coupling proof technique enables a highly modular style of reasoning guided by the code of the program.

Proofs by coupling are also surprisingly flexible—many probabilistic relational properties, including the examples listed above, can be proved in this style. Individual couplings can also be combined in subtle ways, giving rise to a rich diversity of coupling proofs.

After reviewing probability theory basics (Section 5.2), we introduce probabilistic couplings and their key properties (Section 5.3). Then, we present intuition behind proof by coupling (Section 5.4). To formalize these arguments, we draw a connection to the program logic pRHL (Barthe et al., 2009). Proofs in the pRHL are formal proofs by coupling: valid judgments imply the existence of a coupling, and logical rules describing how to combine couplings to construct new couplings (Section 5.5). We demonstrate several examples of coupling proofs in the logic (Section 5.6), and conclude by briefly discussing related lines of work (Section 5.7).

Bibliographic note. This chapter is an updated and expanded version of the first two chapters of the second author's PhD thesis (Hsu, 2017).

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

5.2 Preliminaries

A discrete probability distribution associates each element of a set with a number in the unit interval $[0, 1]$, representing its *probability*. In order to model programs that may not terminate, we work with a slightly more general notion called a *sub-distribution*.

Definition 5.1 A (discrete) *sub-distribution* over a countable set $\mathcal{A}$ is a map $\mu : \mathcal{A} \to [0, 1]$ taking each element of $\mathcal{A}$ to a numeric weight such that the weights sum to at most 1:

$$\sum_{a \in \mathcal{A}} \mu(a) \leq 1.$$

We write $\mathbf{SDistr}(\mathcal{A})$ for the set of all sub-distributions over $\mathcal{A}$. When the weights sum to 1, we call μ a *proper* distribution; we write $\mathbf{Distr}(\mathcal{A})$ for the set of all proper distributions over $\mathcal{A}$. The *empty* or *null sub-distribution* $\perp$ assigns weight 0 to all elements.

We work with discrete sub-distributions throughout. While this is certainly a restriction—excluding, for instance, some well-known distributions over the real numbers—many interesting coupling proofs can already be expressed in our setting. Our results should mostly carry over to the continuous setting, as couplings are frequently used on continuous distributions in probability theory, but the general case introduces measure-theoretic technicalities (e.g., working with integrals rather than sums, checking sets are measurable, etc.) that would distract from our primary focus.

We need several concepts and notations related to discrete distributions. First, the probability of a set $\mathcal{S} \subseteq \mathcal{A}$ is given by a sum:

$$\mu(\mathcal{S}) \triangleq \sum_{a \in \mathcal{S}} \mu(a).$$

The *support* of a sub-distribution is the set of elements with positive probability:

$$\mathrm{supp}(\mu) \triangleq \{a \in \mathcal{A} \mid \mu(a) > 0\}.$$

The *weight* of a sub-distribution is the total probability of all elements:

$$|\mu| \triangleq \sum_{a \in \mathcal{A}} \mu(a).$$

Finally, the *expected value* of a real-valued function $f : \mathcal{A} \to \mathbb{R}$ over a sub-distribution μ is

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

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

Under light assumptions, the expected value is guaranteed to exist (for instance, when f is a bounded function).

To transform sub-distributions, we can lift a function $f : \mathcal{A} \to \mathcal{B}$ on sets to a map $f^\sharp : \mathbf{SDistr}(\mathcal{A}) \to \mathbf{SDistr}(\mathcal{B})$ via $f^\sharp(\mu)(b) \triangleq \mu(f^{-1}(b))$. For example, let $p_1 : \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{A}_1$ and $p_2 : \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{A}_2$ be the first and second projections from a pair. The corresponding *probabilistic projections* $\pi_1 : \mathbf{SDistr}(\mathcal{A}_1 \times \mathcal{A}_2) \to \mathbf{SDistr}(\mathcal{A}_1)$ and $\pi_2 : \mathbf{SDistr}(\mathcal{A}_1 \times \mathcal{A}_2) \to \mathbf{SDistr}(\mathcal{A}_2)$ are defined by

$$\pi_1(\mu)(a_1) \triangleq p_1^\sharp(\mu)(a_1) = \sum_{a_2 \in \mathcal{A}_2} \mu(a_1, a_2)$$
$$\pi_2(\mu)(a_2) \triangleq p_2^\sharp(\mu)(a_2) = \sum_{a_1 \in \mathcal{A}_1} \mu(a_1, a_2).$$

We call a sub-distribution μ over pairs a *joint sub-distribution*, and the projected sub-distributions $\pi_1(\mu)$ and $\pi_2(\mu)$ the *first* and *second marginals*, respectively.

5.3 Couplings and liftings: definitions and basic properties

A probabilistic coupling models two distributions with a single joint distribution.

Definition 5.2 Given μ_1, μ_2 sub-distributions over $\mathcal{A}_1$ and $\mathcal{A}_2$, a sub-distribution μ over pairs $\mathcal{A}_1 \times \mathcal{A}_2$ is a *coupling* for (μ_1, μ_2) if $\pi_1(\mu) = \mu_1$ and $\pi_2(\mu) = \mu_2$.

Generally, couplings are not unique—different witnesses represent different ways to share randomness between two distributions. To give a few examples, we first introduce some standard distributions.

Definition 5.3 Let $\mathcal{A}$ be a finite, non-empty set. The *uniform distribution* over $\mathcal{A}$, written $\mathbf{Unif}(\mathcal{A})$, assigns probability $1/|\mathcal{A}|$ to each element. We write **Flip** for the uniform distribution over the set $\{0, 1\}$. This can be viewed as the distribution of a fair coin flip.

Example 5.4 (Couplings from bijections) We can give two distinct couplings of $(\mathbf{Flip}, \mathbf{Flip})$:

Identity coupling:

$$\mu_{\mathrm{id}}(a_1, a_2) \triangleq \begin{cases} 1/2 & : a_1 = a_2 \\ 0 & : \text{otherwise.} \end{cases}$$

Negation coupling:

$$\mu_{\neg}(a_1, a_2) \triangleq \begin{cases} 1/2 & : a_1 = 1 - a_2 \\ 0 & : \text{otherwise.} \end{cases}$$

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

More generally, any bijection $f : \mathcal{A} \to \mathcal{A}$ yields a coupling of $(\mathbf{Unif}(\mathcal{A}), \mathbf{Unif}(\mathcal{A}))$:

$$\mu_f(a_1, a_2) \triangleq \begin{cases} 1/|\mathcal{A}| & : f(a_1) = a_2 \\ 0 & : \text{otherwise.} \end{cases}$$

This coupling matches samples: each sample a from the first distribution is paired with a corresponding sample $f(a)$ from the second distribution. To take two correlated samples from this coupling, we can imagine first sampling from the first distribution, and then applying f to produce a sample for the second distribution. When f is a bijection, this gives a valid coupling for two uniform distributions: viewed separately, both the first and second correlated samples are distributed uniformly.

For more general distributions, if a_1 and a_2 have different probabilities under μ_1 and μ_2 then the correlated distribution cannot return $(a_1, -)$ and $(-, a_2)$ with equal probabilities; for instance, a bijection with $f(a_1) = a_2$ would not give a valid coupling. However, general distributions can be coupled in other ways.

Example 5.5 Let μ be a sub-distribution over $\mathcal{A}$. The *identity coupling* of (μ, μ) is

$$\mu_{\text{id}}(a_1, a_2) \triangleq \begin{cases} \mu(a) & : a_1 = a_2 = a \\ 0 & : \text{otherwise.} \end{cases}$$

Sampling from this coupling yields a pair of equal values.

Example 5.6 Let μ_1, μ_2 be sub-distributions over $\mathcal{A}_1$ and $\mathcal{A}_2$. The *independent* or *trivial* coupling is

$$\mu_\times(a_1, a_2) \triangleq \mu_1(a_1) \cdot \mu_2(a_2).$$

This coupling models μ_1 and μ_2 as independent distributions: sampling from this coupling is equivalent to first sampling from μ_1 and then pairing with an independent draw from μ_2. The coupled distributions must be proper in order to ensure the marginal conditions.

Since any two proper distributions can be coupled by the trivial coupling, the mere existence of a coupling yields little information. Couplings are more useful when the joint distribution satisfies additional conditions, for instance when all elements in the support satisfy some property.

Definition 5.7 (Lifting) Let μ_1, μ_2 be sub-distributions over $\mathcal{A}_1$ and $\mathcal{A}_2$, and let $\mathcal{R} \subseteq \mathcal{A}_1 \times \mathcal{A}_2$ be a relation. A sub-distribution μ over pairs $\mathcal{A}_1 \times \mathcal{A}_2$ is a *witness* for the $\mathcal{R}$-*lifting* of (μ_1, μ_2) if:
(i) μ is a coupling for (μ_1, μ_2), and
(ii) $\text{supp}(\mu) \subseteq \mathcal{R}$.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

If there exists μ satisfying these two conditions, we say μ_1 and μ_2 are related by the *lifting of* $\mathcal{R}$ and write

$$\mu_1 \; \mathcal{R}^\sharp \; \mu_2.$$

We typically express $\mathcal{R}$ using set notation, i.e.,

$$\mathcal{R} = \{(a_1, a_2) \in \mathcal{A}_1 \times \mathcal{A}_2 \mid \Phi(a_1, a_2)\}$$

where Φ is some logical formula. When the sets $\mathcal{A}_1$ and $\mathcal{A}_2$ are clear from the context, we leave them implicit and just write Φ, sometimes enclosed by parentheses (Φ) for clarity.

Example 5.8 Many of the couplings we saw before are more precisely described as liftings.

Bijection coupling. For a bijection $f : \mathcal{A} \to \mathcal{A}$, the coupling in Theorem 5.4 witnesses the lifting

$$\mathbf{Unif}(\mathcal{A}) \; G_f^\sharp \; \mathbf{Unif}(\mathcal{A}).$$

where the relation $G_f \triangleq \{(a_1, a_2) \mid f(a_1) = a_2\}$ models the graph of f.

Identity coupling. The coupling in Theorem 5.5 witnesses the lifting

$$\mu \; (=)^\sharp \; \mu.$$

Trivial coupling. The coupling in Theorem 5.6 witnesses the lifting

$$\mu_1 \; \top^\sharp \; \mu_2,$$

where $\top \triangleq \mathcal{A}_1 \times \mathcal{A}_2$ is the trivial relation relating all pairs of elements.

5.3.1 Useful consequences of couplings and liftings

A coupling μ between (μ_1, μ_2) can be used for proving probabilistic properties about μ_1 and μ_2. Surprisingly, many properties already follow from the *existence* of a lifting from some relation R—no analysis of the coupling distribution μ is required. First of all, two coupled distributions have equal weight.

Proposition 5.9 (Equality of weight) *Suppose μ_1 and μ_2 are sub-distributions over $\mathcal{A}$ such that there exists a coupling μ of μ_1 and μ_2. Then $|\mu_1| = |\mu_2|$.*

This follows because μ_1 and μ_2 are both projections of μ, and projections preserve weight. Couplings can also show that two distributions are equal.

Proposition 5.10 (Equality of distributions) *Suppose μ_1 and μ_2 are two sub-distributions over $\mathcal{A}$. Then $\mu_1 = \mu_2$ if and only if $\mu_1 \; (=)^\sharp \; \mu_2$.*

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

Proof For the forward direction, define $\mu(a,a) \triangleq \mu_1(a) = \mu_2(a)$ and $\mu(a_1,a_2) \triangleq 0$ otherwise. Evidently, μ has support in the equality relation (=) and also has the desired marginals: $\pi_1(\mu) = \mu_1$ and $\pi_2(\mu) = \mu_2$. Thus μ is a witness to the desired lifting.

For the reverse direction, let the witness be μ. By the support condition, $\pi_1(\mu)(a) = \pi_2(\mu)(a)$ for every $a \in \mathcal{A}$. Since the left and right sides are equal to $\mu_1(a)$ and $\mu_2(a)$ respectively by the marginal conditions, $\mu_1(a) = \mu_2(a)$ for every a. So, μ_1 and μ_2 are equal. □

In some cases we can show results in the converse direction: if a property of two distributions holds, then there exists a particular lifting. To give some examples, we first introduce a powerful equivalence due to Strassen (1965).

Theorem 5.11 *Let μ_1, μ_2 be sub-distributions over $\mathcal{A}_1$ and $\mathcal{A}_2$, and let $\mathcal{R}$ be a binary relation over $\mathcal{A}_1$ and $\mathcal{A}_2$. Then $\mu_1 \ \mathcal{R}^\sharp \ \mu_2$ implies $\mu_1(\mathcal{S}_1) \leq \mu_2(\mathcal{R}(\mathcal{S}_1))$ for every subset $\mathcal{S}_1 \subseteq \mathcal{A}_1$, where $\mathcal{R}(\mathcal{S}_1) \subseteq \mathcal{A}_2$ is the* image *of $\mathcal{S}_1$ under $\mathcal{R}$:*

$$\mathcal{R}(\mathcal{S}_1) \triangleq \{a_2 \in \mathcal{A}_2 \mid \exists a_1 \in \mathcal{A}_1,\ (a_1,a_2) \in \mathcal{R}\}.$$

(For instance, if $\mathcal{A}_1 = \mathcal{A}_2 = \mathbb{N}$ and $\mathcal{R}$ is the relation $\leq$, then $\mathcal{R}(\mathcal{S})$ is the set of all natural numbers larger than $\min \mathcal{S}$*.) The converse holds if μ_1 and μ_2 have equal weight.*

Strassen proved Theorem 5.11 for continuous (proper) distributions using deep results from probability theory. In our discrete setting, there is an elementary proof by the maximum flow-minimum cut theorem (see, e.g., (Kleinberg and Tardos, 2005)). For now, we use this theorem to illustrate a few more useful consequences of liftings. First, couplings can bound the probability of an event in the first distribution by the probability of an event in the second distribution.

Proposition 5.12 *Suppose μ_1, μ_2 are sub-distributions over $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, and consider two subsets $\mathcal{S}_1 \subseteq \mathcal{A}_1$ and $\mathcal{S}_2 \subseteq \mathcal{A}_2$. Then,*

$$\mu_1 \ \{(a_1,a_2) \mid a_1 \in \mathcal{S}_1 \rightarrow a_2 \in \mathcal{S}_2\}^\sharp \ \mu_2$$

implies $\mu_1(\mathcal{S}_1) \leq \mu_2(\mathcal{S}_2)$. The converse holds when μ_1 and μ_2 have equal weight.

Proof Let $\mathcal{R}$ be the relation $\{(a_1,a_2) \mid a_1 \in \mathcal{S}_1 \rightarrow a_2 \in \mathcal{S}_2\}$. The forward direction is immediate by Theorem 5.11, taking the subset $\mathcal{S}_1$. For the reverse direction, consider any non-empty subset $\mathcal{T}_1 \subseteq \mathcal{A}_1$. If $\mathcal{T}_1$ is not contained in $\mathcal{S}_1$, then $\mathcal{R}(\mathcal{T}_1) = \mathcal{A}_2$ and $\mu_1(\mathcal{T}_1) \leq \mu_2(\mathcal{R}(\mathcal{T}_1))$ since μ_1 and μ_2 have equal weight. Otherwise $\mathcal{R}(\mathcal{T}_1) = \mathcal{S}_2$, so

$$\mu_1(\mathcal{T}_1) \leq \mu_1(\mathcal{S}_1) \leq \mu_2(\mathcal{S}_2) = \mu_2(\mathcal{R}(\mathcal{T}_1)).$$

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

Theorem 5.11 gives the desired lifting:

$$\mu_1 \{(a_1, a_2) \mid a_1 \in \mathcal{S}_1 \to a_2 \in \mathcal{S}_2\}^\sharp \mu_2. \qquad \square$$

A slightly more subtle consequence is *stochastic domination*, an order on distributions over an ordered set.

Definition 5.13 Let $(\mathcal{A}, \leq_{\mathcal{A}})$ be a partially ordered set. For every $k \in \mathcal{A}$, let $k\uparrow \triangleq \{a \in \mathcal{A} \mid k \leq_{\mathcal{A}} a\}$. Suppose μ_1, μ_2 are sub-distributions over $\mathcal{A}$. We say μ_2 *stochastically dominates* μ_1, denoted $\mu_1 \leq_{sd} \mu_2$, if

$$\mu_1(k\uparrow) \leq \mu_2(k\uparrow)$$

for every $k \in \mathcal{A}$.

For an example of stochastic domination, take distributions over the natural numbers $\mathbb{N}$ with the usual order and μ_1 places weight 1 on 0 while μ_2 places weight 1 on 1.

Stochastic domination is precisely the probabilistic lifting of the order relation.

Proposition 5.14 *Suppose μ_1, μ_2 are sub-distributions over a set $\mathcal{A}$ with a partial order $\leq_{\mathcal{A}}$. Then $\mu_1 \, (\leq_{\mathcal{A}})^\sharp \, \mu_2$ implies $\mu_1 \leq_{sd} \mu_2$. The converse also holds when μ_1 and μ_2 have equal weight, as long as the upwards closed subsets of $\mathcal{A}$ are $\emptyset$, $\mathcal{A}$ and $k\uparrow$ with $k \in \mathcal{A}$ (e.g., $\mathcal{A} = \mathbb{N}$ or $\mathbb{Z}$ with the usual order).*

Proof Let $\mathcal{R} \triangleq (\leq_{\mathcal{A}})$. For the forward direction, Theorem 5.11 gives

$$\mu_1(k\uparrow) \leq \mu_2(\mathcal{R}(k\uparrow)) = \mu_2(k\uparrow).$$

This holds for all $k \in \mathcal{A}$, establishing $\mu_1 \leq_{sd} \mu_2$.

For the converse, suppose $\mu_1 \leq_{sd} \mu_2$ and μ_1 and μ_2 have equal weights, and let $\mathcal{S} \subseteq \mathcal{A}$ be any subset. Note that $\mathcal{R}(\mathcal{S})$ is upwards closed so we proceed by case analysis on $\mathcal{R}(\mathcal{S})$. If $\mathcal{R}(\mathcal{S}) = \emptyset$, then $\mathcal{S}$ is also empty and $\mu_1(\mathcal{S}) \leq \mu_2(\mathcal{R}(\mathcal{S}))$. If $\mathcal{R}(\mathcal{S}) = \mathcal{A}$, then $\mu_1(\mathcal{S}) \leq \mu_2(\mathcal{R}(\mathcal{S}))$ since μ_1 and μ_2 have equal weights. Finally, if $\mathcal{R}(\mathcal{S})$ is $k\uparrow$, and we have

$$\mu_1(\mathcal{S}) \leq \mu_1(\mathcal{R}(\mathcal{S})) = \mu_1(k\uparrow) \leq \mu_2(k\uparrow) = \mu_2(\mathcal{R}(\mathcal{S})),$$

where the middle inequality is by stochastic domination. Theorem 5.11 implies $\mu_1 \, (\leq_{\mathcal{A}})^\sharp \, \mu_2$. $\square$

Finally, a typical application of coupling proofs is showing that two distributions are close together.

Definition 5.15 Let μ_1, μ_2 be sub-distributions over $\mathcal{A}$. The *total variation distance*

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

(also known as *TV-distance* or *statistical distance*) between μ_1 and μ_2 is defined as

$$d_{\mathrm{tv}}(\mu_1, \mu_2) \triangleq \frac{1}{2} \sum_{a \in \mathcal{A}} |\mu_1(a) - \mu_2(a)| = \max_{S \subseteq \mathcal{A}} |\mu_1(S) - \mu_2(S)|.$$

In particular, the total variation distance bounds the difference in probabilities of any event.

Couplings are closely related to TV-distance, as captured by the following theorem. Theorem 5.16 is the fundamental result behind the so-called *coupling method* (Aldous, 1983), a technique to show two probabilistic processes converge by constructing a coupling that causes the processes to become equal with high probability. Unlike the previous facts, the target property about μ_1 and μ_2 does not directly follow from the existence of a lifting—we need more detailed information about the coupling μ.

Theorem 5.16 (see, e.g., Lindvall (2002); Levin et al. (2009)) *Let μ_1 and μ_2 be sub-distributions over $\mathcal{A}$ and let μ be a coupling. Then*

$$d_{tv}(\mu_1, \mu_2) \leq \Pr_{(a_1,a_2)\sim\mu}[a_1 \neq a_2].$$

In particular, if $S \subseteq \mathcal{A} \times \mathcal{A}$ and μ witnesses

$$\mu_1 \{(a_1, a_2) \in \mathcal{A} \times \mathcal{A} \mid (a_1, a_2) \in S \rightarrow a_1 = a_2\}^\sharp \mu_2,$$

then their TV-distance is bounded by the probability of the complement of S w.r.t. μ:

$$d_{tv}(\mu_1, \mu_2) \leq \Pr_{(a_1,a_2)\sim\mu}[(a_1, a_2) \notin S].$$

Moreover, there exists a coupling μ, called maximal coupling*, such that*

$$d_{tv}(\mu_1, \mu_2) = \Pr_{(a_1,a_2)\sim\mu}[a_1 \neq a_2].$$

Proof We only prove the inequality. Let μ be a coupling of μ_1 and μ_2. We have:

$$\begin{aligned}
&d_{\mathrm{tv}}(\mu_1, \mu_2) \\
&= \max_P \left| \Pr_{a_1\sim\mu_1}[a_1 \in P] - \Pr_{a_2\sim\mu_2}[a_2 \in P] \right| \\
&= \max_P \left| \Pr_{(a_1,a_2)\sim\mu}[a_1 \in P] - \Pr_{(a_1,a_2)\sim\mu}[a_2 \in P] \right| \\
&= \max_P \Bigg| \Pr_{(a_1,a_2)\sim\mu}[a_1 \in P \wedge a_1 = a_2] + \Pr_{(a_1,a_2)\sim\mu}[a_1 \in P \wedge a_1 \neq a_2] \\
&\qquad - \Pr_{(a_1,a_2)\sim\mu}[a_2 \in P \wedge a_1 = a_2] - \Pr_{(a_1,a_2)\sim\mu}[a_2 \in P \wedge a_1 \neq a_2] \Bigg|
\end{aligned}$$

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

$$
\begin{aligned}
&= \max_P \left| \Pr_{(a_1,a_2)\sim\mu}[a_1 \in P \wedge a_1 \neq a_2] - \Pr_{(a_1,a_2)\sim\mu}[a_2 \in P \wedge a_1 \neq a_2] \right| \\
&\leq \max_P(\max(\Pr_{(a_1,a_2)\sim\mu}[a_1 \in P \wedge a_1 \neq a_2], \Pr_{(a_1,a_2)\sim\mu}[a_2 \in P \wedge a_1 \neq a_2])) \\
&\leq \Pr_{(a_1,a_2)\sim\mu}[a_1 \neq a_2]
\end{aligned}
$$

The proof is similar when μ witnesses

$$\mu_1 \ \{(a_1, a_2) \in \mathcal{A} \times \mathcal{A} \mid (a_1, a_2) \in \mathcal{S} \rightarrow a_1 = a_2\}^\sharp \ \mu_2$$

by case analysis on $(a_1, a_2) \in S$ rather than $a_1 = a_2$. □

5.3.2 Composition properties

Couplings and liftings are closed under various notions of composition. Most important for our purposes will be sequential composition.

Theorem 5.17 *Let $\mu \in \mathbf{Distr}(\mathcal{A}_1 \times \mathcal{A}_2)$ witness $\mu_1 \ R^\sharp \ \mu_2$, where $\mu_1 \in \mathbf{Distr}(\mathcal{A}_1)$ and $\mu_2 \in \mathbf{Distr}(\mathcal{A}_2)$ and $R \subseteq \mathcal{A}_1 \times \mathcal{A}_2$. Let $M : \mathcal{A}_1 \times \mathcal{A}_2 \rightarrow \mathbf{Distr}(\mathcal{B}_1 \times \mathcal{B}_2)$ such that $M(a_1, a_2)$ witnesses for $M_1(a_1) \ S^\sharp \ M_2(a_2)$ for every $(a_1, a_2) \in R$. Then $\mathbf{bind}(\mu, M)$ witnesses*

$$\mathbf{bind}(\mu_1, M_1) \ S^\sharp \ \mathbf{bind}(\mu_2, M_2).$$

While this theorem may appear a bit cryptic at this point, it will play a key role in later developments—informally, this result enables us to build a coupling for a sequential composition of two processes by constructing a coupling for each piece.

5.3.3 Couplings and liftings for Markov chains

The previous results suggest an approach to proving properties of two distributions: demonstrate there exists a coupling of a particular form. This approach is indirect, but surprisingly fruitful, when employed to prove properties about probabilistic processes modelled as discrete-time Markov chains. Recall that a discrete-time Markov chain is given by a state space $\mathcal{A}$, which we assume to be discrete, by an initial distribution $\mu \in \mathbf{Distr}(\mathcal{A})$ and by a transition map $t : \mathcal{A} \rightarrow \mathbf{Distr}(\mathcal{A})$.

Couplings and $\mathcal{R}$-liftings naturally extend to (discrete-time) Markov chains.

Definition 5.18 A coupling between two Markov chains given by initial sub-distributions μ_1 and μ_2 and transition functions t_1 and t_2 is a Markov chain given by an initial sub-distributions μ and a joint transition function $t : (\mathcal{A} \times \mathcal{A}) \rightarrow \mathbf{Distr}(\mathcal{A} \times \mathcal{A})$ such that:

- μ is a coupling for μ_1 and μ_2;

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

- for every x_1 and x_2, $t(x_1, x_2)$ is a coupling for $t_1(x_1)$ and $t_2(x_2)$.

The notion of $\mathcal{R}$-lifting extends similarly.

Definition 5.19 Let $\mathcal{R} \subseteq \mathcal{A}_1 \times \mathcal{A}_2$ be a relation. A $\mathcal{R}$-lifting between two Markov chains given by initial sub-distributions μ_1 and μ_2 and transition functions t_1 and t_2 is a Markov chain given by an initial sub-distributions μ and a joint transition function $t : (\mathcal{A} \times \mathcal{A}) \to \mathbf{Distr}(\mathcal{A} \times \mathcal{A})$ such that:

- μ is a $\mathcal{R}$-lifting for μ_1 and μ_2;
- for every $(x_1, x_2) \in \mathcal{R}$, $t(x_1, x_2)$ is a $\mathcal{R}$-lifting for $t_1(x_1)$ and $t_2(x_2)$.

The definition of $\mathcal{R}$-lifting naturally extends to families of relations $(\mathcal{R}_i)_{i \in \mathbb{N}}$. In this case one requires that the sub-distributions obtained by iterating k times the transition functions on the initial sub-distributions are related by $\mathcal{R}_k$. Other definitions relax these conditions; for instance, in shift couplings the relation needs not be pointwise, i.e. one can relate the two processes at different steps k_1 and k_2.

5.4 Proof by coupling

Finding appropriate couplings requires ingenuity and is often the main intellectual challenge when carrying out a proof by coupling. Given a conjecture, how are we supposed to find a witness distribution with the desired properties? To address this challenge, probability theorists have developed a powerful proof technique called *proof by coupling*. We close this section with an informal explanation and an example of the proof technique in action.

Given two probabilistic processes, a proof by coupling builds a coupling for the output distributions by coupling intermediate samples. In a bit more detail, we imagine stepping through the processes in parallel, one step at a time, starting from two inputs, and decoupling the transition function into a random sampling and a deterministic computation. For the samplings, we pick a valid coupling for the sampled distributions. The selected couplings induce a relation on samples, which we can assume when analyzing the rest of the computation (i.e. the deterministic part). For instance, by selecting couplings for earlier samples carefully, we may be able to assume the samplings yield equal values.

Example 5.20 Consider a probabilistic process that tosses a fair coin T times and returns the number of heads. If μ_1, μ_2 are the output distributions from running this process for $T = T_1, T_2$ iterations respectively and $T_1 \leq T_2$, then $\mu_1 \leq_{sd} \mu_2$.

Proof by coupling For the first T_1 iterations, couple the coin flips to be equal—this ensures that after the first T_1 iterations, the coupled counts are equal. The remaining $T_2 - T_1$ coin flips in the second run can only increase the second count, while

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

preserving the first count. Therefore under the coupling, the first count is no more than the second count at termination, establishing $\mu_1 \leq_{sd} \mu_2$. □

For readers unfamiliar with these proofs, this argument may appear bewildering. The coupling is constructed implicitly, and some of the steps are mysterious. To clarify such proofs, a natural idea is to design a formal logic describing coupling proofs. Somewhat surprisingly, the logic we are looking for was already proposed in the formal verification literature, originally for verifying security of cryptographic constructions.

5.5 A formal logic for coupling proofs

We will work with the logic pRHL (probabilistic Relational Hoare Logic) proposed by Barthe et al. (2009). Before detailing its connection to coupling proofs, we provide a brief introduction to program logics.

5.5.1 Program logics: A brief primer

A *logic* consists of a collection of formulas, also known as *judgments*, and an interpretation describing what it means—in typical, standard mathematics—for judgments to be true (*valid*). While it is possible to prove judgments valid directly by using regular mathematical arguments, this is often inconvenient as the interpretation may be quite complicated. Instead, many logics provide a *proof system*, a set of *logical rules* describing how to combine known judgments (the *premises*) to prove a new judgment (the *conclusion*). Each rule represents a single step in a formal proof. Starting from judgments given by rules with no premises (*axioms*), we can successively apply rules to prove new judgments, building a tree-shaped *derivation* culminating in a single judgment. To ensure that this final judgment is valid, each logical rule should be *sound*: if the premises are valid, then so is the conclusion. Soundness is a basic property, typically one of the first results to be proved about a logic.

Program logics were first introduced by Hoare (1969), building on earlier ideas by Floyd (1967); they are also called *Floyd-Hoare logics*. These logics are really two logics in one: the *assertion logic*, where formulas describe program states, and the program logic proper, where judgments describe program behavior. A judgment in the program logic consists of three parts: a program c and two *assertions* Φ and Ψ from the assertion logic. The *pre-condition* Φ describes the initial conditions before executing c (for instance, assumptions about the input), while the *post-condition* Ψ describes the final conditions after executing c (for instance, properties of the output). Hoare (1969) proposed the original logical rules, which construct a judgment for a

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

program by combining judgments for its sub-programs. This compositional style of reasoning is a hallmark of program logics.

By varying the interpretation of judgments, the assertion logic, and the logical rules, Floyd-Hoare logics can establish a variety of properties about different kinds of imperative programs. Notable extensions reason about non-determinism (Dijkstra, 1976), pointers and memory allocation (O'Hearn et al., 2001; Reynolds, 2001, 2002), concurrency (O'Hearn, 2007), and more. (Readers should consult a survey for a more comprehensive account of Floyd-Hoare logic (Apt, 1981, 1983; Jones, 2003).)

In this tradition, Barthe et al. (2009) introduced the logic pRHL targeting security properties in cryptography. Compared to standard program logics, there are two twists: each judgment describes *two* programs,[1] and programs can use random sampling. In short, pRHL is a *probabilistic Relational Hoare Logic*. Judgments encode probabilistic relational properties of two programs, where a post-condition describes a probabilistic liftings between two output distributions. More importantly, the proof rules represent different ways to combine liftings, formalizing various steps in coupling proofs. Accordingly, we will interpret pRHL as a formal logic for proofs by coupling.

To build up to this connection, we first provide a brief overview of a core version of pRHL, reviewing the programming language, the judgments and their interpretation, and the logical rules.

5.5.2 The logic pRHL: the programming language

Programs in pRHL are defined in terms of *expressions* $\mathcal{E}$ including constants, like the integers and booleans, as well as combinations of constants and variables with primitive operations, like addition and subtraction. We suppose $\mathcal{E}$ also includes terms for basic datatypes, like tuples and lists. Concretely, $\mathcal{E}$ is inductively defined by the following grammar:

$$
\begin{aligned}
\mathcal{E} \quad := \quad & \mathcal{X} \mid \mathcal{L} && \text{(variables)}\\
\mid\ & \mathbb{Z} \mid \mathcal{E}+\mathcal{E} \mid \mathcal{E}-\mathcal{E} \mid \mathcal{E}\cdot\mathcal{E} && \text{(numbers)}\\
\mid\ & \mathbb{B} \mid \mathcal{E}\wedge\mathcal{E} \mid \mathcal{E}\vee\mathcal{E} \mid \neg\mathcal{E} \mid \mathcal{E}=\mathcal{E} \mid \mathcal{E}<\mathcal{E} && \text{(booleans)}\\
\mid\ & (\mathcal{E},\ldots,\mathcal{E}) \mid \pi_i(\mathcal{E}) \mid [] \mid \mathcal{E}::\mathcal{E} \mid \mathcal{O}(\mathcal{E}) && \text{(tuples, lists, operations)}
\end{aligned}
$$

Expressions can mention two classes of variables: a countable set $\mathcal{X}$ of *program variables*, which can be modified by the program, and a set $\mathcal{L}$ of *logical variables*,

[1] Logics reasoning about two programs are known as *relational program logics* and were first considered by Benton (2004); see Section 5.7 for a discussion of other prior systems.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

which model fixed parameters. Expressions are typed as numbers, booleans, tuples, or lists, and primitive operations O have typed signatures; we consider only well-typed expressions throughout. The expressions $(\mathcal{E}, \ldots, \mathcal{E})$ and $\pi_i(\mathcal{E})$ construct and project from a tuple, respectively; [] is the empty list, and $\mathcal{E} :: \mathcal{E}$ adds an element to the head of a list. We typically use the letter e for expressions, $x, y, z, \ldots$ for program variables, and lower-case Greek letters $(\alpha, \beta, \ldots)$ and capital Roman letters $(N, M, \ldots)$ for logical variables.

We write $\mathcal{V}$ for the countable set of *values*, including integers, booleans, tuples, finite lists, etc. We can interpret expressions given maps from variables and logical variables to values.

Definition 5.21 Program states are *memories*, maps $\mathcal{X} \to \mathcal{V}$; we usually write m for a memory and **State** for the set of memories. *Logical contexts* are maps $\mathcal{L} \to \mathcal{V}$; we usually write ρ for a logical context.

We interpret an expression e as a function $[\![e]\!]_\rho : \mathbf{State} \to \mathcal{V}$ in the usual way, for instance:

$$[\![e_1 + e_2]\!]_\rho m \triangleq [\![e_1]\!]_\rho m + [\![e_2]\!]_\rho m.$$

Likewise, we interpret primitive operations o as functions $[\![o]\!]_\rho : \mathcal{V} \to \mathcal{V}$, so that

$$[\![o(e)]\!]_\rho m \triangleq [\![o]\!]_\rho([\![e]\!]_\rho m).$$

We fix a set $\mathcal{DE}$ of *distribution expressions* to model primitive distributions that our programs can sample from. For simplicity, we suppose for now that each distribution expression d is interpreted as a uniform distribution over a finite set. So, we have the coin flip and uniform distributions:

$$\mathcal{DE} \coloneqq \quad \mathbf{Flip} \quad | \quad \mathbf{Unif}(\mathcal{E})$$

where $\mathcal{E}$ is a list, representing the space of samples. We will introduce other primitive distributions as needed. To interpret distribution expressions, we define $[\![d]\!]_\rho : \mathbf{State} \to \mathbf{Distr}(\mathcal{V})$; for instance,

$$[\![\mathbf{Unif}(e)]\!]_\rho m \triangleq \mathcal{U}([\![e]\!]_\rho m)$$

where $\mathcal{U}(\mathcal{S})$ is the mathematical uniform distribution over a set $\mathcal{S}$.

Now let's see the programming language. We work with a standard imperative language with random sampling. The programs, also called *commands* or *statements*, are defined inductively:

$$\begin{array}{lcll} C & \coloneqq & \mathbf{skip} & \text{(no-op)} \\ & | & \mathcal{X} \leftarrow \mathcal{E} & \text{(assignment)} \\ & | & \mathcal{X} \xleftarrow{\$} \mathcal{DE} & \text{(sampling)} \end{array}$$

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

| $C; C$ (sequencing)
| **if** $\mathcal{E}$ **then** C **else** C (conditional)
| **while** $\mathcal{E}$ **do** C (loop)

We assume throughout that programs are well-typed; for instance, the guard expressions in conditionals and loops must be boolean.

We interpret each command as a mathematical function from states to sub-distributions over output states; this function is known as the *semantics* of a command. Since the set of program variables and the set of values are countable, the set of states is also countable so sub-distributions over states are discrete. To interpret commands, we use two basic constructions on sub-distributions.

Definition 5.22 The function $\mathbf{unit} : \mathcal{A} \to \mathbf{SDistr}(\mathcal{A})$ maps every element $a \in \mathcal{A}$ to the sub-distribution that places probability 1 on a. The function $\mathbf{bind} : \mathbf{SDistr}(\mathcal{A}) \times (\mathcal{A} \to \mathbf{SDistr}(\mathcal{B})) \to \mathbf{SDistr}(\mathcal{B})$ is defined by

$$\mathbf{bind}(\mu, f)(b) \triangleq \sum_{a \in \mathcal{A}} \mu(a) \cdot f(a)(b).$$

Intuitively, **bind** applies a randomized function on a distribution over inputs.

We use a discrete version of the semantics considered by Kozen (1981), presented in Fig. 5.1; we write $m[x \mapsto v]$ for the memory m with variable x updated to hold v, and $a \mapsto b(a)$ for the function mapping a to $b(a)$. The most complicated case is for loops. The sub-distribution $\mu^{(i)}(m)$ models executions that exit after entering the loop body at most i times, starting from initial memory m. For the base case $i = 0$, the sub-distribution either returns m with probability 1 when the guard is false and the loop exits immediately, or returns the null sub-distribution $\perp$ when the guard is true. The cases $i > 0$ are defined recursively, by unrolling the loop.

Note that $\mu^{(i)}$ are increasing in i: $\mu^{(i)}(m) \leq \mu^{(j)}(m)$ for all $m \in \mathbf{State}$ and $i \leq j$. In particular, the weights of the sub-distributions are increasing. Since the weights are at most 1, the approximants converge to a sub-distribution as i tends to infinity by the monotone convergence theorem (see, e.g., Rudin (1976, Theorem 11.28), taking the discrete counting measure over **State**).

5.5.3 The logic pRHL: judgments and validity

The program logic pRHL features judgments of the following form:

$$c_1 \sim c_2 : \Phi \Longrightarrow \Psi$$

Here, c_1 and c_2 are commands and Φ and Ψ are predicates on pairs of memories. To describe the inputs and outputs of c_1 and c_2, each predicate can mention two copies

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

$$
\begin{aligned}
[\![\mathbf{skip}]\!]_\rho m &\triangleq \mathbf{unit}(m)\\
[\![x \leftarrow e]\!]_\rho m &\triangleq \mathbf{unit}(m[x \mapsto [\![e]\!]_\rho m])\\
[\![x \xleftarrow{\$} d]\!]_\rho m &\triangleq \mathbf{bind}([\![d]\!]_\rho m, v \mapsto \mathbf{unit}(m[x \mapsto v]))\\
[\![c;\, c']\!]_\rho m &\triangleq \mathbf{bind}([\![c]\!]_\rho m, [\![c']\!]_\rho)\\
[\![\mathbf{if}\ e\ \mathbf{then}\ c\ \mathbf{else}\ c']\!]_\rho m &\triangleq \begin{cases} [\![c]\!]_\rho m & : [\![e]\!]_\rho m = \mathbf{true} \\ [\![c']\!]_\rho m & : [\![e]\!]_\rho m = \mathbf{false} \end{cases}\\
[\![\mathbf{while}\ e\ \mathbf{do}\ c]\!]_\rho m &\triangleq \lim_{i\to\infty} \mu^{(i)}(m)\\
\mu^{(i)}(m) &\triangleq \begin{cases} \bot & : i = 0 \wedge [\![e]\!]_\rho m = \mathbf{true} \\ \mathbf{unit}(m) & : i = 0 \wedge [\![e]\!]_\rho m = \mathbf{false} \\ \mathbf{bind}([\![\mathbf{if}\ e\ \mathbf{then}\ c]\!]_\rho m, \mu^{(i-1)}) & : i > 0 \end{cases}
\end{aligned}
$$

Figure 5.1 Semantics of programs

$x\langle 1\rangle$, $x\langle 2\rangle$ of each program variable x; these *tagged* variables refer to the value of x in the executions of c_1 and c_2 respectively.

Definition 5.23 Let $\mathcal{X}\langle 1\rangle$ and $\mathcal{X}\langle 2\rangle$ be the sets of *tagged variables*, finite sets of variable names tagged with $\langle 1\rangle$ or $\langle 2\rangle$ respectively:

$$\mathcal{X}\langle 1\rangle \triangleq \{x\langle 1\rangle \mid x \in \mathcal{X}\} \quad \text{and} \quad \mathcal{X}\langle 2\rangle \triangleq \{x\langle 2\rangle \mid x \in \mathcal{X}\}.$$

Let $\mathbf{State}\langle 1\rangle$ and $\mathbf{State}\langle 2\rangle$ be the sets of *tagged memories*, maps from tagged variables to values:

$$\mathbf{State}\langle 1\rangle \triangleq \mathcal{X}\langle 1\rangle \to \mathcal{V} \quad \text{and} \quad \mathbf{State}\langle 2\rangle \triangleq \mathcal{X}\langle 2\rangle \to \mathcal{V}.$$

Let $\mathbf{State}_\times$ be the set of *product memories*, which combine two tagged memories:

$$\mathbf{State}_\times \triangleq \mathcal{X}\langle 1\rangle \uplus \mathcal{X}\langle 2\rangle \to \mathcal{V}.$$

For notational convenience we identify $\mathbf{State}_\times$ with pairs of memories $\mathbf{State}\langle 1\rangle \times \mathbf{State}\langle 2\rangle$; for $m_1 \in \mathbf{State}\langle 1\rangle$ and $m_2 \in \mathbf{State}\langle 2\rangle$, we write (m_1, m_2) for the product memory and we use the usual projections on pairs to extract untagged memories from the product memory:

$$p_1(m_1, m_2) \triangleq |m_1| \quad \text{and} \quad p_2(m_1, m_2) \triangleq |m_2|,$$

where the memory $|m| \in \mathbf{State}$ has all variables in $\mathcal{X}$. For commands c and expressions e with variables in $\mathcal{X}$, we write $c\langle 1\rangle$, $c\langle 2\rangle$ and $e\langle 1\rangle$, $e\langle 2\rangle$ for the corresponding *tagged commands* and *tagged expressions* with variables in $\mathcal{X}\langle 1\rangle$ and $\mathcal{X}\langle 2\rangle$.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

We consider a set $\mathcal{P}$ of *predicates* (*assertions*) from first-order logic defined by the following grammar:

$$\begin{aligned}\mathcal{P} :=\quad & \mathcal{E}\langle 1/2\rangle = \mathcal{E}\langle 1/2\rangle \;\mid\; \mathcal{E}\langle 1/2\rangle < \mathcal{E}\langle 1/2\rangle \;\mid\; \mathcal{E}\langle 1/2\rangle \in \mathcal{E}\langle 1/2\rangle \\ \mid\; & \top \;\mid\; \bot \;\mid\; \mathcal{O}(\mathcal{E}\langle 1/2\rangle, \ldots, \mathcal{E}\langle 1/2\rangle) && \text{(predicates)} \\ \mid\; & \mathcal{P} \wedge \mathcal{P} \;\mid\; \mathcal{P} \vee \mathcal{P} \;\mid\; \neg\mathcal{P} \;\mid\; \mathcal{P} \to \mathcal{P} \;\mid\; \forall \mathcal{L} \in \mathbb{Z},\, \mathcal{P} \;\mid\; \exists \mathcal{L} \in \mathbb{Z},\, \mathcal{P} \\ & && \text{(first-order formulas)}\end{aligned}$$

We typically use capital Greek letters ($\Phi, \Psi, \Theta, \Xi, \ldots$) for predicates. $\mathcal{E}\langle 1/2\rangle$ denotes an expression where program variables are tagged with $\langle 1\rangle$ or $\langle 2\rangle$; tags may be mixed within an expression. We consider the usual binary predicates $\{=, <, \in, \ldots\}$ where $e \in e'$ means e is a member of the list e', and we take the always-true and always-false predicates $\top$ and $\bot$, and a set $\mathcal{O}$ of other predicates. Predicates can be combined using the usual connectives $\{\wedge, \vee, \neg, \to\}$ and can quantify over first-order types (e.g., the integers, tuples, etc.). We will often interpret a boolean expression e as the predicate $e = \mathbf{true}$.

Predicates are interpreted as sets of product memories.

Definition 5.24 Let Φ be a predicate. Given a logical context ρ, Φ is interpreted as a set $[\![\Phi]\!]_\rho \subseteq \mathbf{State}_\times$ in the expected way, e.g.,

$$[\![e_1\langle 1\rangle < e_2\langle 2\rangle]\!]_\rho \triangleq \{(m_1, m_2) \in \mathbf{State}_\times \mid [\![e_1]\!]_\rho m_1 < [\![e_2]\!]_\rho m_2\}.$$

We can inject a predicate on single memories into a predicate on product memories; we call the resulting predicate *one-sided* since it constrains just one of two memories.

Definition 5.25 Let Φ be a predicate on $\mathbf{State}$. We define formulas $\Phi\langle 1\rangle$ and $\Phi\langle 2\rangle$ by replacing all program variables x in Φ with $x\langle 1\rangle$ and $x\langle 2\rangle$, respectively, and we define

$$[\![\Phi\langle 1\rangle]\!]_\rho \triangleq \{(m_1, m_2) \mid m_1 \in [\![\Phi]\!]_\rho\} \quad \text{and} \quad [\![\Phi\langle 2\rangle]\!]_\rho \triangleq \{(m_1, m_2) \mid m_2 \in [\![\Phi]\!]_\rho\}.$$

Valid judgments in pRHL relate two output distributions by lifting the post-condition.

Definition 5.26 (Barthe et al. (2009)) A judgment is *valid* in logical context ρ, written $\rho \models c_1 \sim c_2 : \Phi \Longrightarrow \Psi$, if for any two memories $(m_1, m_2) \in [\![\Phi]\!]_\rho$ there exists a lifting of Ψ relating the output distributions:

$$[\![c_1]\!]_\rho m_1 \; [\![\Psi]\!]_\rho^\sharp \; [\![c_2]\!]_\rho m_2.$$

For example, a valid judgment

$$\models c_1 \sim c_2 : \Phi \Longrightarrow (=),$$

states that for any two input memories (m_1, m_2) satisfying Φ, the resulting output

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

distributions from running c_1 and c_2 are related by lifted equality; by Theorem 5.10, these output distributions must be equal.

5.5.4 The logic pRHL: the proof rules

Valid judgments in pRHL state that the output distributions of two programs are related in a specific way. However, it is generally not possible to directly prove validity—even simple probabilistic programs with loops can produce highly complex output distributions, and we typically do not have a description of the output distribution that is more precise than the probabilistic program itself.

To make reasoning about probabilistic programs more tractable, pRHL includes a collection of logical rules to inductively build up a proof of a new judgment from known judgments, combining proofs about sub-programs into proofs about larger programs. The rules are superficially similar to those from standard Hoare logic. However, the interpretation of judgments in terms of liftings means some rules in pRHL are not valid in Hoare logic, and vice versa.

Before describing the rules, we introduce some necessary notation. A system of logical rules inductively defines a set of *derivable* formulas; we use the head symbol $\vdash$ to mark such formulas. As is standard in logic, the premises in each logical rule are written above the horizontal line, and the single conclusion is written below the line; for easy reference, the name of each rule is given to the left of the line.

The main premises are judgments in the program logic, but rules may also use other *side-conditions*. For instance, many rules require an assertion logic formula to be valid in all memories. Other side-conditions state that a program is terminating, or that certain variables are not modified by the program. We use the head symbol $\models$ to mark valid side-conditions; while we could give a separate proof system for these premises, in practice they are simple enough to check directly.

We also use notation for substitution in assertions. We write $\Phi\{e/x\}$ for the formula Φ with every occurrence of the variable x replaced by e. Similarly, $\Phi\{v_1, v_2/x_1\langle 1\rangle, x_2\langle 2\rangle\}$ is the formula Φ where occurrences of the tagged variables $x_1\langle 1\rangle, x_2\langle 2\rangle$ are replaced by v_1, v_2 respectively.

Now, we take a tour through the logical rules of pRHL. While most of these rules were proposed in prior works, we use the coupling interpretation to give a new way of thinking about what the rules mean. It turns out that certain steps of the informal coupling proofs we saw in Section 5.4 correspond to specific logical rules in pRHL, seen from the right perspective. We summarize this reading in Section 5.5.

One important feature of coupling proofs can already be seen in the form of the proof rules: though the notion of validity and target properties are probabilisitic—i.e., they describe pairs of probability *distributions*—assertions describe individual *mem-*

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

$$\textsc{Skip}\ \frac{}{\vdash \mathbf{skip} \sim \mathbf{skip} : \Phi \Longrightarrow \Phi}$$

$$\textsc{Assn}\ \frac{}{\vdash x_1 \leftarrow e_1 \sim x_2 \leftarrow e_2 : \Psi\,\{e_1\langle 1\rangle, e_2\langle 2\rangle / x_1\langle 1\rangle, x_2\langle 2\rangle\} \Longrightarrow \Psi}$$

$$\textsc{Sample}\ \frac{f : \mathrm{supp}(d_1) \to \mathrm{supp}(d_2) \text{ is a bijection}}{\vdash x_1 \xleftarrow{\$} d_1 \sim x_2 \xleftarrow{\$} d_2 : \forall v \in \mathrm{supp}(d_1),\ \Psi\,\{v, f(v)/x_1\langle 1\rangle, x_2\langle 2\rangle\} \Longrightarrow \Psi}$$

$$\textsc{Seq}\ \frac{\vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi \qquad \vdash c_1' \sim c_2' : \Psi \Longrightarrow \Theta}{\vdash c_1; c_1' \sim c_2; c_2' : \Phi \Longrightarrow \Theta}$$

$$\textsc{Cond}\ \frac{\models \Phi \to e_1\langle 1\rangle = e_2\langle 2\rangle \qquad \vdash c_1 \sim c_2 : \Phi \wedge e_1\langle 1\rangle \Longrightarrow \Psi \qquad \vdash c_1' \sim c_2' : \Phi \wedge \neg e_1\langle 1\rangle \Longrightarrow \Psi}{\vdash \mathbf{if}\ e_1\ \mathbf{then}\ c_1\ \mathbf{else}\ c_1' \sim \mathbf{if}\ e_2\ \mathbf{then}\ c_2\ \mathbf{else}\ c_2' : \Phi \Longrightarrow \Psi}$$

$$\textsc{While}\ \frac{\models \Phi \to e_1\langle 1\rangle = e_2\langle 2\rangle \qquad \vdash c_1 \sim c_2 : \Phi \wedge e_1\langle 1\rangle \Longrightarrow \Phi}{\vdash \mathbf{while}\ e_1\ \mathbf{do}\ c_1 \sim \mathbf{while}\ e_2\ \mathbf{do}\ c_2 : \Phi \Longrightarrow \Phi \wedge \neg e_1\langle 1\rangle}$$

Figure 5.2 Two-sided pRHL rules

ories, rather than distributions over memories. Abstracting away from probabilistic aspect of the program makes probabilistic reasoning significantly easier, and is a key reason why the coupling proof technique is so powerful.

The rules of pRHL can be divided into three groups: *two-sided* rules, *one-sided* rules, and *structural* rules. All judgments are parameterized by a logical context ρ, but since this context is assumed to be a fixed assignment of logical variables—constant throughout the proof—we omit it from the rules. The two-sided rules in Fig. 5.2 apply when the two programs in the conclusion judgment have the same top-level shape.

The rule [Skip] simply states that **skip** instructions preserve the pre-condition. The rule [Assn] handles assignment instructions. It is the usual Hoare-style rule: if Ψ holds initially with $e_1\langle 1\rangle$ and $e_2\langle 2\rangle$ substituted for $x_1\langle 1\rangle$ and $x_2\langle 2\rangle$, then Ψ holds after the respective assignment instructions.

The rule [Sample] is more subtle. In some ways it is the key rule in pRHL, allowing us to select a coupling for a pair of sampling instructions. To gain intuition, the following rule is a special case:

$$\textsc{Sample}^*\ \frac{f : \mathrm{supp}(d) \to \mathrm{supp}(d) \text{ is a bijection}}{\vdash x \xleftarrow{\$} d \sim x \xleftarrow{\$} d : \top \Longrightarrow f(x\langle 1\rangle) = x\langle 2\rangle}$$

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

In terms of couplings, the conclusion states that there exists a coupling of a distribution d with itself such that each sample x from d is related to $f(x)$. Soundness of this rule crucially relies on d being *uniform*—as we have seen, any bijection f induces a coupling of uniform distributions (cf. Theorem 5.4). It is possible to support more general distributions at the cost of a more complicated side-condition,[2] but we will not need this generality. The full rule [Sample] can prove a post-condition of any shape: a post-condition holds after sampling if it holds before sampling, where $x\langle 1\rangle$ and $x\langle 2\rangle$ are replaced by any two coupled samples $(v, f(v))$.

More generally, a key feature of coupling proofs can be seen in the rule [Sample]: reducing two separate sources of randomness into a single source of randomness. A priori, two sampling instructions in the two programs are completely independent—we have no reason to think that the results of their random draws are related in any way. The sampling rule shows that for the purpose of the proof, it is sound to *assume* that the results from the two sampling statements are related by a bijection. In this way, we may analyze the two programs with their separate sources of randomness as if they shared a single, common source of randomness. When the original programs have similar shapes—as is the typical case in relational verification—this coordination enables us to limit our attention to pairs of highly similar program executions.

The rule [Seq] resembles the normal rule for sequential composition in Hoare logic, but this superficial similarity hides some complexity. In particular, note that the intermediate assertion Ψ is interpreted differently in the two premises: in the first judgment it is a post-condition and interpreted as a relation between *distributions over memories* via lifting, while in the second judgment it is a pre-condition and interpreted as a relation between *memories*, not distributions over memories.

The next two rules deal with branching commands. Rule [Cond] requires that the guards $e_1\langle 1\rangle$ and $e_2\langle 2\rangle$ are equal assuming the pre-condition Φ. The rule is otherwise similar to the standard Hoare logic rule: if we can prove the post-condition Ψ when the guard is initially true and when the guard is initially false, then we can prove Ψ as a post-condition of the conditional.

Rule [While] uses a similar idea for loops. We again assume that the guards are initially equal, and we also assume that they are equal in the post-condition of the loop body. Since the judgments are interpreted in terms of couplings, this second condition is a bit subtle. For one thing, the rule does *not* require $e_1\langle 1\rangle = e_2\langle 2\rangle$ in all possible executions of the two programs—this would be a rather severe restriction, for instance ruling out programs where $e_1\langle 1\rangle$ and $e_2\langle 2\rangle$ are probabilistic. Rather, the guards only need to be equal *under the coupling of the two programs given by the premise*. The upshot is that by selecting appropriate couplings in the loop body,

[2] Roughly speaking, the probability of any set S under d should be equal to the probability of $f(S)$ under d.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

$$\textsc{Assn-L}\ \frac{}{\vdash x_1 \leftarrow e_1 \sim \textbf{skip} : \Psi\,\{e_1\langle 1\rangle / x_1\langle 1\rangle\} \Longrightarrow \Psi}$$

$$\textsc{Assn-R}\ \frac{}{\vdash \textbf{skip} \sim x_2 \leftarrow e_2 : \Psi\,\{e_2\langle 2\rangle / x_2\langle 2\rangle\} \Longrightarrow \Psi}$$

$$\textsc{Sample-L}\ \frac{}{\vdash x_1 \xleftarrow{\$} d_1 \sim \textbf{skip} : \forall v \in \mathrm{supp}(d_1),\ \Psi\,\{v / x_1\langle 1\rangle\} \Longrightarrow \Psi}$$

$$\textsc{Sample-R}\ \frac{}{\vdash \textbf{skip} \sim x_2 \xleftarrow{\$} d_2 : \forall v \in \mathrm{supp}(d_2),\ \Psi\,\{v / x_2\langle 2\rangle\} \Longrightarrow \Psi}$$

$$\textsc{Cond-L}\ \frac{\vdash c_1 \sim c : \Phi \wedge e_1\langle 1\rangle \Longrightarrow \Psi \qquad \vdash c_1' \sim c : \Phi \wedge \neg e_1\langle 1\rangle \Longrightarrow \Psi}{\vdash \textbf{if}\ e_1\ \textbf{then}\ c_1\ \textbf{else}\ c_1' \sim c : \Phi \Longrightarrow \Psi}$$

$$\textsc{Cond-R}\ \frac{\vdash c \sim c_2 : \Phi \wedge e_2\langle 2\rangle \Longrightarrow \Psi \qquad \vdash c \sim c_2' : \Phi \wedge \neg e_2\langle 2\rangle \Longrightarrow \Psi}{\vdash c \sim \textbf{if}\ e_2\ \textbf{then}\ c_2\ \textbf{else}\ c_2' : \Phi \Longrightarrow \Psi}$$

$$\textsc{While-L}\ \frac{\begin{array}{c}\vdash c_1 \sim \textbf{skip} : \Phi \wedge e_1\langle 1\rangle \Longrightarrow \Phi \\ \models \Phi \to \Phi_1\langle 1\rangle \qquad \Phi_1 \models \textbf{while}\ e_1\ \textbf{do}\ c_1\ \text{lossless}\end{array}}{\vdash \textbf{while}\ e_1\ \textbf{do}\ c_1 \sim \textbf{skip} : \Phi \Longrightarrow \Phi \wedge \neg e_1\langle 1\rangle}$$

$$\textsc{While-R}\ \frac{\begin{array}{c}\vdash \textbf{skip} \sim c_2 : \Phi \wedge e_2\langle 2\rangle \Longrightarrow \Phi \\ \models \Phi \to \Phi_2\langle 2\rangle \qquad \Phi_2 \models \textbf{while}\ e_2\ \textbf{do}\ c_2\ \text{lossless}\end{array}}{\vdash \textbf{skip} \sim \textbf{while}\ e_2\ \textbf{do}\ c_2 : \Phi \Longrightarrow \Phi \wedge \neg e_2\langle 2\rangle}$$

Figure 5.3 One-sided pRHL rules

we can assume the guards are equal when analyzing loops with probabilistic guards. The rule is otherwise similar to the usual Hoare logic rule, where Φ is the loop invariant.

So far, we have seen rules that relate two programs of the same shape. These are the most commonly used rules in pRHL, as relational reasoning is most powerful when comparing two highly similar (or even the same) programs. However, in some cases we may need to reason about two programs with different shapes, even if the two top-level commands are the same. For instance, if we can't guarantee two executions of a program follow the same path at a conditional statement under a coupling, we must relate the two different branches. For this kind of reasoning, we can fall back on the *one-sided* rules in Fig. 5.3. These rules relate a command of a particular shape with **skip** or an arbitrary command. Each rule comes in a left- and a right-side version.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

The assignment rules, [Assn-L] and [Assn-R], relate an assignment instruction to **skip** using the usual Hoare rule for assignment instructions. The sampling rules, [Sample-L] and [Sample-R], are similar; they relate a sampling instruction to **skip** if the post-condition holds for all possible values of the sample. These rules represent couplings where fresh randomness is used, i.e., where randomness is not shared between the two programs.

The conditional rules, [Cond-L] and [Cond-R], are similar to the two-sided conditional rule except there is no assumption of synchronized guards—the other command c might not even be a conditional. If we can relate the general command c to the true branch when the guard is true and relate c to the false branch when the guard is false, then we can relate c to the whole conditional.

The rules for loops, [While-L] and [While-R], can only relate loops to the **skip**; a loop that executes multiple iterations cannot be directly related to an arbitrary command that executes only once. These rules mimic the usual loop rule from Hoare logic, with a critical side-condition: *losslessness*.

Definition 5.27 A command c is Φ-*lossless* if for any memory m satisfying Φ and every logical context ρ, the output $[\![c]\!]_\rho m$ is a proper distribution (i.e., it has total probability 1). We write Φ-lossless as the following judgment:

$$\Phi \models c \text{ lossless}$$

Losslessness is needed for soundness: **skip** produces a proper distribution on any input and liftings can only relate sub-distributions with equal weights (Theorem 5.9), so the loop must also produce a proper distribution to have any hope of coupling the output distributions. For the examples we will consider, losslessness is easy to show since loops execute for a finite number of iterations; when there is no finite bound, proving losslessness may require more sophisticated techniques (e.g., Barthe et al. (2018a); Ferrer Fioriti and Hermanns (2015); Chatterjee et al. (2016b,a, 2017); McIver et al. (2018)).

Finally, pRHL includes a handful of *structural* rules which apply to programs of any shape. The first rule [Conseq] is the usual rule of consequence, allowing us to strengthen the pre-condition and weaken the post-condition—assuming more about the input and proving less about the output, respectively.

The rule [Equiv] replaces programs by equivalent programs. This rule is particularly useful for reasoning about programs of different shapes. Instead of using one-sided rules, which are often less convenient, we can sometimes replace a program with an equivalent version and then apply two-sided rules. For simplicity, we use a strong notion of equivalence:

$$c_1 \equiv c_2 \triangleq [\![c_1]\!]_\rho = [\![c_2]\!]_\rho$$

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

$$\textsc{Conseq}\ \frac{\vdash c_1 \sim c_2 : \Phi' \Longrightarrow \Psi' \qquad \models \Phi \to \Phi' \qquad \models \Psi' \to \Psi}{\vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi}$$

$$\textsc{Equiv}\ \frac{\vdash c_1' \sim c_2' : \Phi \Longrightarrow \Psi \qquad c_1 \equiv c_1' \qquad c_2 \equiv c_2'}{\vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi}$$

$$\textsc{Case}\ \frac{\vdash c_1 \sim c_2 : \Phi \wedge \Theta \Longrightarrow \Psi \qquad \vdash c_1 \sim c_2 : \Phi \wedge \neg\Theta \Longrightarrow \Psi}{\vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi}$$

$$\textsc{Trans}\ \frac{\vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi \qquad \vdash c_2 \sim c_3 : \Phi' \Longrightarrow \Psi'}{\vdash c_1 \sim c_3 : \Phi' \circ \Phi \Longrightarrow \Psi' \circ \Psi}$$

$$\textsc{Frame}\ \frac{\vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi \qquad \mathrm{FV}(\Theta) \cap \mathrm{MV}(c_1, c_2) = \varnothing}{\vdash c_1 \sim c_2 : \Phi \wedge \Theta \Longrightarrow \Psi \wedge \Theta}$$

Figure 5.4 Structural pRHL rules

for every logical context ρ; more refined notions of equivalence are also possible, but will not be needed for our purposes. For our examples, we just use a handful of basic program equivalences, e.g., c; **skip** $\equiv c$ and **skip**; $c \equiv c$.

The rule [Case] performs a case analysis on the input. If we can prove a judgment when Θ holds initially and a judgment when Θ does not hold initially, then we can combine the two judgments provided they have the same post-condition.

The rule [Trans] is the transitivity rule: given a judgment relating $c_1 \sim c_2$ and a judgment relating $c_2 \sim c_3$, we can glue these judgments together to relate $c_1 \sim c_3$. The pre- and post-conditions of the conclusion are given by composing the pre- and post-conditions of the premises; for binary relations $\mathcal{R}$ and $\mathcal{S}$, relation composition is defined by

$$\mathcal{R} \circ \mathcal{S} \triangleq \{(x_1, x_3) \mid \exists x_2.\ (x_1, x_2) \in \mathcal{S} \wedge (x_2, x_3) \in \mathcal{R}\}.$$

The last rule [Frame] is the frame rule (also called the *rule of constancy*): it states that an assertion Θ can be carried from the pre-condition through to the post-condition as long as the variables $\mathrm{MV}(c_1, c_2)$ that may be modified by the programs c_1 and c_2 don't include any of the variables $\mathrm{FV}(\Theta)$ appearing free in Θ; as usual, MV and FV are defined syntactically by collecting the variables that occur in programs and assertions.

As expected, the proof system of pRHL is sound.

Theorem 5.28 (Barthe et al. (2009)) *Let ρ be a logical context. If a judgment is*

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

derivable

$$\rho \vdash c_1 \sim c_2 : \Phi \Longrightarrow \Psi,$$

then it is valid:

$$\rho \models c_1 \sim c_2 : \Phi \Longrightarrow \Psi.$$

5.5.5 The coupling interpretation

Now that we have seen the core logical rules of pRHL, we revisit the connection with couplings. Not only do valid judgments assert the existence of a coupling between two output distributions, the proof system itself is a formalization of proofs by coupling, which we saw in Section 5.4. This perspective gives a better, more precise understanding of what proofs by coupling are, and how they work.

In more detail, a valid judgment $\rho \models c_1 \sim c_2 : \Phi \Longrightarrow \Psi$ implies that for any two input memories related by Φ, there exists a coupling with support in Ψ between the two output distributions. By applying the results in Section 5.3, valid judgments imply relational properties of programs.

Moreover, we can identify common steps in standard coupling proofs in the form of specific logical rules. For instance, [Sample] selects a coupling for corresponding sampling statements; the function f lets us choose among different bijection couplings. The rule [Seq] formalizes the sequential composition principle for couplings; when two processes produce samples related by Ψ under a particular coupling, we can continue to assume this relation when analyzing the remainder of the program. The structural rule [Case] shows we can select between two possible couplings depending on whether a predicate Θ holds. In short, not only is pRHL a logic for verifying cryptographic constructions, it is also a formal logic for proofs by coupling.

5.6 Constructing couplings, formally

The coupling proof technique has been applied to a variety of probabilistic properties, using the same basic pattern: construct a coupling of a specific form between the output distributions of two programs, then use the existence of this coupling to conclude a relational property using known consequences of couplings (cf. Section 5.3). Given the close connection between coupling proofs and our logic, we carry out this proof pattern in pRHL to build formal proofs of three classical properties: equivalence, stochastic domination, and convergence.

Remark 5.29 There are some inherent challenges in presenting formal proofs on paper. Fundamentally, our proofs are branching derivation trees. When such a proof

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

is linearized, it becomes more difficult to follow which part of the derivation tree the paper proof corresponds to. To help organize the proof, we generally proceed in a top-down fashion, giving proofs and judgments for the most deeply nested parts of the program first and then gradually zooming out to consider larger and larger parts of the whole program.

Applications of sequential composition are also natural places to signpost the proof. We typically consider the commands in order, unless the second command is much more complex than the first. Finally, for space reasons we will gloss over applications of the assignment rule [Assn] and minor uses of the rule of consequence [Conseq]; a completely formal proof would necessarily include these details.

5.6.1 Probabilistic equivalence

To warm up, we prove two programs to be probabilistically equivalent. Our example models perhaps the most basic encryption scheme: the XOR cipher. Given a boolean s representing the secret message, the XOR cipher flips a fair coin to draw the secret key k and then returns $k \oplus s$ as the encrypted message. A receiving party who knows the secret key can decrypt the message by computing $k \oplus (k \oplus s) = s$.

To prove secrecy of this scheme, we consider the following two programs:

$$\begin{array}{l|l} k \xleftarrow{\$} \textbf{Flip}; & k \xleftarrow{\$} \textbf{Flip}; \\ r \leftarrow k \oplus s & r \leftarrow k \end{array}$$

The first program xor_1 implements the encryption function, storing the encrypted message into r. The second program xor_2 simply stores a random value into r. If we can show the distribution of r is the same in both programs, then the XOR cipher is secure: the distribution on outputs is completely random, leaking no information about the secret message s. In terms of pRHL, it suffices to prove the following judgment:

$$\vdash xor_1 \sim xor_2 : \top \Longrightarrow r\langle 1\rangle = r\langle 2\rangle$$

By validity of the logic, this judgment implies that for any two memories m_1, m_2, the output distributions are related by a coupling that always returns outputs with equal values of r; by reasoning similar to Theorem 5.10, this implies that the output distributions over $r\langle 1\rangle$ and $r\langle 2\rangle$ are equal.[3]

Before proving this judgment in the logic, we sketch the proof by coupling. If $s\langle 1\rangle$ is true, then we couple k to take opposite values in the two runs. If $s\langle 1\rangle$ is false, then we couple k to be equal in the two runs. In both cases, we conclude that the results $r\langle 1\rangle, r\langle 2\rangle$ are equal under the coupling.

[3] To be completely precise, Theorem 5.10 assumes that we have lifted equality, while here we only have a lifting where the variables r are equal. An analogous argument shows that the marginal distributions of variable r must be equal.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

To formalize this argument in pRHL, we use the [Case] rule:

$$\textsc{Case}\ \frac{\vdash xor_1 \sim xor_2 : s\langle 1\rangle = \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle \qquad \vdash xor_1 \sim xor_2 : s\langle 1\rangle \neq \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}{\vdash xor_1 \sim xor_2 : \top \implies r\langle 1\rangle = r\langle 2\rangle}\ .$$

For the first premise we select the negation coupling using the bijection $f = \neg$ in [Sample], apply the assignment rule [Assn], and combine with the sequencing rule [Seq]. Concretely, we have

$$\textsc{Sample}\ \frac{f = \neg}{\vdash k \xleftarrow{\$} \mathbf{Flip} \sim k \xleftarrow{\$} \mathbf{Flip} : s\langle 1\rangle = \mathbf{true} \implies k\langle 1\rangle = \neg k\langle 2\rangle \wedge s\langle 1\rangle = \mathbf{true}}$$

$$\textsc{Assn}\ \frac{}{\vdash r \leftarrow k \oplus s \sim r \leftarrow k : k\langle 1\rangle = \neg k\langle 2\rangle \wedge s\langle 1\rangle = \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}$$

and we combine the two judgments to give:

$$\textsc{Seq}\ \frac{\vdash k \xleftarrow{\$} \mathbf{Flip} \sim k \xleftarrow{\$} \mathbf{Flip} : s\langle 1\rangle = \mathbf{true} \implies k\langle 1\rangle = \neg k\langle 2\rangle \wedge s\langle 1\rangle = \mathbf{true} \qquad \vdash r \leftarrow k \oplus s \sim r \leftarrow k : k\langle 1\rangle = \neg k\langle 2\rangle \wedge s\langle 1\rangle = \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}{\vdash xor_1 \sim xor_2 : s\langle 1\rangle = \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}\ .$$

For the other case $s\langle 1\rangle \neq \mathbf{true}$, we give the same proof except with the identity coupling in [Sample]:

$$\textsc{Sample}\ \frac{f = \mathrm{id}}{\vdash k \xleftarrow{\$} \mathbf{Flip} \sim k \xleftarrow{\$} \mathbf{Flip} : s\langle 1\rangle \neq \mathbf{true} \implies k\langle 1\rangle = k\langle 2\rangle \wedge s\langle 1\rangle \neq \mathbf{true}}$$

and the assignment rule, we have

$$\textsc{Assn}\ \frac{}{\vdash r \leftarrow k \oplus s \sim r \leftarrow k : k\langle 1\rangle = k\langle 2\rangle \wedge s\langle 1\rangle \neq \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}\ .$$

Combining the conclusions, we get

$$\textsc{Seq}\ \frac{\vdash k \xleftarrow{\$} \mathbf{Flip} \sim k \xleftarrow{\$} \mathbf{Flip} : s\langle 1\rangle \neq \mathbf{true} \implies k\langle 1\rangle = \neg k\langle 2\rangle \wedge s\langle 1\rangle \neq \mathbf{true} \qquad \vdash r \leftarrow k \oplus s \sim r \leftarrow k : k\langle 1\rangle = k\langle 2\rangle \wedge s\langle 1\rangle \neq \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}{\vdash xor_1 \sim xor_2 : s\langle 1\rangle \neq \mathbf{true} \implies r\langle 1\rangle = r\langle 2\rangle}\ .$$

By [Case], we conclude the desired post-condition $r\langle 1\rangle = r\langle 2\rangle$.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

5.6.2 Stochastic domination

For our second example, we revisit Theorem 5.20 and replicate the proof in pRHL. The following program *sdom* flips a coin T times and returns the number of coin flips that come up true:

$$\begin{array}{l} i \leftarrow 0; ct \leftarrow 0; \\ \textbf{while } i < T \textbf{ do} \\ \quad i \leftarrow i + 1; \\ \quad s \xleftarrow{\$} \textbf{Flip}; \\ \quad ct \leftarrow s \ ? \ ct + 1 : ct \end{array}$$

(The last line uses the *ternary conditional operator*—$s \ ? \ ct + 1 : ct$ is equal to $ct + 1$ if s is true, otherwise equal to ct.)

We consider two runs of this program executing T_1 and T_2 iterations, where $T_1 \leq T_2$ are logical variables; call the two programs $sdom_1$ and $sdom_2$. By soundness of the logic and Theorem 5.14, the distribution of ct in the second run stochastically dominates the distribution of ct in the first run if we can prove the judgment

$$\vdash sdom_1 \sim sdom_2 : \top \Longrightarrow ct\langle 1 \rangle \leq ct\langle 2 \rangle.$$

Encoding the argument from Theorem 5.20 in pRHL requires a bit of work. The main obstacle is that the two-sided loop rule in pRHL can only analyze loops in a synchronized fashion, but this is not possible here: when $T_1 < T_2$ the two loops run for different numbers of iterations, no matter how we couple the samples. To get around this problem, we use the equivalence rule [Equiv] to transform *sdom* into a more convenient form using the following equivalence:

$$\textbf{while } e \textbf{ do } c \equiv \textbf{while } e \wedge e' \textbf{ do } c; \textbf{while } e \textbf{ do } c$$

This transformation, known in the compilers literature as *loop splitting* (Callahan and Kennedy, 1988), separates out the first iterations where e' holds, and then runs the original loop to completion. We transform $sdom_2$ as follows:

$$sdom'_{2a} \triangleq \left\{ \begin{array}{ll} i \leftarrow 0; ct \leftarrow 0; & i \leftarrow 0; ct \leftarrow 0; \\ \textbf{while } i < T_2 \wedge i < T_1 \textbf{ do} & \textbf{while } i < T_1 \textbf{ do} \\ \quad i \leftarrow i + 1; & \quad i \leftarrow i + 1; \\ \quad s \xleftarrow{\$} \textbf{Flip}; & \quad s \xleftarrow{\$} \textbf{Flip}; \\ \quad ct \leftarrow s \ ? \ ct + 1 : ct; & \quad ct \leftarrow s \ ? \ ct + 1 : ct; \end{array} \right\} \triangleq sdom_1$$

$$sdom'_{2b} \triangleq \left\{ \begin{array}{l} \textbf{while } i < T_2 \textbf{ do} \\ \quad i \leftarrow i + 1; \\ \quad s \xleftarrow{\$} \textbf{Flip}; \\ \quad ct \leftarrow s \ ? \ ct + 1 : ct \end{array} \right.$$

We aim to relate $sdom'_{2a}; sdom'_{2b}$ to $sdom_1$. First, we apply the two-sided rule

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

[While] to relate $sdom_1$ to $sdom'_{2a}$. Taking the identity coupling with $f = \mathrm{id}$ in [Sample], we relate the sampling in the loop body via

$$\textsc{Sample}\ \frac{f = \mathrm{id}}{\vdash s \xleftarrow{\$} \mathbf{Flip} \sim s \xleftarrow{\$} \mathbf{Flip} : \top \Longrightarrow s\langle 1\rangle = s\langle 2\rangle}$$

and establish the loop invariant

$$\Theta \triangleq i\langle 1\rangle = i\langle 2\rangle \wedge ct\langle 1\rangle = ct\langle 2\rangle,$$

proving the judgment

$$\vdash sdom_1 \sim sdom'_{2a} : \top \Longrightarrow \Theta.$$

Then we use the one-sided rule [While-R] for the loop $sdom'_{2b}$ with loop invariant $ct\langle 1\rangle \leq ct\langle 2\rangle$:

$$\vdash \mathbf{skip} \sim sdom'_{2b} : \Theta \Longrightarrow ct\langle 1\rangle \leq ct\langle 2\rangle.$$

Composing these two judgments with [Seq] and applying [Equiv] gives the desired judgment:

$$\textsc{Equiv}\ \frac{\vdash sdom_1; \mathbf{skip} \sim sdom'_{2a}; sdom'_{2b} : \top \Longrightarrow ct\langle 1\rangle \leq ct\langle 2\rangle}{\vdash sdom_1 \sim sdom_2 : \top \Longrightarrow ct\langle 1\rangle \leq ct\langle 2\rangle}$$

using the equivalence $sdom_1; \mathbf{skip} \equiv sdom_1$.

5.6.3 Probabilistic convergence

In our final example, we build a coupling witnessing convergence of two *random walks*. Each process begins at an integer starting point *start*, and proceeds for T steps. At each step it flips a fair coin. If true, it increases the current position by 1; otherwise, it decreases the position by 1. Given two random walks starting at different initial locations, we want to bound the distance between the two resulting output distributions in terms of T. Intuitively, the position distributions spread out as the random walks proceed, tending towards the uniform distribution on the even integers or the uniform distribution over the odd integers depending on the parity of the initial position and the number of steps. If two walks initially have the same parity (i.e., their starting positions differ by an even integer), then their distributions after taking the same number of steps T should approach one another in total variation distance.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

We model a single random walk with the following program *rwalk*:

$$
\begin{array}{l}
pos \leftarrow start; i \leftarrow 0; hist \leftarrow [start]; \\
\textbf{while } i < T \textbf{ do} \\
\quad i \leftarrow i + 1; \\
\quad r \xleftarrow{\$} \textbf{Flip}; \\
\quad pos \leftarrow pos + (r \ ? \ 1 : -1); \\
\quad hist \leftarrow pos :: hist
\end{array}
$$

The last command records the history of the walk in *hist*; this *ghost variable* does not affect the final output value, but will be useful for our assertions.

By Theorem 5.16, we can bound the TV-distance between the position distributions by constructing a coupling where the probability of $pos\langle 1\rangle \neq pos\langle 2\rangle$ tends to 0 as T increases. We don't have the tools yet to reason about this probability (we will revisit this point in the next chapter), but for now we can build the coupling and prove the judgment

$$
\begin{aligned}
&\vdash rwalk \sim rwalk : start\langle 2\rangle - start\langle 1\rangle = 2K \\
&\Longrightarrow K + start\langle 1\rangle \in hist\langle 1\rangle \rightarrow pos\langle 1\rangle = pos\langle 2\rangle
\end{aligned}
$$

where K is an integer logical variable. The pre-condition states that the initial positions are an even distance apart. To read the post-condition, the predicate $K + start\langle 1\rangle \in hist\langle 1\rangle$ holds if and only if the first walk has moved to position $K + start\langle 1\rangle$ at some time in the past; if this has happened, then the two coupled positions must be equal.

Our coupling mirrors the two walks. Each step, we have the walks make symmetric moves by arranging opposite samples. Once the walks meet, we have the walks match each other by coupling the samples to be equal. In this way, if the first walk reaches $start\langle 1\rangle + K$, then the second walk must be at $start\langle 2\rangle - K$ since both walks are coupled to move symmetrically. In this case, the initial condition $start\langle 2\rangle - start\langle 1\rangle = 2K$ gives

$$
pos\langle 1\rangle = start\langle 1\rangle + K = start\langle 2\rangle - K = pos\langle 2\rangle
$$

so the walks meet and continue to share the same position thereafter. This argument requires the starting positions to be an even distance apart so the positions in the two walks always have the same parity; if the two starting positions are an odd distance apart, then the two distributions after T steps have disjoint support and the coupled walks can never meet.

To formalize this argument in pRHL, we handle the loop with the two-sided rule

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

[While] and invariant

$$\Theta \triangleq \begin{cases} |hist\langle 1\rangle| > 0 \land |hist\langle 2\rangle| > 0 \\ K + start\langle 1\rangle \in hist\langle 1\rangle \to pos\langle 1\rangle = pos\langle 2\rangle \\ K + start\langle 1\rangle \notin hist\langle 1\rangle \to pos\langle 2\rangle - pos\langle 1\rangle = \\ \qquad\qquad 2(K - (hd(hist\langle 1\rangle) - start\langle 1\rangle)), \end{cases}$$

where $hd(hist)$ is the first element (the *head*) of the non-empty list *hist*. The last two conditions model the two cases. If the first walk has already visited $K + start\langle 1\rangle$, the walks have already met under the coupling and they must have the same position. Otherwise, the walks have not met. If $d \triangleq hd(hist\langle 1\rangle) - start\langle 1\rangle$ is the (signed) distance the first walk has moved away from its starting location and the two walks are initially $2K$ apart, then the current distance between coupled positions must be $2(K - d)$.

To show the invariant is preserved, we perform a case analysis with [Case]. If $K + start\langle 1\rangle \in hist\langle 1\rangle$ holds then the walks have already met in the past and currently have the same position (by Θ). So, we select the identity coupling in [Sample]:

$$\textsc{Sample}\ \frac{f = \mathrm{id}}{\vdash r \xleftarrow{\$} \mathbf{Flip} \sim r \xleftarrow{\$} \mathbf{Flip} : K + start\langle 1\rangle \in hist\langle 1\rangle \implies r\langle 1\rangle = r\langle 2\rangle}\ .$$

Since $K + start\langle 1\rangle \in hist\langle 1\rangle \to pos\langle 1\rangle = pos\langle 2\rangle$ holds at the start of the loop, we know $pos\langle 1\rangle = pos\langle 2\rangle$ at the end of the loop; since $K + start\langle 1\rangle \in hist\langle 1\rangle$ is preserved by the loop body, the invariant Θ holds.

Otherwise if $K + start\langle 1\rangle \notin h\langle 1\rangle$, then the walks have not yet met and should be mirrored. So, we select the negation coupling with $f = \neg$ in [Sample]:

$$\textsc{Sample}\ \frac{f = \neg}{\vdash r \xleftarrow{\$} \mathbf{Flip} \sim r \xleftarrow{\$} \mathbf{Flip} : K + start\langle 1\rangle \notin hist\langle 1\rangle \implies \neg r\langle 1\rangle = r\langle 2\rangle}$$

To show the loop invariant, there are two cases. If $K + start\langle 1\rangle \in h\langle 1\rangle$ holds after the body, the two walks have just met for the first time and $pos\langle 1\rangle = pos\langle 2\rangle$ holds. Otherwise, the walks remain mirrored: $pos\langle 1\rangle$ increased by $r\langle 1\rangle$ and $pos\langle 2\rangle$ decreased by $r\langle 1\rangle$, so $pos\langle 2\rangle - pos\langle 1\rangle = 2(K + (hd(hist\langle 1\rangle) - start\langle 1\rangle))$ and the invariant Θ is preserved.

Putting it all together, we have the desired judgment:

$$\vdash rwalk \sim rwalk : start\langle 2\rangle - start\langle 1\rangle = 2K \implies K + start\langle 1\rangle \in h\langle 1\rangle \to pos\langle 1\rangle = pos\langle 2\rangle.$$

While this judgment describes a coupling between the position distributions, we need to analyze finer properties of the coupling distribution to apply Theorem 5.16 – namely, we must bound the probability that $pos\langle 1\rangle$ is not equal to $pos\langle 2\rangle$. We will consider how to extract this information in the next chapter.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

5.6.4 Verifying non-relational properties: independence and uniformity

As we have seen, couplings are a natural fit for probabilistic relational properties. Properties describing a single program can also be viewed relationally in some cases, enabling cleaner proofs by coupling. Barthe et al. (2017b) develop this idea to prove *uniformity*, *probabilistic independence*, and *conditional independence*, examples of probabilistic non-relational properties. We briefly sketch their main reductions.

A uniform distribution places equal probability on every value in some range. Given a distribution μ over **State** and an expression e with finite range $\mathcal{S}$ (say, the booleans), e is *uniform* in μ if for all a and a' in $\mathcal{S}$, we have

$$\Pr_{m\sim\mu}[[\![e]\!]m = a] = \Pr_{m\sim\mu}[[\![e]\!]m = a'].$$

When μ is the output distribution of a program c, uniformity follows from the pRHL judgment

$$\forall a, a' \in \mathcal{S}, \vdash c \sim c : (=) \Longrightarrow e\langle 1\rangle = a \leftrightarrow e\langle 2\rangle = a'.$$

This reduction is a direct consequence of Theorem 5.12. Moreover, the resulting judgment is ideally suited to relational verification since it relates two copies of the same program c.

Handling independence is only a bit more involved. Given a distribution μ and expressions e, e' with ranges $\mathcal{S}$ and $\mathcal{S}'$, we say e and e' are *probabilistically independent* if for all $a \in \mathcal{S}$ and $a' \in \mathcal{S}'$, we have

$$\Pr_{m\sim\mu}[[\![e]\!]m = a \wedge [\![e']\!]m = a'] = \Pr_{m\sim\mu}[[\![e]\!]m = a] \cdot \Pr_{m\sim\mu}[[\![e']\!]m = a'].$$

This useful property roughly implies that properties involving e and e' can be analyzed by focusing on e and e' separately. When e and e' are uniformly distributed, independence follows from uniformity of the tuple (e, e') over the product set $\mathcal{S}\times\mathcal{S}'$ so the previous reduction applies. In general, we can compare the distributions of e and e' in two experiments: when both are drawn from the output distribution of a single execution, and when they are drawn from two independent executions composed sequentially. If the expressions are independent, these two experiments should look the same. Concretely, independence follows from the relational judgment

$$\forall a \in \mathcal{S}, a' \in \mathcal{S}', \vdash c \sim c^{(1)}; c^{(2)} : \Phi \Longrightarrow e\langle 1\rangle = a \wedge e'\langle 1\rangle = a' \leftrightarrow$$
$$e^{(1)}\langle 2\rangle = a \wedge e'^{(2)}\langle 2\rangle = a',$$

where $c^{(1)}$ and $c^{(2)}$ are copies of c with variables x renamed to $x^{(1)}$ and $x^{(2)}$ respectively; this construction is also called *self-composition* since it sequentially composes c with itself (Barthe et al., 2011). The pre-condition Φ states that the three copies of each variable are initially equal: $x\langle 1\rangle = x^{(1)}\langle 2\rangle = x^{(2)}\langle 2\rangle$. Handling conditional independence requires a slightly more complex encoding, but the general

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

pattern remains the same: encode products of probabilities by self-composition and equalities by lifted equivalence $(\leftrightarrow)^\sharp$.

These reductions give a simple method to prove uniformity and independence. Other non-relational properties could benefit from a similar approach, especially in conjunction with more sophisticated program transformations in pRHL to relate different copies of the same sampling instruction. Albarghouthi and Hsu (2018a) consider how to automatically construct such coupling proofs by program synthesis and verification techniques.

5.7 Related work

Relational Hoare logics and probabilistic couplings have been extensively studied and (re)discovered in various research communities.

5.7.1 Relational Hoare logics

The logic pRHL is a prime example of a *relational* program logic, which extend standard Floyd-Hoare logics to prove properties about two programs. Benton (2004) first designed a relational version of Hoare logic called RHL to prove equivalence between two (deterministic) programs. Benton used his logic to verify compiler transformations, showing the original program is equivalent to the transformed program. Relational versions of other program logics have also been considered, including an extension of separation logic by Yang (2007) to prove relational properties of pointer-manipulating programs. There is nothing particularly special about relating exactly *two* programs; recently, Sousa and Dillig (2016) give a Hoare logic for proving properties of k executions of the same program for arbitrary k. Barthe et al. (2017a) give an extended logic ×pRHL where judgments also include probabilistic product program simulating two coupled runs of the related programs; in a sense, coupling proofs are probabilistic product programs.

Barthe et al. (2009) extended Benton's work to prove relational properties of probabilistic programs, leading to the logic pRHL. As we have seen, the key technical insight is to interpret the relational post-condition as a probabilistic lifting between two output distributions. Barthe et al. (2009) used pRHL to verify security properties for a variety of cryptographic constructions by mimicking the so-called *game-hopping* proof technique (Shoup, 2004; Bellare and Rogaway, 2006), where the original program is transformed step-by-step to an obviously secure version (e.g., a program returning a random number). Security follows if each transformation approximately preserves the program semantics. Our analysis of the XOR cipher is a very simple example of this technique; more sophisticated proofs chain together dozens of transformations.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

5.7.2 Probabilistic couplings and liftings

Couplings are a well-studied tool in probability theory; readers can consult the lecture notes by Lindvall (2002) or the textbooks by Thorisson (2000) and Levin et al. (2009) for entry points into this vast literature.

Probabilistic liftings were also considered in connection with *probabilistic bisimulation*, a powerful technique for proving equivalence of probabilistic transition systems. Larsen and Skou (1991) were the first to consider a probabilistic notion of bisimulation. Roughly speaking, their definition considers an equivalence relation E on states and requires that any two states in the same equivalence class have the same probability of stepping to any other equivalence class. The construction for arbitrary relations arose soon after, when researchers generalized probabilistic bisimulation to probabilistic simulation; Jonsson and Larsen (1991, Definition 4.3) proposes a *satisfaction relation* using witness distributions, similar to the definition used in pRHL. Desharnais (1999, Definition 3.6.2) and Segala and Lynch (1995, Definition 12) give an alternative characterization without witness distributions, similar to Strassen's theorem (Strassen, 1965); Desharnais (1999, Theorem 7.3.4) observed that both definitions are equivalent in the finite case via the max flow-min cut theorem. Probabilistic (bi)simulation can be characterized logically, i.e., two systems are (bi)similar if and only if they satisfy the same formulas in some modal logic (Larsen and Skou, 1991; Desharnais et al., 2002, 2003; Fijalkow et al., 2017). Deng and Du (2011) survey logical, metric, and algorithmic characterizations of these relations.

While proofs by bisimulation and proofs by coupling are founded on the same mathematical concept, they have been applied to different kinds of target properties. Verification techniques based on bisimulation, for instance, typically focus on possibly large, but finite state systems. In this setting, there are many algorithmic techniques known for constructing these proofs. On the other hand, the main strength of proofs by coupling is that by describing a binary relation between states using a logical formula, the proof technique directly extends to infinite state systems and systems with unknown parameters. At the same time, it appears that the coupling proof technique requires enough structure on the states in order to express possibly infinite relations with a compact logical formula. In contrast to the well-established algorithmic techniques for bisimulation, there has been little research to date on automating proofs of coupling (Albarghouthi and Hsu, 2018a,b). This direction deserves further exploration.

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

5.7.3 Approximate couplings and differential privacy

Differential privacy is a recent, statistical notion of privacy for database queries, proposed by Dwork et al. (2006). This property is essentially a form of sensitivity: pairs of similar input databases should lead to pairs of indistinguishable output distributions. Inspired by pRHL, Barthe et al. (2013) developed an approximate version of the program logic called apRHL to verify this property. The logic apRHL relies on a probabilistic notion of relation lifting but unlike pRHL, this lifting is approximate in a quantitative sense—it approximately models two given distributions with a joint distribution.[4] Much like for probabilistic couplings, the existence of an approximate lifting with a particular support relating two distributions can imply relational properties about the two distributions. For instance, the approximate lifting of the equality relation relates pairs of indistinguishable distributions.

Approximate liftings can be fruitfully viewed as an approximate generalization of probabilistic coupling. Moreover, the proof rules in apRHL immediately suggest a clean method to build approximate couplings. Informally, we can construct approximate couplings much like regular probabilistic couplings, while keeping track of two numeric approximation parameters (ϵ, δ) on the side. To reason about pairs of sampling instructions, we can apply any approximate coupling of the two primitive distributions if we increment (ϵ, δ) by the parameters of the selected coupling; in this way, we can think of (ϵ, δ) as kind of a *cost* that must be paid in order to apply approximate couplings. This idea enables new proofs of differential privacy by coupling, conceptually simpler and more amenable to formal verification than existing arguments (Barthe et al., 2016b,a). The interested reader can consult the thesis (Hsu, 2017) for more on the theory and applications of approximate couplings.

5.7.4 Expectation couplings and the Kantorovich metric

Probabilistic couplings and approximate probabilistic couplings apply to distributions over sets. In some cases, the ground sets may naturally come with a notion of distance. For instance, programs may generate distributions over the real numbers (or vectors) with the Euclidean distance. A classical way of comparing such distributions is with the *Kantorovich metric*, which lifts a distance on the ground space to a distance on distributions over the ground space. This metric is closely related to probabilistic couplings—one way to define the Kantorovich metric is as the minimum average distance over all couplings of the two given distributions. By constructing specific couplings and reasoning about the expected distance, we can upper-bound the

[4] There are several definitions of approximate relation lifting (Barthe and Olmedo, 2013; Olmedo, 2014; Sato, 2016; Albarghouthi and Hsu, 2018b); many of which can be shown equivalent (Barthe et al., 2017c).

https://doi.org/10.1017/9781108770750.006 Published online by Cambridge University Press

Kantorovich metric between two output distributions and derive finer relational properties of probabilistic programs. Barthe et al. (2018b) develop a program logic $\mathbb{E}$pRHL extending pRHL to carry out this kind of reasoning, and establish quantitative relational properties including algorithmic stability of machine learning algorithms and rapid mixing of Markov chains.

References

Albarghouthi, Aws, and Hsu, Justin. 2018a. Constraint-Based Synthesis of Coupling Proofs. In: *International Conference on Computer Aided Verification (CAV), Oxford, England*. To appear.

Albarghouthi, Aws, and Hsu, Justin. 2018b. Synthesizing Coupling Proofs of Differential Privacy. *Proceedings of the ACM on Programming Languages*, **2**(POPL). Appeared at ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Los Angeles, California.

Aldous, David. 1983. Random Walks on Finite Groups and Rapidly Mixing Markov Chains. Pages 243–297 of: *Séminaire de Probabilités XVII 1981/82*. Lecture Notes in Mathematics, vol. 986. Springer-Verlag.

Apt, Krzysztof R. 1981. Ten Years of Hoare's Logic: A Survey–Part I. *ACM Transactions on Programming Languages and Systems*, **3**(4), 431–483.

Apt, Krzysztof R. 1983. Ten years of Hoare's logic: A survey–Part II: Nondeterminism. *Theoretical Computer Science*, **28**(1), 83–109.

Barthe, Gilles, and Olmedo, Federico. 2013. Beyond Differential Privacy: Composition Theorems and Relational Logic for f-Divergences between Probabilistic Programs. Pages 49–60 of: *International Colloquium on Automata, Languages and Programming (ICALP), Riga, Latvia*. Lecture Notes in Computer Science, vol. 7966. Springer-Verlag.

Barthe, Gilles, Grégoire, Benjamin, and Zanella-Béguelin, Santiago. 2009. Formal Certification of Code-Based Cryptographic Proofs. Pages 90–101 of: *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Savannah, Georgia*.

Barthe, Gilles, D'Argenio, Pedro R., and Rezk, Tamara. 2011. Secure Information Flow by Self-Composition. *Mathematical Structures in Computer Science*, **21**(06), 1207–1252.

Barthe, Gilles, Köpf, Boris, Olmedo, Federico, and Zanella-Béguelin, Santiago. 2013. Probabilistic Relational Reasoning for Differential Privacy. *ACM Transactions on Programming Languages and Systems*, **35**(3), 9:1–9:49.

Barthe, Gilles, Fong, Noémie, Gaboardi, Marco, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2016a. Advanced Probabilistic Couplings for Differential Privacy. Pages 55–67 of: *ACM SIGSAC Conference on Computer and Communications Security (CCS), Vienna, Austria*. There is an error in the treatment of advanced composition; please see my thesis for the correction.

Barthe, Gilles, Gaboardi, Marco, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2016b. Proving Differential Privacy via Probabilistic Couplings. Pages 749–758 of: *IEEE Symposium on Logic in Computer Science (LICS), New York, New York*.

Barthe, Gilles, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2017a. Coupling Proofs Are Probabilistic Product Programs. Pages 161–174 of: *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*.

Barthe, Gilles, Espitau, Thomas, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2017b. Proving Uniformity and Independence by Self-Composition and Coupling. Pages 385–403 of: *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Maun, Botswana*. EPiC Series in Computing, vol. 46.

Barthe, Gilles, Espitau, Thomas, Hsu, Justin, Sato, Tetsuya, and Strub, Pierre-Yves. 2017c. ⋆-Liftings for Differential Privacy. Pages 102:1–102:12 of: *International Colloquium on Automata, Languages and Programming (ICALP), Warsaw, Poland*. Leibniz International Proceedings in Informatics, vol. 80. Schloss Dagstuhl–Leibniz Center for Informatics.

Barthe, Gilles, Espitau, Thomas, Gaboardi, Marco, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2018a. An Assertion-Based Program Logic for Probabilistic Programs. In: *European Symposium on Programming (ESOP), Thessaloniki, Greece*.

Barthe, Gilles, Espitau, Thomas, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2018b. Proving Expected Sensitivity of Probabilistic Programs. *Proceedings of the ACM on Programming Languages*, **2**(POPL). Appeared at ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Los Angeles, California.

Bellare, Mihir, and Rogaway, Phillip. 2006. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. Pages 409–426 of: *IACR International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Saint Petersburg, Russia*. Lecture Notes in Computer Science, vol. 4004. Springer-Verlag.

Benton, Nick. 2004. Simple Relational Correctness Proofs for Static Analyses and Program Transformations. Pages 14–25 of: *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Venice, Italy*.

Callahan, David, and Kennedy, Ken. 1988. Compiling Programs for Distributed-Memory Multiprocessors. *The Journal of Supercomputing*, **2**(2), 151–169.

Chatterjee, Krishnendu, Fu, Hongfei, Novotný, Petr, and Hasheminezhad, Rouzbeh. 2016a. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. Pages 327–342 of: *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Saint Petersburg, Florida*.

Chatterjee, Krishnendu, Fu, Hongfei, and Goharshady, Amir Kafshdar. 2016b. Termination Analysis of Probabilistic Programs through Positivstellensatz's. Pages 3–22 of: *International Conference on Computer Aided Verification (CAV), Toronto, Ontario*. Lecture Notes in Computer Science, vol. 9779. Springer-Verlag.

Chatterjee, Krishnendu, Novotný, Petr, and Žikelić, D. 2017. Stochastic Invariants for Probabilistic Termination. Pages 145–160 of: *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*.

Deng, Yuxin, and Du, Wenjie. 2011 (March). *Logical, Metric, and Algorithmic Characterisations of Probabilistic Bisimulation*. Tech. rept. CMU-CS-11-110. Carnegie Mellon University.

Desharnais, Josée. 1999. *Labelled Markov Processes*. Ph.D. thesis, McGill University.

Desharnais, Josée, Edalat, Abbas, and Panangaden, Prakash. 2002. Bisimulation for Labelled Markov Processes. *Information and Computation*, **179**(2), 163–193.

Desharnais, Josée, Gupta, Vineet, Jagadeesan, Radha, and Panangaden, Prakash. 2003. Approximating Labelled Markov Processes. *Information and Computation*, **184**(1), 160–200.

Dijkstra, Edsger W. 1976. *A Discipline of Programming*. Series in Automatic Computation. Prentice Hall.

Dwork, Cynthia, McSherry, Frank, Nissim, Kobbi, and Smith, Adam D. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. Pages 265–284 of: *IACR Theory of Cryptography Conference (TCC), New York, New York*. Lecture Notes in Computer Science, vol. 3876. Springer-Verlag.

Ferrer Fioriti, Luis María, and Hermanns, Holger. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. Pages 489–501 of: *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Mumbai, India*.

Fijalkow, Nathanaël, Klin, Bartek, and Panangaden, Prakash. 2017. Expressiveness of Probabilistic Modal Logics, Revisited. Pages 105:1–105:12 of: Chatzigiannakis, Ioannis, Indyk, Piotr, Kuhn, Fabian, and Muscholl, Anca (eds), *International Colloquium on Automata, Languages and Programming (ICALP), Warsaw, Poland*. Leibniz International Proceedings in Informatics, vol. 80. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz Center for Informatics.

Floyd, Robert W. 1967. Assigning Meanings to Programs. In: *Symposium on Applied Mathematics*. American Mathematical Society.

Hoare, Charles A. R. 1969. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, **12**(10), 576–580.

Hsu, Justin. 2017. *Probabilistic Couplings for Probabilistic Reasoning*. Ph.D. thesis, University of Pennsylvania.

Jones, Cliff B. 2003. The Early Search for Tractable Ways of Reasoning about Programs. *Annals of the History of Computing*, **25**(2), 26–49.

Jonsson, Bengt, and Larsen, Kim Guldstrand. 1991. Specification and Refinement of Probabilistic Processes. Pages 266–277 of: *IEEE Symposium on Logic in Computer Science (LICS), Amsterdam, The Netherlands*.

Kleinberg, Jon, and Tardos, Eva. 2005. *Algorithm Design*. Addison-Wesley.

Kozen, Dexter. 1981. Semantics of Probabilistic Programs. *Journal of Computer and System Sciences*, **22**(3), 328–350.

Larsen, Kim Guldstrand, and Skou, Arne. 1991. Bisimulation through Probabilistic Testing. *Information and Computation*, **94**(1), 1–28.

Levin, David A., Peres, Yuval, and Wilmer, Elizabeth L. 2009. *Markov Chains and Mixing Times*. American Mathematical Society.

Lindvall, Torgny. 2002. *Lectures on the Coupling Method*. Courier Corporation.

McIver, Annabelle, Morgan, Carroll, Kaminski, Benjamin Lucien, and Katoen, Joost-Pieter. 2018. A new proof rule for almost-sure termination. *Proceedings of the ACM on Programming Languages*, **2**(POPL), 33:1–33:28.

O'Hearn, Peter W. 2007. Resources, Concurrency, and Local Reasoning. *Theoretical Computer Science*, **375**(1), 271–307. Festschrift for John C. Reynolds's 70th birthday.

O'Hearn, Peter W., Reynolds, John C., and Yang, Hongseok. 2001. Local Reasoning about Programs That Alter Data Structures. Pages 1–19 of: *International Workshop on Computer Science Logic (CSL), Paris, France*. Lecture Notes in Computer Science, vol. 2142. Springer-Verlag.

Olmedo, Federico. 2014. *Approximate Relational Reasoning for Probabilistic Programs*. Ph.D. thesis, Universidad Politécnica de Madrid.

Reynolds, John C. 2001. Intuitionistic Reasoning about Shared Mutable Data Structure. *Millennial Perspectives in Computer Science*, **2**(1), 303–321.

Reynolds, John C. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. Pages 55–74 of: *IEEE Symposium on Logic in Computer Science (LICS), Copenhagen, Denmark*.

Rudin, Walter. 1976. *Principles of Mathematical Analysis*. Third edn. International Series in Pure and Applied Mathematics. McGraw-Hill.

Sato, Tetsuya. 2016. Approximate Relational Hoare Logic for Continuous Random Samplings. In: *Conference on the Mathematical Foundations of Programming Semantics (MFPS), Pittsburgh, Pennsylvania*.

Segala, Roberto, and Lynch, Nancy A. 1995. Probabilistic Simulations for Probabilistic Processes. *Nordic Journal of Computing*, **2**(2), 250–273.

Shoup, Victor. 2004. *Sequences of Games: A Tool for Taming Complexity in Security Proofs*. Cryptology ePrint Archive, Report 2004/332.

Sousa, Marcelo, and Dillig, Isil. 2016. Cartesian Hoare Logic for Verifying k-Safety Properties. Pages 57–69 of: *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Santa Barbara, California*.

Strassen, Volker. 1965. The Existence of Probability Measures with Given Marginals. *The Annals of Mathematical Statistics*, 423–439.

Thorisson, Hermann. 2000. *Coupling, Stationarity, and Regeneration*. Springer-Verlag.

Yang, Hongseok. 2007. Relational Separation Logic. *Theoretical Computer Science*, **375**(1), 308–334. Festschrift for John C. Reynolds's 70th birthday.