Higher-Order Probabilistic Adversarial Computations: Categorical Semantics and Program Logics

ALEJANDRO AGUIRRE*, Aarhus University, Denmark
GILLES BARTHE, MPI-SP, Germany and IMDEA Software Institute, Spain
MARCO GABOARDI, Boston University, USA
DEEPAK GARG, Max Planck Institute for Software Systems, Germany
SHIN-YA KATSUMATA, National Institute of Informatics, Japan
TETSUYA SATO, Tokyo Institute of Technology, Japan

Adversarial computations are a widely studied class of computations where resource-bounded probabilistic adversaries have access to oracles, i.e., probabilistic procedures with private state. These computations arise routinely in several domains, including security, privacy and machine learning.

In this paper, we develop program logics for reasoning about adversarial computations in a higher-order setting. Our logics are built on top of a simply typed λ-calculus extended with a graded monad for probabilities and state. The grading is used to model and restrict the memory footprint and the cost (in terms of oracle calls) of computations. Under this view, an adversary is a higher-order expression that expects as arguments the code of its oracles. We develop unary program logics for reasoning about error probabilities and expected values, and a relational logic for reasoning about coupling-based properties. All logics feature rules for adversarial computations, and yield guarantees that are valid for all adversaries that satisfy a fixed resource policy. We prove the soundness of the logics in the category of quasi-Borel spaces, using a general notion of graded predicate liftings, and we use logical relations over graded predicate liftings to establish the soundness of proof rules for adversaries. We illustrate the working of our logics with simple but illustrative examples.

CCS Concepts: • **Theory of computation → Semantics and reasoning**.

Additional Key Words and Phrases: Probabilistic programming, semantic models, program logics.

ACM Reference Format:
Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, Shin-ya Katsumata, and Tetsuya Sato. 2021. Higher-Order Probabilistic Adversarial Computations: Categorical Semantics and Program Logics. *Proc. ACM Program. Lang.* 5, ICFP, Article 93 (August 2021), 30 pages. https://doi.org/10.1145/3473598

1 INTRODUCTION

Probabilistic programs occur widely in privacy, security, and other domains where formal guarantees are required. These guarantees are often expressed using expectations, e.g, one may want to prove that the expected value of a randomized algorithm remains close to some deterministic function of its input. This can be established by means of expectation-based methods that originate from

*This work was carried out while A.A. was affiliated to the IMDEA Software Institute and Universidad Politécnica de Madrid

Authors' addresses: Alejandro Aguirre, Aarhus University, Denmark, alejandro@cs.au.dk; Gilles Barthe, MPI-SP, Germany and IMDEA Software Institute, Spain, gbarthe@mpi-sp.org; Marco Gaboardi, Boston University, USA, gaboardi@bu.edu; Deepak Garg, Max Planck Institute for Software Systems, Germany, dg@mpi-sws.org; Shin-ya Katsumata, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan, s-katsumata@nii.ac.jp; Tetsuya Sato, Tokyo Institute of Technology, Japan, tsato@c.titech.ac.jp.

2475-1421/2021/8-ART93
https://doi.org/10.1145/3473598

the works of Kozen [1985] and of Morgan et al. [1996]. Another class of guarantees is concerned with proving the probability of events; e.g., one may want to prove that a randomized algorithm has a small probability of returning an incorrect answer. Such properties are the target of so-called Boolean-based methods, such as the union bound logic of [Barthe et al. 2016b]. These two approaches are traditionally used to reason about properties concerning a single program execution. However, many security and privacy properties are naturally expressed by relating two program executions; we call such properties relational properties. Relational counterparts to expectation-based and Boolean-based methods have been proposed, including the relational expectation-based logic of Barthe et al. [2018], and probabilistic relational Hoare logic [Barthe et al. 2009].

Some of these logics additionally support reasoning about *adversarial computations*, where resource-bounded but otherwise unconstrained adversaries interact with oracles, i.e. probabilistic procedures with private state. These logics view adversaries as uninterpreted procedures, and restrict their power by adding constraints on the memory they can read and write, and on the number of times they can call other procedures. These constraints are captured by a notion of valid adversary, and it is reasonably simple to define proof rules for valid adversaries. The combination of program logics and adversary rules yield powerful frameworks that have been used to reason about many examples, including security of cryptographic constructions [Barthe et al. 2009] and stability of machine learning algorithms [Barthe et al. 2018].

The aforementioned works are developed on top of a core probabilistic imperative language. However, it is often desirable to reason about higher-order programs, either because the programs of interest are written in a higher-order language, or more fundamentally because the programs manipulate higher-order objects. Unfortunately, program logics for higher-order probabilistic languages are not as well understood as their counterparts for imperative languages. One potential reason for this is that denotational semantics of higher-order probabilistic programs have been lacking. Indeed, a classic result by Aumann et al. [1961] shows that the category of Borel spaces is not Cartesian closed, and therefore it cannot be used to interpret programs. Fortunately, recent works propose elegant semantics for higher-order probabilistic programs, such as Probabilistic Coherent Spaces (or PCoh) [Danos and Ehrhard 2011] and Quasi-Borel Spaces (or QBS) [Heunen et al. 2017]. These semantics can be used as a basis for developing program logics, as shown for instance by Sato et al. [2019], who develop unary and relational logics over QBS. However, reasoning in this system is based on an axiomatization of probabilities, and is intricate. Moreover, this system does not support reasoning about state and adversarial computations.

Goals and technical outline. In this paper we set out to develop a *general framework* for designing program logics that reason about resource-constrained adversarial computations in a higher-order probabilistic language. The reasoning principles themselves are fairly natural, and have been considered in the first-order setting before [Barthe et al. 2009], but generalizing them to the higher-order requires addressing the following challenges:

- How can we enforce the restrictions on the adversaries?
- Can we support relational or expectation-based logics?
- How can we formalize the reasoning principles into a *common set of* proof rules? How can we prove these rules sound?
- How can we give a denotational model to these logics?

Program properties in an adversarial setting usually make some assumptions about adversaries by restricting the number of times they can invoke the oracle, and denying them access to the private state of the oracle (formally, the oracle is a function with hidden local state, passed as an argument to the adversary). While this can be achieved by restricting the syntax of adversaries in an ad-hoc manner, a more principled approach would involve using the type system to enforce these

restrictions. Another idea would be to use local state and some sort of separation logic [Tassarotti and Harper 2019], but it is not clear how such features can be added to denotational models for higher-order probabilistic programs. The solution we propose here first involves grading a monad for global state and probabilities by two parameters Σ and k: Σ represents the memory footprint of the computation and k represents the number of oracle calls. Thus, our language has types of the form $\mathsf{T}_{\Sigma,k}(\tau)$ to represent computations with memory footprint Σ and at most k oracle calls. Then, we allow quantification over memory grading, which can be seen as a lightweight form of polymorphism. We then model adversaries as computations of second-order types, e.g. the type $\forall\alpha.(\sigma \to \mathsf{T}_{\alpha,1}(\tau)) \to \mathsf{T}_{\Sigma\cup\alpha,k}(\tau')$ captures an adversary that has access to an oracle of type $\sigma \to \mathsf{T}_{\alpha,1}(\tau)$ and that returns values of type τ'. The grading ensures that the adversary can call the oracle at most k times, and separation between adversary and oracle memories is enforced by a parametricity property derived from the quantification in the type: the adversary can only read and write the memory region Σ; and in particular, it cannot access the private memory of the oracle (denoted α). To our knowledge, this is the first use of this form of parametricity.

On top of this language, we develop a Boolean-based unary logic, an expectation-based unary logic, and a Boolean-based relational logic. The first of these logics can be used to reason about the probability that the output of a program satisfies some assertion. Its judgments are based on generalized Hoare triples of the form $\{\phi\}\ t : \mathsf{T}_{\Sigma,\epsilon}(A)\ \{\{\psi\}\}_\delta$, with the meaning that if the initial state satisfies ϕ, then the final state after running t satisfies ψ with probability at least $1-\delta$. The logic's interpretation is based on a graded monad lifting, which maps the postcondition ψ and the grading δ to an assertion over probability distributions.

Crucially, soundness of this first logic does *not* depend on the concrete definition of the lifting, but only on some algebraic properties of the lifting, so the logic can be generalized. We use this observation to develop a second higher-order program logic for a completely different purpose, namely, proving properties of expectations, similar to Morgan et al. [1996]. In this logic, *assertions are real-valued functions*, as opposed to Boolean-valued assertions of the first logic. Remarkably, most of the proof rules of the two logics are the same, thanks to the common algebraic structure of the underlying liftings. By exploiting this structure, we can get similar proof rules to prove completely different properties with different truth values. We believe that building two differently-valued logics (real-valued and Boolean-valued) from common rules is novel.

Our third logic is a relational logic that can be used to prove properties [Barthe et al. 2016a] of *pairs* of higher-order probabilistic programs using couplings. Again, we exploit the structure of liftings (couplings are particular cases of liftings), this time for relational reasoning.

To each of the three logics we add (structurally very similar) proof rules for reasoning about adversaries. Adversary rules combine all of the features of our framework, and can be used to reason about the interaction of an oracle $\mathcal{O}$, whose code we know, with an adversary $\mathcal{A}$ of which we only know the type. Our type system enforces that the adversary can only call the oracle at most k times and it cannot access the oracle's private memory. The adversary rules of all three logics have similar structure and follow the same underlying pattern: Assuming some invariant about the oracle's private state (which we can discharge in our logics), derive a property of *any* adversary that can call the oracle at most k times. For instance, in the first logic above, the adversary rule says that if the oracle preserves an invariant ϕ with probability at least $1-\delta$, then any adversary calling the oracle at most k times preserves ϕ with probability at least $1-k\delta$.

Next, we define a semantic model for our language and the three logics. Just modeling the language with its higher-order nature and probabilities is nontrivial as explained earlier. Concretely, we model our language in the category **QBS** of Quasi-Borel spaces. We then interpret monadic types using the monad $\mathcal{T}(-) \triangleq M \Rightarrow \mathcal{P}(- \times M)$ for some QBS M of memories, where $\mathcal{P}$ denotes the monad of probability measures over **QBS**.

Next, we wish to build a uniform framework to model our three logics, their different notions of truth-values, different liftings, and both unary and relational reasoning. For this, we build our theory using the notion of *Heyting-valued predicates*, which are maps from a set X to a Heyting algebra Ω. By instantiating Ω differently, we are able to model our different logics. Further, to interpret logics themselves we employ *graded monad liftings* [Katsumata 2014], which map a Heyting-valued predicate over a set X to a Heyting-valued predicate over the set of distributions over X. We also introduce a novel concept of *stateful lifting*, which combines graded monad liftings with the state monad. This gives a categorical semantics of our new Hoare-triple type (c.f. [Nanevski et al. 2008]): "$\{\phi\}t : \mathsf{T}_{\Sigma,\epsilon}(A)\{\{\psi\}\}_\delta$", where ϕ, ψ are Ω-valued predicates, δ is a grading and the whole type specifies properties of probabilities of state transformers. In doing so, we carefully design a categorical framework that unifies qualitative and quantitative assertions using Heyting algebras, and admits interpretations of the triples under a generic graded lifting. Soundness of the different logics follows *uniformly* by suitably instantiating the liftings and the Heyting algebras.

The soundness of the adversary rules needs separate proofs, since we must show that the rules are sound for any term inhabiting the adversary's type. This can usually be done with logical relations, but an approach based on standard logical relations would fail here, since it would not take into account the latent effect of the types and their relation to the invariant. Therefore, we develop a novel logical relation that is parametrized by the invariant we want to preserve, and graded by the probability of failure.

Contributions. In summary, our contributions are the following:

- We design a type system for a higher-order probabilistic language to model adversaries and restrict their capabilities. This is achieved through the use of a monad graded by the memory footprint and the cost the computations, and exploiting parametricity over the memory usage. This novel application of parametricity allows us to enforce a separation between the adversary and the oracle memories in a setting with global state.
- We design three unary and relational logics to reason about probabilistic programs in this setting. We go beyond logics in which assertions are Boolean by also presenting a logic in which assertions are real-valued functions, whose expected value the logic establishes. Assertions in our logics are also graded, to allow us to reason about the probability of failure, or the tightness of bounds. The logics are instances of a generic structure – both in the proof rules and the semantics – showcasing the common structure behind them.
- We introduce a notion of stateful lifting, which is used to interpret the triples in our judgments, from which we can construct a categorical model for the rest of the framework. This model is parametrized by a Heyting algebra of truth values and a graded lifting that interprets assertions. This allows us to have a uniform categorical model which is general enough for all the logics that we present.
- We introduce rules to reason about the interaction between adversaries, from which we only know their type, and oracles. This uses the parametricity above, to show that an invariant is preserved, and moreover it uses the cost restriction on the adversary to compute the grading of the interaction. Soundness of these rules follows from a novel logical relation.

2 ILLUSTRATIVE EXAMPLES

We introduce two illustrative examples, which we use to motivate our modeling of adversaries, and later to showcase the mechanics of our different logics. Our examples are deliberately simple; further examples are discussed in the conclusion and the appendix.[1]

[1]The appendix can be found on the full version of the paper (https://arxiv.org/pdf/2107.01155.pdf)

Pollution attacks against Bloom Filters [Gerbet et al. 2015]. Bloom Filters [Bloom 1970] are probabilistic data structures useful to represent sets efficiently at the cost of a loss in precision. Informally, a Bloom Filter is a data structure with two procedures: a procedure for inserting a value into the current set, and a procedure to query whether a value belongs to the current set. For simplicity, we assume that values are taken from the set $[n] = \{0, \dots, n-1\}$ for some n. A Bloom Filter represents subsets of $[n]$ as an array L of bits of fixed size m. Initially all bits in L are set to 0. The insertion procedure is parametrized by a hash function $H : ([n] \times [\ell]) \to [m]$ sampled uniformly at random, where ℓ is a parameter of the Bloom Filter. The procedure $\mathsf{insert}(x)$ updates to 1 the value of the array at positions $h(x,1), \dots, h(x,\ell)$. The procedure $\mathsf{member}(x)$ computes $h(x,1), \dots, h(x,\ell)$ and returns 1 if all these bits are set to 1, and 0 otherwise. The main advantage of Bloom Filters is their space-efficiency over other classical data structures for sets. Their downside is that membership queries may possibly return a false positive due to a hash collision. Therefore, an adversary may attempt to pollute the Bloom Filter in order to trigger false positives [Gerbet et al. 2015]. Here we consider a very simple form of pollution attacks, where an adversary adaptively performs insertion queries with the goal to set to 1 a maximal number of bits of the Bloom Filter. Since the adversary is probabilistic, we use the *expected* number of bits set to 1 as a measure of the adversary's success. Assuming that the Bloom Filter is initially empty, i.e. all bits are set to 0, one can prove that for *every* adversary $\mathcal{A}$ making at most k queries to the insertion oracle, the expected number of bits set to 1 after the adversary returns is upper bounded by $m \cdot (1 - ((m-1)/m)^{\ell \cdot k})$.

We model this example in a simply typed calculus enriched with graded monadic type constructors. Concretely, we model adversaries carrying a pollution attack against a Bloom Filter as computations $\mathcal{A}$ of type $\forall\alpha.([n] \to \mathsf{T}_{\alpha,1}(\mathbb{U})) \to \mathsf{T}_{\alpha \cup \Sigma, k}(\mathbb{U})$ where $\mathbb{U}$ is the unit type and by abuse of notation we view $[n]$ as a type. The intended argument of the adversary is the insertion oracle. The monadic type $\mathsf{T}_{\alpha,1}(\tau)$ should be seen as stateful probabilistic computations that can read and write to the set of locations α (but not others) and have cost 1. Therefore, the grading ensures that each oracle call has cost 1, and that the adversary can make at most k calls to the oracle. The universal quantification on α ensures that the adversary can only read and write locations in Σ, and that its effect on other memory locations like the $L[i]$s is only indirect, through calls to its oracle.

We assume that hash functions are implemented as random oracles, i.e. stateful probabilistic functions that lazily sample their output when queried with a fresh input. The pseudo-code of the insertion oracle insert is deferred to Section 4.2. Under this modeling we upper bound the success of pollution attacks via the judgment:

$$\bullet \mid \mathcal{A} : \tau \mid \bullet \mid \bullet \vdash \{m \cdot (1 - ((m-1)/m)^{\ell \cdot k})\}\ \mathcal{A}\ \mathsf{insert} : \mathsf{T}_{\Sigma \cup \{r,L,h\},k}(\mathbb{U})\ \left\{\left\{\textstyle\sum_{i=0}^{m-1} L[i]\right\}\right\}$$

where $\tau \triangleq \forall\alpha.([n] \to \mathsf{T}_{\alpha,1}(\mathbb{U})) \to \mathsf{T}_{\alpha \cup \Sigma, k}(\mathbb{U})$, insert is the insertion oracle, and $\{r, L, h\}$ are the memory locations used by the oracle. The adversary, represented by the variable $\mathcal{A}$, is declared in a special *adversary context*. The other contexts for grading variables, standard variables and logical assumptions are empty (the contexts are explained in Sections 3 and 4). The statement on the right hand side of the turnstile can be seen as a generalized Hoare triple, given by two assertions (between curly braces) and a program, as in Hoare Type Theory [Nanevski et al. 2008]. We have a generic syntax of judgments and a generic set of generic inference rules, that can later be instantiated to different notions of assertions and different interpretations. For the specific instantiation used here (Section 4.2), the assertions are *quantities* – maps from states to the non-negative reals – that are known as the pre-expectation (the one on the left) and the post-expectation (on the right), respectively. The interpretation of such a statement is that the *expected* value of the post-expectation over the output distribution of the program is upper bounded by the pre-expectation.

We note that pollution attacks are a very simple example. More advanced attacks are considered by Clayton et al. [2019]; Naor and Yogev [2019], who develop an elaborate theory of Bloom Filters and probabilistic data structures under adversarial environments.

PRF/PRP Switching Lemma. The PRF/PRP Switching Lemma [Impagliazzo and Rudich 1989] is a classical tool in provable security. Let $\{0,1\}^l$ denote the set of bitstrings of length l. The lemma states that the probability of a bounded adversary to distinguish between a pseudo-random function (PRF) and a pseudo-random permutation (PRP) is upper bounded by $k(k+1)/2^{l+1}$ where k is the maximal number of calls allowed to the adversary. The PRF/PRP Switching Lemma is a popular benchmark for computer-aided cryptography, so multiple formalizations are available, e.g. [Barthe et al. 2009].

We model the adversary as a computation of type $\forall\alpha.(\{0,1\}^l \to \mathsf{T}_{\alpha,1}(\{0,1\}^l)) \to \mathsf{T}_{\Sigma\cup\alpha,k}(\{0,1\})$ where Σ models the private memory of the adversary. Similar to the case of Bloom filters, we follow a lazy modeling of PRF and PRP. The pseudo-code of PRF and PRP is given below:

$$PRF(x_1 : \{0,1\}^l) \triangleq \text{if } x_1 \notin \text{dom } L_1 \text{ then } \{z_1 = \mathsf{Unif}(\{0,1\}^l); \qquad L_1[x_1] := z_1\}; \text{return } L_1[x_1]$$

$$PRP(x_2 : \{0,1\}^l) \triangleq \text{if } x_2 \notin \text{dom } L_2 \text{ then } \{z_2 = \mathsf{Unif}(\{0,1\}^l \setminus (\text{im } L_2)); L_2[x_2] := z_2\}; \text{return } L_2[x_2]$$

We show that for every adversary $\mathcal{A}$ with the aforementioned type, the statistical distance between $\mathcal{A}$ *PRF* and $\mathcal{A}$ *PRP* is upper bounded by $k(k+1)/2^{l+1}$ using an approximate relational logic (i.e., a logic that can prove approximations rather than equalities). We establish the following judgment:

$$\bullet \mid \mathcal{A} : \tau \mid \bullet \mid \bullet \vdash \{\mathbf{s}_1 = \mathbf{s}_2\}\, \mathcal{A}\ PRF : \mathsf{T}_{\Sigma\cup\{L_1\},k}(\{0,1\}) \sim \mathcal{A}\ PRP : \mathsf{T}_{\Sigma\cup\{L_2\},k}(\{0,1\})\ \{\{\mathbf{s}_1 = \mathbf{s}_2 \wedge \mathbf{v}_1 = \mathbf{v}_2\}\}_{k(k+1)/2^{l+1}}$$

where $\tau \triangleq \forall\alpha.(\{0,1\}^l \to \mathsf{T}_{\alpha,1}(\{0,1\}^l)) \to \mathsf{T}_{\Sigma\cup\alpha,k}(\{0,1\})$. This judgment has the following interpretation: if we have two memories $\mathbf{s}_1, \mathbf{s}_2$ that are equal and we run the computation on the left and the computation on the right with input memories $\mathbf{s}_1$ and $\mathbf{s}_2$ respectively, then the output distributions are going to be close, and their statistical distance is upper bounded by $k(k+1)/2^{l+1}$.

Although our proof uses an approximate logic, there is an alternative proof that uses an exact relational logic, and the Union Bound logic. The latter proof uses the so-called up-to-bad technique, and defaults to the union bound logic to prove that the probability of collisions in a PRF is upper bounded by $k(k+1)/2^{l+1}$. This is captured in the union bound logic by the judgment:

$$\bullet \mid \mathcal{A} : \tau \mid \bullet \mid \bullet \vdash \{|dom\ L_1| = 0\}\, \mathcal{A}\ PRF : \mathsf{T}_{\Sigma\cup\{L_1\},k}(\{0,1\})\{\{|dom\ L_1| = |im\ L_1|\}\}_{k(k+1)/2^{l+1}}$$

This specification has the same syntax as the specification of the pollution attacks, but uses different notions of predicates and a different intrepretation (but crucially, the same set of inference rules). The assertions are Boolean predicates, and the interpretation of this judgment is that if the initial state satisfies $|dom\ L_1| = 0$, then the final state satisfies $|dom\ L_1| = |im\ L_1|$ with probability $1 - \frac{k(k+1)}{2^{l+1}}$. In other words, the judgment behaves as a Hoare triple that has some probability of failure.

3 LANGUAGE

We consider a core language that models higher-order, stateful, probabilistic computations over algebraic datatypes.

Syntax. The language combines the usual constructs of λ-calculus and monadic constructs. Monadic computations are introduced and composed by unit and let. In addition, we have operations for sampling from a distribution in a set $\mathcal{D}$ of base distributions, and for reading or writing at a location in a set Loc of locations. We also consider a primitive skip operation that represents an empty computation (we could also define skip as $\mathsf{unit}(*)$, where $*$ is the sole inhabitant of the unit type), and a primitive mfold for nesting monadic computations (the reason why we make mfold monadic will become apparent in the next paragraph, when typing is considered). Finally,

for technical reasons that will become apparent when defining the logic, we distinguish between adversarial variables and standard variables. Formally, the terms of the language are given by the following grammar:

$$t, u ::= x \mid \mathcal{A} \mid * \mid 0 \mid S\ u \mid \lambda x.u \mid t\ u \mid \langle t, u\rangle \mid \mathsf{if}\ t\ \mathsf{then}\ u_1\ \mathsf{else}\ u_2 \mid \pi_1(t) \mid \pi_2(t) \mid$$
$$\mathsf{read}\ a \mid a := u \mid \mathsf{skip} \mid \mathsf{unit}(t) \mid \mathsf{let}\ x = t\ \mathsf{in}\ u \mid \mathsf{mfold}\ t\ u_1\ u_2 \mid \mathsf{sample}(\nu)$$

where x ranges over variables, $\mathcal{A}$ ranges over adversary variables, a ranges over a set Loc of memory locations and ν ranges over a set $\mathcal{D}$ of distribution symbols. We assume that each distribution has arity $\tau_{\nu,1} \times \cdots \times \tau_{\nu,|\nu|} \to \sigma_\nu$, that accounts for the parameters of the distribution. The meaning of the expressions is standard, except for the monadic fold operation for naturals, which sequences computations in the monadic step, that is:

$$\mathsf{mfold}\ 0\ t\ u = t \qquad \mathsf{mfold}\ (S\ n)\ t\ u = \mathsf{let}\ x = (\mathsf{mfold}\ n\ t\ u)\ \mathsf{in}\ u\ x$$

Syntactic sugar. In our examples we use some syntactic sugar to simplify the code. Concretely, we will write $x = t; u$ instead of $(\lambda x.u)\ t$, $l := t$ instead of $\mathsf{let}\ \_\ \mathsf{in}\ l := t$ (i.e., we do not bind the returned value) and $\mathsf{inc}\ l$ instead of $\mathsf{let}\ y = \mathsf{read}\ l\ \mathsf{in}\ l := y + 1$, where we assume y is a free variable.

Effects. We use a type-and-simple effect system to model the memory footprint and oracle complexity of computations. We model the memory footprint as (an overapproximation of) the set Σ of memory locations read and written by a computation. In addition, our effect system supports abstract effects and effect polymorphism. These are used essentially to model adversaries. The grading k tracks how many times an adversary calls its oracles. For simplicity, we use a single natural number for tracking oracle calls; however, it is possible to track oracle calls more finely by having a number per oracle. Semantically, effects form an ordered commutative monoid: memory effects are modeled using $(\mathbb{P}(\mathsf{Loc} \cup \mathcal{R}), \emptyset, \cup, \subseteq)$, where $\mathcal{R}$ is a set of memory regions, $\mathbb{P}$ is the powerset operator, and cost is modeled using $(\mathbb{N}, 0, +, \leq)$.

Types. Our language is essentially simply typed. As base types we consider the unit type $\mathbb{U}$, booleans $\mathbb{B}$, and natural numbers, which are indexed by either a constant natural number K or by infinity, to indicate an upper bound on the inhabitants of the type. We will simply write $\mathbb{N}$ for $\mathbb{N}[\infty]$. On top of this we consider extended computations, which are given a type $\mathsf{T}_{\Sigma,k}(\tau)$. Here, τ is the return type, and Σ, k is a grading that accounts for the memory effect and cost of the computation. We assume all locations in memory contain the same type $\mathbb{V}$. We keep this abstract in the current presentation, but we will instantiate it to a concrete type (e.g. $\mathbb{N}, \mathbb{B}, \ldots$) in the examples. Finally, we include a type $\mathbb{M}$ for memories. These cannot be explicitly manipulated in the language, but are used in specifications, see later in the section.

Formally, the set of types is given by the following syntax:

$$\tau, \sigma ::= B \mid \mathbb{B} \mid \mathbb{N}[K] \mid \mathbb{M} \mid \mathbb{U} \mid \mathbb{V} \mid \tau \to \sigma \mid \tau \times \sigma \mid \mathsf{T}_{\Sigma,k}(\tau) \mid \forall \alpha.\tau$$

where B ranges over a set of base types, K ranges over natural numbers and the expression Σ is built from region variables and memory locations. Note that bounded natural types $\mathbb{N}[K]$ are used to compute the grading of monadic folds.

Type system. A typing judgment $\Xi; \Delta; \Gamma \vdash t : \tau$ is a relation between contexts, terms and types. Contexts are triples of the form $\Xi; \Delta; \Gamma$, where Ξ is a grading context, Δ is an adversary context, and Γ is a variable context. A grading context Ξ is a collection of variables $\alpha, \beta, \cdots$ representing the memory regions manipulated by the computation. The adversary and variable contexts are functions from a finite set of variables (adversary and standard, respectively). For a context Γ and n distinct variables x_i such that $x_i \notin \mathrm{dom}\,\Gamma$, by $\Gamma, x_1 : \tau_1, \cdots, x_n : \tau_n$ we mean the context obtained by extending Γ with $x_1 : \tau_1 \cdots x_n : \tau_n$. We use a similar notation for Δ.

$$\frac{}{\Xi;\Delta;\Gamma \vdash \star : \mathbb{U}}\ \textsf{Star} \qquad \frac{}{\Xi;\Delta;\Gamma \vdash 0 : \mathbb{N}[0]}\ \textsf{Zero} \qquad \frac{\Xi;\Delta;\Gamma \vdash t : \mathbb{N}[K]}{\Xi;\Delta;\Gamma \vdash S\ t : \mathbb{N}[K+1]}\ \textsf{Succ}$$

$$\frac{\Delta;\Gamma \vdash t : \mathbb{N}[K] \qquad \Delta;\Gamma \vdash u_1 : \mathsf{T}_{\Sigma,k}(\tau) \qquad \Delta;\Gamma \vdash u_2 : \tau \to \mathsf{T}_{\Sigma',k'}(\tau)}{\Delta;\Gamma \vdash \mathsf{mfold}\ t\ u_1\ u_2 : \mathsf{T}_{\Sigma\cup\Sigma',k+K\cdot k'}(\tau)}\ \textsf{Fold}$$

$$\frac{\Xi;\Delta;\Gamma \vdash t : \tau}{\Xi;\Delta;\Gamma \vdash \mathsf{unit}(t) : \mathsf{T}_{\emptyset,0}(\tau)}\ \textsf{Unit} \qquad \frac{\Xi;\Delta;\Gamma \vdash t_1 : \mathsf{T}_{\Sigma_1,k_1}(\tau) \qquad \Xi;\Delta;\Gamma, x:\tau \vdash t_2 : \mathsf{T}_{\Sigma_2,k_2}(\sigma)}{\Xi;\Delta;\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 : \mathsf{T}_{\Sigma_1\cup\Sigma_2\cup\mathbf{Eff}(\tau),k_1+k_2}(\sigma)}\ \textsf{Bind}$$

$$\frac{a \in \mathsf{Loc}}{\Xi;\Delta;\Gamma \vdash \mathsf{read}\ a : \mathsf{T}_{\{a\},0}(\mathbb{V})}\ \textsf{Read} \qquad \frac{\Xi;\Delta;\Gamma \vdash u : \mathbb{V} \qquad a \in \mathsf{Loc}}{\Xi;\Delta;\Gamma \vdash a := u : \mathsf{T}_{\{a\},0}(\mathbb{U})}\ \textsf{Write} \qquad \frac{}{\Xi;\Delta;\Gamma \vdash \mathsf{skip} : \mathsf{T}_{\emptyset,0}(\mathbb{U})}\ \textsf{Skip}$$

$$\frac{\Xi;\Delta;\Gamma \vdash t_i : \tau_{\nu,i} \qquad (\forall 1 \le i \le |\nu|)}{\Xi;\Delta;\Gamma \vdash \mathsf{sample}(\nu(t_1,\ldots,t_{|\nu|})) : \mathsf{T}_{\emptyset,0}(\sigma_\nu)}\ \textsf{Sample} \qquad \frac{\Xi;\Delta;\Gamma \vdash t : \tau' \qquad \Xi \vdash \tau' \preceq \tau}{\Xi;\Delta;\Gamma \vdash t : \tau}\ \textsf{Subtype}$$

$$\frac{\Xi,\alpha;\Delta;\Gamma \vdash t : \tau \qquad \alpha \notin FV(\Delta;\Gamma)}{\Xi;\Delta;\Gamma \vdash t : \forall\alpha.\tau}\ \textsf{ForAll-I} \qquad \frac{\Xi;\Delta;\Gamma \vdash t : \forall\alpha.\tau \qquad \Sigma \in \mathbb{P}(\mathsf{Loc})}{\Xi;\Delta;\Gamma \vdash t : \tau[\Sigma/\alpha]}\ \textsf{ForAll-E}$$

$$\frac{(\mathcal{A} : \forall\alpha.(\sigma \to \mathsf{T}_{\alpha,k}(\tau)) \to \mathsf{T}_{\alpha\cup\Sigma,k'}(\tau')) \in \Delta \qquad \Xi,\alpha;\Delta;\Gamma \vdash t : \sigma \to \mathsf{T}_{\Sigma',k}(\tau)}{\Xi;\Delta;\Gamma \vdash \mathcal{A}\ t : \mathsf{T}_{\Sigma\cup\Sigma',k'}(\tau')}\ \textsf{Adv}$$

$$\frac{\begin{array}{c}(\mathcal{A} : \forall\alpha.(\sigma \to \mathsf{T}_{\alpha,k}(\tau)) \to \mathsf{T}_{\alpha\cup\Sigma,k'}(\tau')) \in \Delta \\ \Xi;\Delta;\Gamma \vdash t : \mathsf{T}_{\Sigma_1,k_1}(\tau') \qquad \Xi;\Delta\backslash\mathcal{A};\bullet \vdash u : \forall\alpha.(\sigma \to \mathsf{T}_{\alpha,k}(\tau)) \to \mathsf{T}_{\alpha\cup\Sigma,k'}(\tau')\end{array}}{\Xi;\Delta\backslash\mathcal{A};\Gamma \vdash t[u/\mathcal{A}] : \mathsf{T}_{\Sigma_1,k_1}(\tau')}\ \textsf{Adv-Inst}$$

Fig. 1. Selected typing rules

Typing rules are presented in Figure 1. Many rules are standard so we focus on the remaining rules. The read and the write rules assume that locations store values of type $\mathbb{V}$. The effect of a read or write is the location itself. The unit and bind rules act on the grading as the unit and multiplication of the monoid from which the gradings are taken, but the bind rule also adds the effect $\mathbf{Eff}(\tau)$ of the type τ encapsulated by the monad. This will be important for proving soundness of the adversary rules, and it is defined as:

$$\mathbf{Eff}(B) \triangleq \emptyset \qquad \mathbf{Eff}(\tau \to \sigma) \triangleq \mathbf{Eff}(\sigma) \qquad \mathbf{Eff}(\mathsf{T}_{\Sigma,k}(\tau)) \triangleq \Sigma \cup \mathbf{Eff}(\tau) \qquad \mathbf{Eff}(\forall\alpha.\tau) = \mathbf{Eff}(\tau[\emptyset/\alpha])$$

Monadic fold (for natural numbers) defines an iterator. It receives a natural number bounded by K, a computation u_1 with cost k for the zero case, and a computation u_2 with cost k for the successor case. The operational semantics imply that u_2 will be run at most K times, and u_1 will be run exactly once, so we can give a bound $K \cdot k' + k$ on the total cost. The rules for quantifier introduction and elimination are a lightweight version of effect polymorphism. We can quantify over any grading in Ξ that does not appear free in Γ and Δ, and we can instantiate a quantifier to any concrete memory region. Since this does not actually have any computational content, we choose to not reflect these rules in the term.

The adversary rule allows applying an adversary variable to an expression with matching type. The instantiation rule for adversary variables substituting an adversary variable by a *closed* expression of the same type. This is the only distinction between standard and adversary variables – adversary variables represent closed expressions, while standard variables represent arbitrary expressions. The reason for making this distinction will become clear when we describe logics.

Types are ordered by subtyping $\tau' \preceq \tau$, which is used in the [Subtype] rule. Subtyping is mostly standard. On the type $\mathsf{T}_{\Sigma,k}(\tau)$, subtyping allows increasing k, Σ and weakening τ. The rules for subtyping can be found in the appendix.

Expressions about memories. In previous work [Aguirre et al. 2017], the terms appearing in logical assertions and the terms (i.e., the programs) they specify about are derived from the same grammar. In the current setting, program specifications contain distinguished variables representing the state, because they need to be able to refer explicitly to initial or final states and their contents, but we do not want programs to have this capability. Therefore, terms appearing in logical assertions will be derived from a grammar that extends the grammar of programs:

$$\tilde{t}, \tilde{u} ::= \ldots \mid \tilde{t}[a] \mid \tilde{t}[a \mapsto \tilde{u}]$$

Here, $\tilde{t}[a]$ denotes the contents of state $\tilde{t}$ at location a, $\tilde{t}[a \mapsto \tilde{u}]$ denotes the state resulting by replacing the contents of location a in $\tilde{t}$ by $\tilde{u}$, and the ellipsis contains all the other term constructors. The (obvious) typing rules for these new constructs can be found in the appendix.

4 HIGHER-ORDER UNARY LOGICS

In this section, we describe two program logics for our language. Both use the same syntactic proof rules derived from a common template, but they differ significantly in their semantics and apply to very different verification problems. The first one is a higher-order Union Bound Logic, in the line of [Barthe et al. 2016b]. This logic allows proving postconditions (for probabilistic computations) that may not hold with an explicit "error" probability δ. The second one is a higher-order expectation logic, in the line of [Barthe et al. 2018; Kozen 1985; Morgan et al. 1996]. Instead of specifying programs with qualitative assertions (that can either be true or false), this logic uses *quantitative* assertions ranging over the non-negative reals. The logic can be used to prove bounds on the expected values of quantitative postconditions. Both logics have adversaries and state.

4.1 Higher-Order Union Bound Logic

The syntax of our union bound logic (HO-UBL) is shown below. *Propositions* ϕ, ψ are standard (intuitionistic) HOL formulas over terms. Quantifiers range over these terms. In contrast, *assertions* P, Q denote pre- and post-conditions that relate the language's expressions *and* the current heap state. R and f denote atomic propositions and atomic assertions, respectively. $\langle\phi\rangle$ is an injection of propositions into assertions. The connectives $\sqcap$ and $\sqcup$ respectively denote conjunction ($\wedge$) and disjunction ($\vee$) at the level of assertions.[2] [3] To refer to the state, assertions P, Q may contain a distinguished variable $\mathbf{s}$, which stands for the current state. Similarly, propositions ϕ, ψ can contain a distinguished variable $\mathbf{v}$ that stands for (the value of) the term being verified.

$$\begin{array}{lll}
\text{Propositions} & \phi, \psi & ::= R(t_1, \ldots, t_n) \mid \top \mid \bot \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \Rightarrow \psi \mid \neg\phi \mid \forall x : \sigma.\phi \mid \exists x : \sigma.\phi \\
\text{Assertions} & P, Q & ::= f(\tilde{t_1}, \ldots, \tilde{t_n}) \mid \top \mid \bot \mid \langle\phi\rangle \mid P \sqcup Q \mid P \sqcap Q \\
\text{Assumptions} & \Psi & ::= \bullet \mid \Psi, \psi \\
\text{Judgments} & \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash \phi & \\
& \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash P \Rrightarrow Q & \\
& \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash t : \sigma\{\phi\} & \\
& \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta &
\end{array}$$

[2] There is a reason for using different symbols for these connectives in propositions and assertions: In the expectation logic (Section 4.2), we want to reuse the same syntax, but give assertions *quantitative* interpretations while retaining the Boolean interpretations for propositions. Using different symbols for the connectives prevents confusion there.

[3] We can add quantifiers $\forall$, $\exists$ to assertions, but we elide them here. Our examples only use *finite* quantification in assertions, which can be encoded using $\sqcap$ and $\sqcup$.

The logic has four judgments that rely on four contexts—Ξ, Δ and Γ that were described earlier—and the new context Ψ, which contains logical assumptions (propositions) ranging over the variables in Δ and Γ. Since most of our rules do not modify or read the contexts and Ξ and Δ, we omit them from most of the discussion below. This simplifies the judgments, e.g., we write $\Gamma \mid \Psi \vdash \phi$ instead of $\Xi \mid \Delta \mid \Gamma \mid \Psi \vdash \phi$.

The first judgment $\Gamma \mid \Psi \vdash \phi$ is HOL's standard entailment judgment. It means that the proposition ψ holds for all typed instantiations of the variables in Γ satisfying all propositions in Ψ. The second judgment $\Gamma \mid \Psi \vdash P \Rrightarrow Q$ is entailment of assertions; it means that the assertion P entails the assertion Q for all typed instantiations of Γ satisfying Ψ.

The third judgment $\Gamma \mid \Psi \vdash t : \sigma\{\phi\}$ means that for all typed instantiations of Γ satisfying Ψ, the term t (of type σ) satisfies $\phi[t/\mathbf{v}]$. (Recall that $\mathbf{v}$ is a distinguished variable.) In other words, the judgment specifies a property ϕ of the term t being verified. This judgment's proof rules are directed by the syntax of t and are taken as-is from the prior logic UHOL [Aguirre et al. 2017]. The work on UHOL also shows that these syntax-directed rules are sound and complete relative to HOL: $\Gamma \mid \Psi \vdash t : \sigma\{\phi\}$ iff $\Gamma \mid \Psi \vdash \phi[t/\mathbf{v}]$. We reproduce these rules in the appendix.

The fourth judgment $\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta$ is new to our logic. It specifies a pre-condition P and a post-condition Q for a *monadic computation* t of type $\mathsf{T}_{\Sigma,k}(\sigma)$. Recall that a monadic computation of this type is stateful and probabilistic — it takes a state and produces a *distribution* on results of type σ and final states. The judgment means that, for any instantiation of Γ satisfying all propositions in Ψ, starting the execution of t in any state m that satisfies $P[m/\mathbf{s}]$, the final state m' and result t' (of type σ) satisfy $Q[m'/\mathbf{s}][t'/\mathbf{v}]$ *with probability at least* $1-\delta$. Additionally, only locations in the set Σ are modified. In other words, the judgment represents a standard Hoare-triple for stateful computations, but with a small twist: the postcondition may not hold with an error probability δ. The semantics of this judgment is defined by lifting standard Hoare triples to distributions (Section 6.5).

Formally, the pre-condition P can contain the free variable $\mathbf{s} : \mathbb{M}$ (where $\mathbb{M}$ is the type of memories), while Q can contain the variables $\mathbf{s} : \mathbb{M}$ and $\mathbf{v} : \sigma$. Additionally, both may mention variables from Γ and the elided context Δ. The restriction that only locations in Σ be modified during t's reduction is needed for handling adversaries as we explain soon. The judgment also does not make use of the grade k in the type $\mathsf{T}_{\Sigma,k}(\sigma)$; this grade is also used for handling adversaries.

Monadic rules. Figure 2 presents the main rules of the fourth judgment. As before, we omit the contexts Ξ and Δ; these transfer unchanged from the conclusion to the premises in all rules. All our rules are syntax-directed. The rule UNIT-U applies to the term $\mathsf{unit}(t)$, which returns the term t without modifying the state with probability 1. The rule just restates this differently. Formally, if t satisfies ϕ (first premise), then executing $\mathsf{unit}(t)$ from a state satisfying P results in a state and return term that satisfy $\langle\phi\rangle \sqcap P$. The probability of this *not* happening is 0. (The premise $\Gamma, \mathbf{s} : \mathbb{M} \vdash P$ just means that the assertion P is a well-formed predicate over the typed variables in $\Gamma, \mathbf{s} : \mathbb{M}$.)

The rule MLET-U for the monadic bind is a generalization of the usual sequencing rule of Hoare logic. The error probabilities δ and δ' are summed in the conclusion. This is easy to see: From the first premise, with probability at least $1-\delta$, the postcondition Q of t holds and, hence, from the second premise, with probability at least $(1-\delta)-\delta'$, the postcondition R holds. Hence, the error probability is at most $\delta + \delta'$.

The rules READ-U and WRITE-U propagate heap changes backwards, as in standard Hoare logic. We also have the rule MCASE-U for conditionals of monadic type. Again, this rule follows the rule for conditionals in Hoare logic. The rule MFOLD-U applies to $\mathsf{mfold}\ n\ t_1\ (\lambda x.t_2)$. Here, K is a bound on the number of iterations (first premise). The error probability in the conclusion is the error probability k' of the iteration's body scaled by K.

$$\frac{\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta \qquad \Gamma, x:\tau \mid \Psi \vdash \{Q[x/\mathbf{v}]\}u : \mathsf{T}_{\Sigma',k'}(\sigma)\{\{R\}\}_{\delta'} \qquad x \notin R}{\Gamma \mid \Psi \vdash \{P\}\mathsf{let}\ x = t\ \mathsf{in}\ u : \mathsf{T}_{\Sigma\cup\Sigma'\cup\mathbf{Eff}(\tau),k+k'}(\sigma)\{\{R\}\}_{\delta+\delta'}}\ \text{MLET-U}$$

$$\frac{\Gamma \mid \Psi \vdash t : \tau\{\phi\} \qquad \Gamma, \mathbf{s} : \mathbb{M} \vdash P}{\Gamma \mid \Psi \vdash \{P\}\mathsf{unit}(t) : \mathsf{T}_{\emptyset,0}(\tau)\{\{\langle\phi\rangle \sqcap P\}\}_0}\ \text{UNIT-U} \qquad \frac{}{\Gamma \mid \Psi \vdash \{P[\mathbf{s}[a]/\mathbf{v}]\}\mathsf{read}\ a : \mathsf{T}_{\{a\},0}(\mathbb{V})\{\{P\}\}_0}\ \text{READ-U}$$

$$\frac{\Xi;\Gamma \vdash t : \mathbb{V}}{\Gamma \mid \Psi \vdash \{P[\mathbf{s}[a \mapsto t]/\mathbf{s}]\}a := t : \mathsf{T}_{\{a\},0}(\mathbb{U})\{\{P\}\}_0}\ \text{WRITE-U}$$

$$\frac{\Gamma \vdash b : \mathbb{B} \qquad \Gamma \mid \Psi, b = \mathbf{tt} \vdash \{P_1\}t_1 : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta \qquad \Gamma \mid \Psi, b = \mathbf{ff} \vdash \{P_2\}t_2 : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta}{\Gamma \mid \Psi \vdash \{\langle b = \mathbf{tt}\rangle \sqcap P_1) \sqcup (\langle b = \mathbf{ff}\rangle \sqcap P_2)\}\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta}\ \text{MCASE-U}$$

$$\frac{\begin{array}{c}\Gamma \vdash n : \mathbb{N}[K] \\ \Gamma \mid \Psi \wedge n = 0 \vdash \{P\}t_1 : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta \qquad \Gamma, x:\tau \mid \Psi \wedge n \neq 0 \vdash \{Q\}t_2 : \mathsf{T}_{\Sigma',k'}(\tau)\{\{Q\}\}_{\delta'}\end{array}}{\Gamma \mid \Psi \vdash \{P\}\mathsf{mfold}\ n\ t_1\ (\lambda x.t_2) : \mathsf{T}_{\Sigma\cup\Sigma',k+K\cdot k'}(\tau)\{\{Q\}\}_{\delta+K\cdot\delta'}}\ \text{MFOLD-U}$$

Fig. 2. Monadic proof rules of our higher-order union bound logic. These rules are reused for the higher-order expectation logic with a different interpretation of $\langle\phi\rangle$, $\sqcap$, and $\sqcup$.

$$\frac{\Gamma \mid \Psi \vdash P \Rrightarrow P' \qquad \Gamma \mid \Psi \vdash \{P'\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q'\}\}_{\delta'} \qquad \Gamma \mid \Psi \vdash Q' \Rrightarrow Q \qquad \delta' \leq \delta}{\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta}\ \text{CONSEQ-U}$$

$$\frac{\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta \qquad \Gamma \mid \Psi \vdash \{P'\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta}{\Gamma \mid \Psi \vdash \{P \sqcup P'\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta}\ \text{OR-PRE-U}$$

$$\frac{\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta \qquad \Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q'\}\}_{\delta'}}{\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q \sqcap Q'\}\}_{\delta+\delta'}}\ \text{AND-POST-U}$$

Fig. 3. Selected structural rules of the higher-order union bound logic. These rules are reused for the higher-order expectation logic with a different interpretation of $\sqcap$, $\sqcup$ and $\Rrightarrow$.

Structural rules. Figure 3 shows selected structural rules of our logic. The rule of consequence, CONSEQ-U, allows weakening postconditions and error probabilities, and strengthening preconditions. The rule OR-PRE-U allows case analysis in the precondition. Finally, the rule AND-POST-U allows splitting a conjunction in the postcondition. Note that in this case, the error probability $\delta + \delta'$ is the sum of the error probabilities of the two conjuncts. This sum is a standard union bound on (error) probabilities, which explains the name of our logic.

Rules for monadic primitives. Additionally, we include rules for monadic primitives that we use in examples. For instance, the following rule is used for typing the term $\mathsf{Unif}(\sigma)$, which samples a value uniformly from the *finite* type σ. The sampling does not change the state, so the rule copies the precondition P to the postcondition. The sampled value additionally satisfies any predicate ϕ of cardinality N with probability $1 - N/|\sigma|$.

$$\frac{\Gamma \mid \Psi \vdash P \Rrightarrow (|\{x \in \sigma \mid x \in \phi\}| \,/\, |\sigma| \;=\; \delta)}{\Gamma \mid \Psi \vdash \{P\}\mathsf{Unif}(\sigma) : \mathsf{T}_{\emptyset,0}(\sigma)\{\{P \wedge \phi\}\}_{1-\delta}}\ \text{SAMPLE-UBL}$$

4.1.1 Adversary rule. In security applications, one often wants to prove properties of "adversarial" code, of which very little is known statically. Typically, one may know or assume that the adversarial code is closed, has a specific simple type and that it has a certain bounded complexity, but not

much else. Verification of such unknown code, unsurprisingly, relies on parametricity properties of the language. To this end, we need proof rules that internalize parametric reasoning into the logic. Below we show one such rule, ADV-U, which suffices for our examples. We first explain the rule informally and then give more formal details:

$$\frac{(\mathcal{A} : \forall\alpha.(\sigma \to \mathsf{T}_{\alpha,1}(\tau)) \to \mathsf{T}_{\Sigma\cup\alpha,k}(\tau')) \in \Delta \qquad \Delta \mid x : \sigma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma',1}(\tau)\{\{P\}\}_\delta \quad (x \notin \Psi, P) \qquad \Delta \mid \mathbf{s} : \mathbb{M} \vdash P \qquad P \in \mathsf{Safe}(\Sigma) \qquad \sigma, \tau, \tau' \text{ non-monadic types}}{\Delta \mid \bullet \mid \Psi \vdash \{P\}\mathcal{A}\ (\lambda x.t) : \mathsf{T}_{\Sigma\cup\Sigma',k}(\tau)\{\{P\}\}_{k\cdot\delta}}\ \text{ADV-U}$$

Informal explanation. Informally, ADV-U says the following. Suppose:

- $\mathcal{A}$ is an arbitrary (adversarial) *closed second-order program* whose side effects are limited to the locations in Σ, and that uses its argument at most k times (first premise),
- $\lambda x.t$ is an argument for $\mathcal{A}$ such that t *preserves* the assertion P on memories, except with probability δ (second premise), and
- P does not depend on the values in any locations in Σ (premise $P \in \mathsf{Safe}(\Sigma)$, which is defined formally later).

Then, $\mathcal{A}$ applied to $(\lambda x.t)$ preserves the assertion P except with probability $k \cdot \delta$.

We can easily see why the conclusion holds. First, since P depends only on values of locations outside Σ, to violate P, $\mathcal{A}$ must modify locations outside Σ. Next, the only way $\mathcal{A}$ can even hope to modify variables outside Σ is by invoking its argument $(\lambda x.t)$. This is because $\mathcal{A}$'s own effects are limited to Σ and it is closed, so it cannot get access to other effects due to additional substitutions. Hence, the only way for $\mathcal{A}$ to violate P is by invoking $\lambda x.t$. However, t violates P with probability at most δ and $\mathcal{A}$ cannot apply $\lambda x.t$ more than k times. Hence, by a straightforward union bound, $\mathcal{A}$'s chances of violating P are bounded by $k \cdot \delta$, which is exactly the conclusion.

The remarkable aspect of the rule is how little it assumes about the adversarial expression $\mathcal{A}$ – just that $\mathcal{A}$ is closed, that it uses its argument at most k times and that its side-effects are limited to Σ. The derived conclusion – that P is preserved except with probability $k \cdot \delta$ – holds for *any* closed, simply typed substitution for the variable $\mathcal{A}$. This is what makes this rule very powerful and useful. For example, in security applications, $\mathcal{A}$ can model an arbitrary, unknown "adversary" of bounded complexity k that is given a known "oracle" as argument. The rule then proves properties of any instance of the adversary applied to a given oracle $(\lambda x.t)$, without having to verify the adversary.

Formal notes. The type of $\mathcal{A}$ (first premise) ensures that its argument incurs an effect of at least 1 unit at each use, and that the total effect of $\mathcal{A}$ is k. Hence, $\mathcal{A}$ cannot use its argument more than k times. Further, the rule insists that $\mathcal{A}$ exist in Δ, not Γ. This ensures that $\mathcal{A}$ represents a closed term. Finally, the quantification over the effect set α ensures that $\mathcal{A}$ *itself* writes only to the locations in Σ.

The condition $P \in \mathsf{Safe}(\Sigma)$ is formally defined as $\forall m_1\ m_2 : \mathbb{M}.\ (\forall a \notin \Sigma.\ m_1[a] = m_2[a]) \Rightarrow P[m_1/\mathbf{s}] \Leftrightarrow P[m_2/\mathbf{s}]$, and means that P is independent of the values in locations in Σ. The condition that σ, τ and τ' be non-monadic is a technical simplification: the rule is proven sound using a logical relation, and terms of non-monadic types trivially inhabit the relation. The restriction can be lifted by imposing additional logical conditions on the argument and result value of t, as well as requiring that P must be also safe for the effects in τ'. Similarly, the restriction to closed adversaries can be lifted by requiring that every free variable is instantiated to a term that inhabits the logical relation.

We chose a particular second-order type for adversaries in this paper, which was the most convenient for our examples. However, the soundness argument can be used to easily derive adversary rules for different adversaries, e.g. adversaries that accept multiple oracles or third-order adversaries, that interact with oracles that receive functions as arguments.

In the examples it is often convenient to use a mild extension of the rule, where the invariant P_i and the error bound δ_i depend on some natural number i and we show that (1) each oracle call with precondition P_i satisfies the postcondition P_{i+1} with error probability δ_i and (2) every P_i implies P_{i+1}, from which we deduce that the adversarial computation satisfies the postcondition P_k with error probability $\delta_1 + \cdots + \delta_k$. To avoid cluttering the notation, here and in Section 5 we present the rules without this indexing.

4.1.2 Example: Probability of collisions. We now exercise our proof system to upper bound the probability of collisions for all adversaries making at most k queries to a PRF. Recall that our goal is to prove

$$\mathcal{A} : \forall\alpha.(\{0,1\}^l \to \mathsf{T}_{\alpha,1}(\{0,1\}^l)) \to \mathsf{T}_{\Sigma\cup\alpha,k}(\{0,1\}) \vdash \{Empty\}\mathcal{A}\ PRF : \mathsf{T}_{\Sigma,k}(\{0,1\})\{\{\Phi_k\}\}_{k(k+1)/2^{l+1}}$$

where $\Phi_i \triangleq |dom(L)| = |im(L)| \wedge |dom(L)| \leq i$.

Applying the proof rule for adversaries, it suffices to prove that the i-th call of the oracle preserves the assertion $NoColl$ with error probability at most $i/2^l$. Here we use the fact that at the i-th iteration the domain of L has size at most i. So we have to prove:

$$x : \{0,1\}^l \vdash \{|dom(L)| = |im(L)| \wedge |dom(L)| \leq i\} e : \mathsf{T}_{\alpha,1}(\{0,1\}^l)\{\{|dom(L)| = |im(L)| \wedge |dom(L)| \leq i+1\}\}_{i/2^l}$$

where e is the body of PRF. We work our way backwards starting with $P_{i+1} \triangleq \{\{|dom(L)| = |im(L)| \wedge |dom(L)| \leq i+1\}\}_{i/2^l}$ from the end of the program and compute the precondition of each statement. Note that we keep the grading because every precondition is the postcondition of the previous statement. The last instruction is a return, which we can skip since our assertion does not mention the return value.

Now we encounter the case split. The else branch is empty, so its precondition is still P_{i+1}. On the then branch we start by strengthening the postcondition to $\{\{x \notin dom(L) \wedge P_{i+1}\}\}_{i/2^l}$. Then we have the assignment $L_1[x_1] = z_1$, whose precondition is

$$\{\{x_1 \notin dom(L) \wedge |dom(L) \cup \{x_1\}| = |im(L) \cup \{z_1\}| \wedge |dom(L) \cup \{x_1\}| \leq i+1\}\}_{i/2^l}$$

Now we can strengthen this to

$$\{\{z_1 \notin im(L) \wedge x_1 \notin dom(L) \wedge |dom(L) \cup \{x_1\}| = |im(L) \cup \{z_1\}| \wedge |dom(L) \cup \{x_1\}| \leq i+1\}\}_{i/2^l}$$

which is equivalent to

$$\{\{z_1 \notin im(L) \wedge x_1 \notin dom(L) \wedge |dom(L)| + 1 = |im(L)| + 1 \wedge |dom(L)| + 1 \leq i+1\}\}_{i/2^l}$$

and by the SAMPLE-UBL rule, we know that the probability of sampling something outside $im(L)$ is at least $1 - i/(2^l)$, so the precondition of this is

$$\{\{x_1 \notin dom(L) \wedge |dom(L)| + 1 = |im(L)| + 1 \wedge |dom(L)| + 1 \leq i+1\}\}_0$$

This is the precondition of the then branch. By the MCASE-U rule, the precondition of the whole case construct is

$$\left\{\!\!\left\{ \begin{array}{c} (x \notin dom(L) \wedge (x \notin dom(L) \wedge |dom(L)| + 1 = |im(L)| + 1 \wedge |dom(L)| + 1 \leq i+1)) \vee \\ (x \in dom(L) \wedge (|dom(L)| = |im(L)| \wedge |dom(L)| \leq i+1)) \end{array} \right\}\!\!\right\}_0.$$

By strengthening, we finally get $\{\{(|dom(L)| = |im(L)| \wedge |dom(L)| \leq i))\}\}_0$, which is exactly P_i.

4.2 Higher-Order Expectation Logic

Our second logic (HO-EXP) is a *quantitative* (non-Boolean) higher-order expectation logic that proves upper bounds on expected values of functions of program results and final memories. The logic extends expectation calculi [Kaminski et al. 2016; Morgan et al. 1996] to the higher-order setting.

This logic is *syntactically* very similar to the higher-order union bound logic of Section 4.1 in the formulas, the judgments and most of the proof rules, but it is very different in the interpretation of *assertions* P, Q. Specifically, assertions in this logic are non-negative real-valued functions of their free variables $(\Delta, \Gamma, \mathbf{v}, \mathbf{s})$. The assertion connectives $\sqcap$ and $\sqcup$ are the pointwise supremum and infimum operators on such functions, as defined below. (Propositions ϕ, ψ still have Boolean interpretations, as in the union bound logic.)

To upper bound expected values, the monadic judgment $\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta$ is interpreted quantitatively, in terms of expectations.[4] Specifically, the inner judgment $\{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta$ means that for every state $m : \mathbb{M}$, if we run t from state m, then the *expected value of* Q over all possible final states is upper-bounded by $P[m/\mathbf{s}]+\delta$. That is, $\mathbb{E}_{(m',t')\sim t(m)}[Q[m'/\mathbf{s}][t'/\mathbf{v}]] \leq P[m/\mathbf{s}] + \delta$. The whole judgment $\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta$ means that this inequality holds for all substitutions for Γ that satisfy Ψ. Again, the formal semantics of this judgment is defined by a lifting. Conventionally, P and Q are respectively called the *pre-expectation* and the *post-expectation* of the term t. Informally, the judgment $\{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta$ means that the expected value of the post-expectation is upper-bounded by the pre-expectation plus an error δ.

Syntax. The logic reuses the syntax of the union bound logic (Section 4.1), but we extend assertions with some connectives that are specific to quantities. These new connectives are shown in **blue-bold** font below.

$$\text{Assertions} \quad P, Q \quad ::= \quad f(t_1, \ldots, t_n) \mid \top \mid \bot \mid \langle\phi\rangle \mid P \sqcup Q \mid P \sqcap Q \mid [\phi] \mid \mathbf{P + Q} \mid \mathbf{k \cdot P}$$

Assertions are quantities ranging over $[0, \infty]$. f denotes a function with codomain $[0, \infty]$. Assertion connectives have the following interpretations.

$$\begin{array}{lcl@{\qquad}lcl}
[\![\top]\!] & \triangleq & 0 & [\![\bot]\!] & \triangleq & \infty \\
[\![P \sqcup Q]\!] & \triangleq & \inf\{[\![P]\!], [\![Q]\!]\} & [\![P \sqcap Q]\!] & \triangleq & \sup\{[\![P]\!], [\![Q]\!]\} \\
[\![\langle\phi\rangle]\!] & \triangleq & \begin{cases} 0 & \phi \text{ holds} \\ \infty & \phi \text{ does not hold} \end{cases} & [\![[\phi]]\!] & \triangleq & \begin{cases} 1 & \phi \text{ holds} \\ 0 & \phi \text{ does not hold} \end{cases} \\
[\![P + Q]\!] & \triangleq & [\![P]\!] + [\![Q]\!] & [\![k \cdot P]\!] & \triangleq & k \cdot [\![P]\!]
\end{array}$$

Note that $\bot$ is interpreted as ∞, not 0. Similarly, $\sqcap$ corresponds to supremum, not infimum. This reversal of the usual order is due to the fact that we want to prove upper bounds. The connective $[\phi]$ is also called the Iverson bracket [Iverson 1962].

The judgment $\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}_\delta$ is interpreted as explained above. The judgment $\Gamma \mid \Psi \vdash P \Rrightarrow P'$ means that $P \geq P'$ for all instantiations of Γ that satisfy Ψ.

Proof rules. The expectation logic reuses the monadic and structural proof rules of the union bound logic as is (Figures 2 and 3). However, the rules' meanings are quantitative and their soundness is completely different. We illustrate the new meanings of some of these rules by explaining why they are still sound.

In rule UNIT-U, the premise ensures that ϕ holds (for the term t). So, $\langle\phi\rangle$ equals 0 semantically, and $\langle\phi\rangle \sqcap P$ is equivalent to P. Combined with the fact that $\mathsf{unit}(t)$ returns t and does not modify the memory, both with probability 1, the rule is trivially sound. The rule MLET-U corresponds

[4] As in Section 4.1, the contexts Ξ, Δ also exist but are elided from most of the presentation for brevity.

to the standard composition of random functions. In the rule MCASE-U, the precondition $\langle b = \mathbf{tt}\rangle \sqcap P_1) \sqcup (\langle b = \mathbf{ff}\rangle \sqcap P_2)$ in the conclusion is semantically equal to P_1 when $b = \mathbf{tt}$ and P_2 when $b = \mathbf{ff}$. Hence, the conclusion reduces to either the second or the third premise.[5]

We also have a new structural rule (LIN-EXP) that allows combining two different Hoare triples for the same program, relying on the linearity of expectations.

$$\frac{\Gamma \mid \Psi \vdash \{P_1\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q_1\}\}_{\delta_1} \qquad \Gamma \mid \Psi \vdash \{P_2\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q_2\}\}_{\delta_2}}{\Gamma \mid \Psi \vdash \{P_1 + P_2\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q_1 + Q_2\}\}_{\delta_1+\delta_2}}\ \text{LIN-EXP}$$

Rules for monadic primitives. Finally, we include rules for monadic primitives that we use in examples. For instance, the rule UNIF-EXP below applies to the term $\mathsf{Unif}(K)$, which samples from the uniform distribution over $\{0, 1, \ldots, K-1\}$. For $U \subseteq \{0, \ldots, K-1\}$, a value sampled from this distribution is in U with probability exactly $|U|/(K)$. Hence, the expected value of $[\mathbf{v} \in U]$ is exactly $|U|/(K)$, which is the pre-expectation.

$$\frac{U \subseteq \{0, \ldots, K-1\}}{\Gamma \mid \Psi \vdash \{(|U|/K) \cdot P\}\mathsf{Unif}(K) : \mathsf{T}_{\emptyset,0}(\mathbb{N}[K])\{\{[\mathbf{v} \in U] \cdot P\}\}_0}\ \text{UNIF-EXP}$$

Adversary rule. The expectation logic admits the adversary rule ADV-U of the union bound logic but with a quantitative definition of the meta-predicate Safe. Here, $P \in \mathsf{Safe}(\Sigma)$ is defined as $\forall m_1\, m_2 : \mathbb{M}.\ (\forall a \notin \Sigma.\ m_1[a] = m_2[a]) \Rightarrow P[m_1/\mathbf{s}] = P[m_2/\mathbf{s}]$. With this change to the definition of Safe, the rule is sound for expectations.

Example. We consider pollution attacks on Bloom Filters motivated in Section 2. We consider an arbitrary adversary $\mathcal{A}$ with access to the insert oracle of a Bloom filter. The goal of the adversary is to set as many bits in the Bloom filter to 1 as possible using k queries to the oracle. We assume that the Bloom filter is initially empty and, for simplicity, that it uses only one hash function, i.e., $\ell = 1$ (our proof easily generalizes to any ℓ). We model the hash function as a random oracle that is sampled lazily. The Bloom filter is implemented as a vector of m bits in locations $L[0], \ldots, L[m-1]$. The inserted elements are from the set $[n] = \{0, \ldots, n-1\}$, and $h[0] \ldots h[s-1]$ are auxiliary locations that hold integers. Additionally, we assume a location r that holds a counter. This is a ghost variable to help us in our verification effort, it is concretely used to make the invariant depend on the number of previous calls. Initially, each $L[i]$ is set to 0, each $h[i]$ is set to -1 and r is set to 0. The code of the insert oracle is shown below:

$$\begin{array}{lcl}
\mathsf{insert}(x : [n]) & \triangleq & \mathsf{let}\ b = \mathsf{read}\ h[x]\ \mathsf{in} \\
& & \mathsf{if}\ b \neq -1\ \mathsf{then} \\
& & \quad \mathsf{let}\ y = \mathsf{Unif}(m)\ \mathsf{in} \\
& & \quad h[x] := y; L[y] := 1; \mathsf{inc}\ r \\
& & \mathsf{else}\ \mathsf{inc}\ r
\end{array}$$

We want to show that the *expected* number of bits any adversary can set after making k calls to the adversary is upper bounded by $m(1 - ((m-1)/m)^k)$. For this, we prove that for any $\mathcal{A}$: $\forall\alpha.([n] \to \mathsf{T}_{\alpha,1}(\mathbb{U})) \to \mathsf{T}_{\alpha\cup\Sigma,k}(\mathbb{U})$, we have

$$\vdash \{F\}\mathcal{A}\ \mathsf{insert} : \mathsf{T}_{\Sigma\cup\{L,h,r\},k}(\tau)\{\{F\}\}_0,$$

where the expectation F is defined as

$$F = \left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right)((m-1)/m)^{k-\mathbf{s}[r]} + m\left(1 - ((m-1)/m)^{k-\mathbf{s}[r]}\right).$$

[5]The precondition $\langle b = \mathbf{tt}\rangle \sqcap P_1) \sqcup (\langle b = \mathbf{ff}\rangle \sqcap P_2)$ is semantically equivalent to $([b = \mathbf{tt}] \cdot P_1) + ([b = \mathbf{ff}] \cdot P_2)$. The latter is a more conventional way of writing the precondition [Morgan et al. 1996], but we prefer the former because it shows the correspondence to the union bound logic.

The idea behind this choice of F is that in the initial state where r and all $L[i]$s are 0, F equals the upper bound we want (shown above), and after the execution, when $\mathbf{s}[r] = k$, F equals $\sum_{i\in[m]} \mathbf{s}[L[i]]$, whose expectation is what we want to upper bound.

By the adversary rule ADV-U rule, we need to show that insert preserves F, i.e., $\vdash \{F\}\mathsf{insert}\ x : \mathsf{T}_{\{L,h,r\},1}(\tau)\{\{F\}\}_0$. We first use the rule MLET-U. Since read $h[x]$ trivially preserves F, we need to show that the if-then-else preserves F. We use CONSEQ-U to replace the pre-condition's F with the equivalent $(\langle(b \neq -1) = \mathbf{tt}\rangle \sqcap F) \sqcup (\langle(b \neq -1) = \mathbf{ff}\rangle \sqcap F)$. Using the rule MCASE-U, we then need to prove that the "then" and "else" branches preserve F.

The else branch is fairly straightforward. We need to show that

$$\vdash \{F\}\mathsf{let}\ c = \mathsf{read}\ r\ \mathsf{in}\ r := c + 1 : \mathsf{T}_{\{L,h,r\},1}(\tau)\{\{F\}\}_0.$$

Using the rules MLET-U, READ-U and WRITE-U, we get

$$\vdash \{F[\mathbf{s}[r \mapsto (\mathbf{s}[r] + 1)]/\mathbf{s}]\}\mathsf{let}\ c = \mathsf{read}\ r\ \mathsf{in}\ r := c + 1 : \mathsf{T}_{\{L,h,r\},1}(\tau)\{\{F\}\}_0.$$

Hence, by CONSEQ-U, it suffices to show that $F \geq F[\mathbf{s}[r \mapsto (\mathbf{s}[r] + 1)]/\mathbf{s}] = F[(\mathbf{s}[r] + 1)/\mathbf{s}[r]]$, which follows immediately because F is a decreasing function of $\mathbf{s}[r]$ (this uses the fact that each $\mathbf{s}[L[i]]$ is either 0 or 1).

On the then branch, we take into account the following property of the uniform distribution:

$$\vdash \left\{((m-1)/m)\left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right) + 1\right\}\ \mathsf{Unif}(m)\ \left\{\left\{1 + \textstyle\sum_{i\in([m]\setminus\{\mathbf{v}\})} \mathbf{s}[L[i]]\right\}\right\}. \tag{1}$$

To prove this property, we first note that the post-expectation is equal to

$$[\mathbf{v} \in \{i \mid \mathbf{s}[L[i]] = 1\}] \cdot \left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right) + [\mathbf{v} \in \{i \mid \mathbf{s}[L[i]] = 0\}] \cdot \left(1 + \textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right).$$

By LIN-EXP and UNIF-EXP, the pre-expectation of this with respect to $\mathsf{Unif}(m)$ is

$$(1/m)\left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right)\left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right) + (1/m)\left(m - \textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right)\left(1 + \textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right),$$

which equals the pre-expectation of (1).

We return to the proof of the then branch and reason backwards from the end of the branch. After going backwards over $h[x] := y; L[y] := 1; \mathsf{let}\ c = \mathsf{read}\ r\ \mathsf{in}\ r := c + 1$, our pre-expectation becomes $F[(\mathbf{s}[r \mapsto \mathbf{s}[r] + 1][L[y] \mapsto 1])/\mathbf{s}]$, which expands to

$$\left(1 + \textstyle\sum_{i\in([m]\setminus\{y\})} \mathbf{s}[L[i]]\right) \cdot ((m-1)/m)^{k-\mathbf{s}[r]-1} + m(1 - ((m-1)/m)^{k-\mathbf{s}[r]-1}).$$

Using (1) and linearity to compute the pre-expectation of the sampling command, which is

$$\left((m-1)/m \cdot \left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right) + 1\right) \cdot ((m-1)/m)^{k-\mathbf{s}[r]-1} + m(1 - ((m-1)/m)^{k-\mathbf{s}[r]-1}),$$

and by some rearranging of the terms, this is equal to

$$\left(\textstyle\sum_{i\in[m]} \mathbf{s}[L[i]]\right) \cdot ((m-1)/m)^{k-\mathbf{s}[r]} + m(1 - ((m-1)/m)^{k-\mathbf{s}[r]}),$$

which coincides with F. This concludes the proof.

5 HIGHER-ORDER PROBABILISTIC RELATIONAL LOGIC

In this section, we present a logic (HO-RPL) to reason about relations between two computations. The syntax of the logic is shown below, where the propositions, assertions and assumptions have the same meaning as in Section 4.1. Here we note that, although we keep the abstract syntax of assertions, in this section we consider only their Boolean interpretation, where the connectives are replaced by their usual Boolean counterparts and $\langle\phi\rangle$ is equivalent to ϕ. Researching a quantitative interpretation, where assertions are interpreted as distances, is an interesting direction for future work.

$$
\begin{array}{lll}
\text{Propositions} & \phi, \psi & ::= R(t_1, \ldots, t_n) \mid \top \mid \bot \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \Rightarrow \psi \mid \neg\phi \mid \forall x : \sigma.\phi \mid \exists x : \sigma.\phi \\
\text{Assertions} & P, Q & ::= f(\tilde{t_1}, \ldots, \tilde{t_n}) \mid \top \mid \bot \mid \langle\phi\rangle \mid P \sqcup Q \mid P \sqcap Q \\
\text{Assumptions} & \Psi & ::= \bullet \mid \Psi, \psi \\
\text{Judgments} & \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash \phi & \\
& \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash P \Rrightarrow Q & \\
& \Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \ \{\phi\} & \\
& \Gamma \mid \Psi \vdash \{P\} t : \mathsf{T}_{\Sigma_1, k_1}(\sigma_1) \sim t_2 : \mathsf{T}_{\Sigma_2, k_2}(\sigma_2) \{\{Q\}\}_\delta &
\end{array}
$$

We have already explained the first two judgments in previous sections. The third form of judgment $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \ \{\phi\}$ constitutes the non-monadic fragment of the logic and comes from RHOL [Aguirre et al. 2017], a logic to prove relational properties of pure higher-order programs directed by the syntax of the programs. In these judgments, $\Gamma, \mathbf{r}_1, \mathbf{r}_2 \vdash \phi$ is a HOL formula depending on two distinguished variables $\mathbf{r}_1, \mathbf{r}_2$ that represent the term on the left of the judgment and the term on the right, respectively. The interpretation is given by the equivalence $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \ \{\phi\} \Leftrightarrow \Gamma \mid \Psi \vdash \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$, which follows from the relative completeness theorem of RHOL. We present the rules for RHOL in the appendix.

The fourth kind of judgments is new to our presentation, and is introduced to reason about monadic computations. These have the syntax $\Gamma \mid \Psi \vdash \{P\} t_1 : \mathsf{T}_{\Sigma,k}(\sigma_1) \sim t_2 : \mathsf{T}_{\Sigma,k}(\sigma_2) \{\{Q\}\}_\delta$ where P is a Boolean-valued assertion (called the pre-condition) well-formed in the context $\Gamma, \mathbf{s}_1 : M, \mathbf{s}_2 : M$, and Q is another Boolean-valued assertion (called the post-condition) well-formed in the context $\Gamma, \mathbf{s}_1 : M, \mathbf{s}_2 : M, \mathbf{v}_1 : \sigma_1, \mathbf{v}_2 : \sigma_2$. Here, the variables $\mathbf{s}_1, \mathbf{s}_2$ refer to (resp. left or right-side) memories, and $\mathbf{v}_1, \mathbf{v}_2$ refer to (resp. left or right-side) result values. Here δ is a quantitative bound taken in an ordered monoid; in the simplest case, the monoid consists of a single element 0. For the particular interpretation presented in this section, we take the monoid of non-negative reals with addition. Following the convention of RHOL, we assume that the free variables of t_1 and t_2 are disjoint.

The semantics of judgments is based on the notion of statistical distance. For a general Q, the meaning of the judgment depends on the lifting defined in Example 6.6. Here we give an intuition for the case where Q is of the form $\mathbf{s}_1 = \mathbf{s}_2 \sqcap \mathbf{v}_1 = \mathbf{v}_2$, sufficient for our examples. If we can derive

$$\Gamma \mid \Psi \vdash \{P\} t_1 : \mathsf{T}_{\Sigma,k}(\tau) \sim t_2 : \mathsf{T}_{\Sigma,k}(\tau) \{\{\mathbf{s}_1 = \mathbf{s}_2 \sqcap \mathbf{v}_1 = \mathbf{v}_2\}\}_\delta$$

then for every instantiation of Γ satisfying Ψ, and every pair of initial memories $m_1, m_2 \in M$, such that $m_1, m_2 \in P$, the statistical distance between the output distributions $t_1(m_1)$ and $t_2(m_2)$ is at most δ, i.e., for every event S the absolute difference between the probabilities of S in $t_1(m_1)$ and $t_2(m_2)$ is at most δ. In particular, when t_2 is a renaming of t_1, $\delta = 0$, and P and Q define partial equivalences on memories, the judgment enforces a form of generalized non-interference.

Monadic and structural rules. Figure 4 presents selected monadic and structural rules. Following a pattern that is standard for relational logics, we have 2-sided rules, such as [UNIT − R], [MLET − R], [READ − R], [WRITE − R] and [MCASE − R], where the two expressions have the same top-level structure, and 1-sided rules, such as [L − UNIT − R] and [L − MLET − R], which exclusively consider the top-level construct of one expression. These generalize their unary counterparts. In particular, the rule [MCASE − R] has an extra assumption ensuring that the two computations go to the same branch, so we only need to prove a relation between the then branches and a relation between the else branches. A 1-sided rule without this assumption also exists, allowing to consider the 4 possible pairs of branches, but we do not show it here.

Rules for sampling. Our logic also features rules for reasoning about sampling. In contrast to the other rules, these are only valid to the particular interpretation based on statistical distance that

$$\frac{\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \mid \phi \qquad \Gamma, \mathbf{s}_1 : M, \mathbf{s}_2 : M \vdash P}{\Gamma \mid \Psi \vdash \{P\}\mathsf{unit}(t_1) : \mathsf{T}_{\emptyset,0}(\tau_1) \sim \mathsf{unit}(t_2) : \mathsf{T}_{\emptyset,0}(\tau_2)\{\{\langle\phi\rangle \sqcap P)\}\}_0}\ \text{UNIT-R}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash \{P\}t_1 : \mathsf{T}_{\Sigma_1,k_1}(\tau_1) \sim t_2 : \mathsf{T}_{\Sigma_2,k_2}(\tau_2)\{\{Q\}\}_\delta \\ \Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi \vdash \{Q[x_1/\mathbf{v}_1][x_2/\mathbf{v}_2]\}u_1 : \mathsf{T}_{\Sigma'_1,k'_1}(\sigma_1) \sim u_2 : \mathsf{T}_{\Sigma'_2,k'_2}(\sigma_2)\{\{R\}\}_{\delta'} \qquad x_1, x_2 \notin R\end{array}}{\Gamma \mid \Psi \vdash \{P\}\mathsf{let}\ x_1 = t_1\ \mathsf{in}\ u_1 : \mathsf{T}_{\Sigma_1\cup\Sigma'_1,k_1+k'_1}(\sigma_1) \sim \mathsf{let}\ x_2 = t_2\ \mathsf{in}\ u_2 : \mathsf{T}_{\Sigma_2\cup\Sigma'_2,k_2+k'_2}(\sigma_2)\{\{R\}\}_{\delta+\delta'}}\ \text{MLET-R}$$

$$\frac{\Xi;\Gamma \mid \Psi \vdash a_1 : \mathsf{Loc}\{\psi\} \qquad \Xi;\Gamma \mid \Psi \vdash a_2 : \mathsf{Loc}\{\psi\}}{\Gamma \mid \Psi \vdash \{P[\mathbf{s}_1[a_1]/\mathbf{v}_1][\mathbf{s}_2[a_2]/\mathbf{v}_2]\}\mathsf{read}\ a_1 : \mathsf{T}_{\{a_1\},0}(\mathbb{V}) \sim \mathsf{read}\ a_2 : \mathsf{T}_{\{a_2\},0}(\mathbb{V})\{\{P\}\}_0}\ \text{READ-R}$$

$$\frac{\Xi;\Gamma \vdash a_1 : \mathsf{Loc} \qquad \Xi;\Gamma \vdash t_1 : \mathbb{V} \qquad \Xi;\Gamma \vdash a_2 : \mathsf{Loc} \qquad \Xi;\Gamma \vdash t_2 : \mathbb{V}}{\Gamma \mid \Psi \vdash \{P[\mathbf{s}_1[a_1 \mapsto t_1]/\mathbf{s}_1][\mathbf{s}_1[a_2 \mapsto t_2]/\mathbf{s}_1]\}a_1 := t_1 : \mathsf{T}_{\{a_1\},0}(\mathbb{U}) \sim a_2 := t_2 : \mathsf{T}_{\{a_2\},0}(\mathbb{U})\{\{P\}\}_0}\ \text{WRITE-R}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash b_1 : \mathbb{B} \sim b_1 : \mathbb{B}\ \{b_1 = b_2\} \qquad \Gamma \mid \Psi \wedge b_1 = \mathbf{tt} \vdash \{P_1\}t_1 : \mathsf{T}_{\Sigma_1,k_1}(\tau_1) \sim t_2 : \mathsf{T}_{\Sigma_2,k_2}(\tau_2)\{\{Q\}\}_\delta \\ \Gamma \mid \Psi \wedge b_1 = \mathbf{ff} \vdash \{P_2\}u_1 : \mathsf{T}_{\Sigma_1,k_1}(\tau_1) \sim u_2 : \mathsf{T}_{\Sigma_2,k_2}(\tau_2)\{\{Q\}\}_\delta \\ P \triangleq (\langle b_1 = \mathbf{tt}\rangle \sqcap P_1)) \sqcup (\langle b_1 = \mathbf{ff}\rangle \sqcap P_2)\end{array}}{\Gamma \mid \Psi \vdash \{P\}\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ u_1 : \mathsf{T}_{\Sigma_1,k_1}(\tau_1) \sim \mathsf{if}\ b\ \mathsf{then}\ t_2\ \mathsf{else}\ u_2 : \mathsf{T}_{\Sigma_2,k_2}(\tau_2)\{\{Q\}\}_\delta}\ \text{MCASE-R}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \tau_1\{\phi\} \qquad \Gamma, \mathbf{s}_1 : M, \mathbf{s}_2 : M \vdash P}{\Gamma \mid \Psi \vdash \{P\}\mathsf{unit}(t_1) : \mathsf{T}_{\emptyset,0}(\tau_1) \sim \mathsf{skip} : \mathsf{T}_{\emptyset,0}(\mathbb{U})\{\{\langle\phi\rangle \sqcap P\}\}_0}\ \text{L-UNIT-R}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash \{P\}t_1 : \mathsf{T}_{\Sigma_1,k_1}(\tau_1) \sim \mathsf{skip} : \mathsf{T}_{\emptyset,0}(\mathbb{U})\{\{Q\}\}_\delta \\ \Gamma, x_1 : \tau_1, x_2 : \mathbb{U} \mid \Psi \vdash \{Q[x_1/\mathbf{v}_1][x_2/\mathbf{v}_2]\}u_1 : \mathsf{T}_{\Sigma'_1,k'_1}(\sigma_1) \sim u_2 : \mathsf{T}_{\Sigma_2,k_2}(\sigma_2)\{\{R\}\}_\delta \qquad x_1, x_2 \notin R\end{array}}{\Gamma \mid \Psi \vdash \{P\}\mathsf{let}\ x_1 = t_1\ \mathsf{in}\ u_1 : \mathsf{T}_{\Sigma_1\cup\Sigma'_1,k_1+k'_1}(\sigma_1) \sim u_2 : \mathsf{T}_{\Sigma_2,k_2}(\sigma_2)\{\{R\}\}_{\delta+\delta'}}\ \text{L-MLET-R}$$

Fig. 4. Relational logic: monadic rules

we present here. We show one rule below:

$$\frac{B_1 \subseteq B_2\ \text{finite}}{\Gamma \mid \Psi \vdash \{P\}\mathsf{Unif}(B_1) : \mathsf{T}_{\emptyset,0}(B_1) \sim \mathsf{Unif}(B_2) : \mathsf{T}_{\emptyset,0}(B_2)\{\{\langle \mathbf{v}_1 = \mathbf{v}_2\rangle \sqcap P\}_{|B_1|/|B_2|}}\ \text{SAMPLE-R}$$

The rule is used to compare uniform samplings from two finite sets. There exists an alternative rule where $\delta = 0$, at the cost of weakening the postcondition; this rule is shown in the appendix.

Adversary rule. The adversary rule for the relational setting is similar in spirit to the adversary rule for the unary setting. However, some mild adjustments are needed. First, we need to modify the notion of safety for a region Σ. In the unary case we only required that writing to Σ preserves the invariant. In the relational case, we also need to require that ϕ is also "safe for reading in Σ", meaning that an adversary reading from two different memories related by ϕ at the same location in Σ sees the same value. This prevents the two executions from diverging due to a read operation:

Definition 1. *Let ϕ be a predicate and $\Sigma \subseteq \mathsf{Loc}$. We say that $\phi \in \mathsf{RSafe}(\Sigma)$ iff*

$$\forall m_1, m_2, \forall l \in \Sigma. \forall v \in \mathbb{V}. \phi(m_1, m_2) \Rightarrow \phi(m_1[l \mapsto v], m_2[l \mapsto v]) \wedge m_1[l] = m_2[l]$$

The adversary rule also allows us to show that the outputs must be *extensionally equal*, which corresponds to the predicate Eq_τ defined below. The reason to use this as opposed to equality in the model is that the logical relation we use in the soundness proof corresponds to extensional

equality for non-monadic types:

$$\mathsf{Eq}_b(x_1, x_2) \triangleq x_1 = x_2$$
$$\mathsf{Eq}_{\tau_1 \to \tau_2}(x_1, x_2) \triangleq \forall y_1, y_2 \in \tau_1 . \mathsf{Eq}_{\tau_1}(y_1, y_2) \Rightarrow \mathsf{Eq}_{\tau_1}(x_1\ y_1, x_2\ y_2)$$
$$\mathsf{Eq}_{\tau_1 \times \tau_2}(x_1, x_2) \triangleq \mathsf{Eq}_{\tau_1}(\pi_1(x_1), \pi_1(x_2)) \wedge \mathsf{Eq}_{\tau_2}(\pi_2(x_1), \pi_2(x_2))$$

The adversary rule can now be stated below:

$$\frac{\begin{array}{c}(\mathcal{A} : \forall \alpha.(\sigma \to \mathsf{T}_{\alpha,1}(\tau)) \to \mathsf{T}_{\Sigma \cup \alpha, k}(\tau')) \in \Delta \\ \mathbf{s}_1 : M, \mathbf{s}_2 : M \vdash \phi \qquad \phi \in \mathsf{RSafe}(\Sigma) \qquad x_1 \notin FV(t_2), x_2 \notin FV(t_1) \qquad \sigma, \tau, \tau' \text{ non-monadic types} \\ x_1 : \sigma, x_2 : \sigma \mid \mathsf{Eq}_\sigma(x_1, x_2) \vdash \{\phi\} t_1 : \mathsf{T}_{\Sigma',1}(\tau) \sim t_2 : \mathsf{T}_{\Sigma',1}(\tau) \{\{\langle \mathsf{Eq}_\tau(\mathbf{v}_1, \mathbf{v}_2)\rangle \sqcap \phi\}\}_\delta\end{array}}{\Delta \mid \cdot \mid \cdot \vdash \{\phi\} \mathcal{A}(\lambda x_1.t_1) : \mathsf{T}_{\Sigma \cup \Sigma', k}(\tau') \sim \mathcal{A}(\lambda x_2.t_2) : \mathsf{T}_{\Sigma \cup \Sigma', k}(\tau') \{\{\langle \mathsf{Eq}_{\tau'}(\mathbf{v}_1, \mathbf{v}_2)\rangle \sqcap \phi\}\}_{k\delta}} \text{ ADV-R}$$

Informally, the premises of the rule state:

- ϕ is a safe for the memory region Σ;
- if their inputs are extensionally equal and their initial memories are related by ϕ, then the oracles produce equal outputs and final memories related by ϕ, with error δ;
- $\mathcal{A}$ is an arbitrary adversary that only writes to and reads from Σ and that can call its argument up to k times

From them, we conclude that executing the adversary with these oracles and initial memories related by ϕ should yield equal values and output memories related by ϕ, with error $k\delta$.

Example: PRF/PRP Switching Lemma. We use our logic to show that the probability that an adversary can distinguish between a PRF and a PRF on bitstrings of fixed length l is upper bounded by $\frac{k(k+1)}{2^{l+1}}$, where k is the maximal number of queries allowed to the adversary. As before, we consider a mild extension of the logic where the error bound can depend on the oracle counter. For readability, we will generally omit from our judgments the effect, adversary and variable contexts, and drop the cost grading from the monadic types, and omit all reasoning about the size of the domain of L. Our goal is to show:

$$\vdash \{\mathbf{s}_1 = \mathbf{s}_2\}\ \mathcal{A}\ PRF : \mathsf{T}_{\Sigma \cup \{L\}, k}(\{0,1\}) \sim \mathcal{A}\ PRP : \mathsf{T}_{\Sigma \cup \{L\}, k}(\{0,1\})\ \{\{\mathbf{v}_1 = \mathbf{v}_2\}\}_{k(k+1)/2^{l+1}}$$

By applying the rule $[\mathsf{ADV} - \mathsf{R}]$ on the strengthened judgment, we are left to prove:

$$x_1 = x_2 \vdash \{\mathbf{s}_1 = \mathbf{s}_2\} e_{PRF} : \mathsf{T}_{\{L\}}(\{0,1\}) \sim e_{PRP} : \mathsf{T}_{\{L\}}(\{0,1\}) \{\{\mathbf{s}_1 = \mathbf{s}_2 \sqcap \mathbf{v}_1 = \mathbf{v}_2\}\}_{i/2^l}$$

where e_{PRF} and e_{PRP} denote the bodies of the PRF and PRP oracles. We then apply the $[\mathsf{MCASE} - \mathsf{R}]$ rule. In the empty *else* branch, we need to prove:

$$x_1 = x_2 \vdash \{\mathbf{s}_1 = \mathbf{s}_2\} (\mathsf{read}\ L)[x_1] : \mathsf{T}_L(\{0,1\}^l) \sim (\mathsf{read}\ L)[x_2] : \mathsf{T}_L(\{0,1\}^l) \{\{\mathbf{s}_1 = \mathbf{s}_2 \sqcap \mathbf{v}_1 = \mathbf{v}_2\}\}_{i/2^l}$$

which is a simple application of the $[\mathsf{READ} - \mathsf{R}]$ rule. In the *then* branch, we first apply the $[\mathsf{WRITE} - \mathsf{R}]$ rule, and then we are left to prove:

$$\vdash \{\mathbf{s}_1 = \mathbf{s}_2\} \mathsf{Unif}(X_1) : \mathsf{T}(\{0,1\}^l) \sim \mathsf{Unif}(X_2) : \mathsf{T}_{L,1}(\{0,1\}^l) \{\{\mathbf{s}_1[x_1 \mapsto \mathbf{v}_1] = \mathbf{s}_2[x_2 \mapsto \mathbf{v}_2] \sqcap \mathbf{v}_1 = \mathbf{v}_2\}\}_{i/2^l}$$

where $X_1 \triangleq \{0,1\}^l$ and $X_2 \triangleq \{0,1\}^l \setminus \mathsf{im}(\mathbf{v}[L])$. By the rule of consequence, this follows from

$$\vdash \{\mathbf{s}_1 = \mathbf{s}_2\} \mathsf{Unif}(X_1) : \mathsf{T}(\{0,1\}^l) \sim \mathsf{Unif}(X_2) : \mathsf{T}_{L,1}(\{0,1\}^l) \{\{\mathbf{s}_1 = \mathbf{s}_2 \sqcap \mathbf{v}_1 = \mathbf{v}_2\}\}_{i/2^l}$$

which we prove using the $[\mathsf{SAMPLE} - \mathsf{R}]$ rule.

6 SEMANTICS

Now we present the formal semantics for our system. We begin with some background, and follow with the semantics of the language and the logics, and their soundness theorems.

6.1 The Category of Quasi-Borel Spaces and Probability Monad

We will assume knowledge of some concepts of category theory, such as bi-Cartesian closed categories (bi-CCC) and strong monads, see e.g. [MacLane 1971] for details. In any biCCC $\mathbb{C}$ in this paper, we fix a terminal object $(1, !_X \in \mathbb{C}(X, 1))$, and for each pair $X, Y \in \mathbb{C}$ of objects, we fix a binary product $(X \times Y, \pi_1, \pi_2, \langle -, - \rangle)$, a binary coproduct $(X + Y, \iota_1, \iota_2, [-, -])$ and an exponential object $(X \Rightarrow Y, \mathrm{ev}, \lambda(-))$. We also equip $\mathbb{C}$ with the symmetric monoidal structure $(1, (\times), l, r, a, s)$ induced by the fixed terminal object and binary products.

We will use the category **QBS** of quasi-Borel spaces [Heunen et al. 2017; Ścibior et al. 2017] for modeling higher-order probabilistic programs introduced in Section 3. The category **QBS** is a well-pointed bi-CCC; in fact it has small products and coproducts. For modeling probabilistic choice, we employ the strong monad $(\mathcal{P}, \eta^{\mathcal{P}}, \mu^{\mathcal{P}}, \theta^{\mathcal{P}})$ for probability measures over QBSs [Heunen et al. 2017]. For a set A and a QBS X, by $(A \cdot X, \{\iota_a^{A,X} : X \to A \cdot X\}_{a \in A}, [-]_{a \in A})$ we mean the coproduct of A-many copies of X.

We write $|-| : \mathbf{QBS} \to \mathbf{Set}$ for the forgetful functor extracting the carrier set of QBS. It preserves finite (actually small) products. To ease calculation, we assume $|1| = 1$ and $|X \times Y| = |X| \times |Y|$ (rather than isomorphic). We also assume that the exponential of **QBS** is defined so that $|X \Rightarrow Y| = \mathbf{QBS}(X, Y)$. Finally, we write $[0, \infty]_{\mathbf{QBS}}$ for the QBS of non-negative extended reals.

6.2 Probabilistic State Monad

Starting from this base, which was already presented in [Heunen et al. 2017] we use the state monad transformer to construct a strong monad given by a functor.

First, we introduce the QBS for memory states. Fix a QBS V corresponding to type $\mathbb{V}$. The QBS for memory states is a product $(M, \{\pi_a : M \to V\}_{a \in \mathsf{Loc}})$ of Loc-many copies of V. We next introduce a memory update function. Given a QBS morphism $f : X \to V$ computing a value from an environment, we define the memory update $u_a(f) : X \times M \to M$ (at location $a \in \mathsf{Loc}$) to be the unique morphism satisfying $\pi_a \circ u_a(f) = f \circ \pi_1$ and $\pi_{a'} \circ u_a(f) = \pi_{a'} \circ \pi_2$ for any $a' \neq a$. We then define the probabilistic state monad by $\mathcal{PS} \triangleq M \Rightarrow \mathcal{P}(- \times M)$. The unit $\eta_X^{\mathcal{PS}} : X \to \mathcal{PS}X$, multiplication $\mu_X^{\mathcal{PS}} : \mathcal{PS}(\mathcal{PS}X) \to \mathcal{PS}X$ and strength $\theta_{X,Y}^{\mathcal{PS}} : X \times \mathcal{PS}(Y) \to \mathcal{PS}(X \times Y)$ of this monad are defined as:

$$\eta^{\mathcal{PS}} \triangleq \lambda(\eta^{\mathcal{P}}_{X \times M}) \qquad \mu^{\mathcal{PS}} \triangleq M \Rightarrow (\mu^{\mathcal{P}} \circ \mathcal{P}(ev)) \qquad \theta^{\mathcal{PS}}_{X,Y} \triangleq \lambda(\mathcal{P}\alpha^{-1} \circ \theta^{\mathcal{P}} \circ (X \times \mathrm{ev}) \circ \alpha).$$

6.3 Semantics of the Language

As demonstrated by Moggi, the computational metalanguage (the simply typed lambda calculus with monadic types) is naturally interpreted in any CCC with a strong monad. The semantics of the language in Section 3 follows the same pattern. To accommodate probabilities we take the category **QBS** with the probabilistic state monad $\mathcal{PS}$.

The semantics of types is defined as objects in **QBS**, assuming we have an object $[\![b_i]\!]$ for every base type $b_i \in B$. Note that the indices of the monad and the quantification over regions are erased at the semantic level (below, for a natural number K, $\bar{K}$ denotes the set $\{0, \cdots, K\}$):

$$[\![\mathbb{B}]\!] \triangleq \{\bot, \top\} \cdot 1 \quad [\![\mathbb{N}[K]]\!] \triangleq \bar{K} \cdot 1 \quad [\![\mathbb{U}]\!] \triangleq 1 \quad [\![\mathbb{V}]\!] \triangleq V \quad [\![\mathbb{M}]\!] \triangleq M$$
$$[\![\sigma \to \tau]\!] \triangleq [\![\sigma]\!] \Rightarrow [\![\tau]\!] \quad [\![\sigma \times \tau]\!] \triangleq [\![\sigma]\!] \times [\![\tau]\!] \quad [\![\mathsf{T}_{\Sigma,k}(\sigma)]\!] \triangleq \mathcal{PS}([\![\sigma]\!]) \quad [\![\forall \alpha.\tau]\!] \triangleq [\![\tau]\!]$$

This categorical semantics erases the effect annotations Σ, k of the monadic type $\mathsf{T}_{\Sigma,k}(\tau)$ and the universal quantification $\forall \alpha.\tau$ over regions, which only play a role in proving soundness of the adversary rules. Adversary variables are placeholders for closed terms, and do not play any special role in the semantics. We therefore give a semantics of the language without contexts Ξ and Δ.

$$\begin{aligned}
[\![\Gamma \vdash \mathsf{unit}(t) : \mathsf{T}_{\Sigma,k}(\sigma)]\!] &\triangleq \eta^{\mathcal{PS}} \circ [\![\Gamma \vdash t : \sigma]\!] \\
[\![\Gamma \vdash \mathsf{let}\ x = t\ \mathsf{in}\ u : \mathsf{T}_{\Sigma,k}(\tau)]\!] &\triangleq [\![\Gamma, x : \sigma \vdash u : \mathsf{T}_{\Sigma,k}(\tau)]\!]^{\#_{\mathcal{PS}}} \circ m_{\Gamma,x:\sigma} \circ \theta^{\mathcal{PS}} \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \vdash t : \mathsf{T}_{\Sigma,k}(\sigma)]\!]\rangle \\
[\![\Gamma \vdash \mathsf{read}\ a : \mathsf{T}_{\Sigma,k}(\mathbb{V})]\!] &\triangleq \lambda(\eta^{\mathcal{P}} \circ \langle \pi_a, id\rangle \circ \pi_2) \\
[\![\Gamma \vdash a := t : \mathsf{T}_{\Sigma,k}(\mathbb{U})]\!] &\triangleq \lambda(\eta^{\mathcal{P}} \circ \langle !, \mathsf{id}\rangle \circ u_a([\![\Gamma \vdash t : \mathbb{V}]\!])) \\
[\![\Gamma \vdash \mathsf{sample}(\nu(t_1, \ldots, t_k)) : \mathsf{T}_{\Sigma,k}(\sigma_\nu)]\!] &\triangleq \lambda(\vartheta^{\mathcal{P}} \circ \langle [\![\nu]\!] \circ \langle [\![\Gamma \vdash t_1 : \tau_{\nu,1}]\!], \ldots, [\![\Gamma \vdash t_k : \tau_{\nu,|\nu|}]\!]\rangle \circ \pi_1, \pi_2\rangle) \\
[\![\Gamma \vdash \mathsf{mfold}\ n\ t_1\ t_2 : \mathsf{T}_{\Sigma\cup\Sigma',k+K\cdot k'}(\sigma)]\!] &\triangleq \\
& fold_{K\cdot 1,[\![\mathsf{T}_{\Sigma,k}(\sigma)]\!]} \circ \langle [\![\Gamma \vdash n : \mathbb{N}[K]]\!], [\![\Gamma \vdash t_1 : \mathsf{T}_{\Sigma,k}(\sigma)]\!], Kl_{[\![\Gamma]\!],[\![\mathsf{T}_{\Sigma',k'}(\sigma)]\!]} \circ [\![\Gamma \vdash t_2 : \sigma \to \mathsf{T}_{\Sigma',k'}(\sigma)]\!]\rangle
\end{aligned}$$

Fig. 5. Semantics of the language

We interpret the subtyping relation $\Xi \vdash \tau \preceq \tau'$ as a coercion morphism $c_{\tau,\tau'} : [\![\tau]\!] \to [\![\tau']\!]$. Most of its definition is routine, except for the case of natural number type: for $K \leq K'$, $c_{\mathbb{N}[K],\mathbb{N}[K']}$ is defined to be the evident morphism $\bar{K} \cdot 1 \to \bar{K'} \cdot 1$.

Semantics of a context Γ is given by the Cartesian product of the interpretation of types in Γ. For convenience, we fix a product $([\![\Gamma]\!], \{\pi^\Gamma_x : [\![\Gamma]\!] \to [\![\Gamma(x)]\!]\}_{x\in\mathrm{dom}(\Gamma)})$ for each context Γ. For a context $\Gamma, x_1 : \tau_1, \cdots, x_n : \tau_n$, by $m_{\Gamma,x_1:\tau_1,\cdots,x_n:\tau_n} : [\![\Gamma]\!] \times ([\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]) \to [\![\Gamma, x_1 : \tau_1, \cdots, x_n : \tau_n]\!]$ we mean the evident isomorphism in $\mathbf{QBS}$. Also, for a well-typed term $\Gamma \vdash t : \tau$ and $x \notin \mathrm{dom}\,\Gamma$, we define the substitution morphism $\mathrm{sub}^{\Gamma\vdash t:\tau}_x : [\![\Gamma]\!] \to [\![\Gamma, x : \tau]\!]$ to be the composite $m_{\Gamma,x:\tau} \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \vdash t : \tau]\!]\rangle$.

Well-typed terms $\Gamma \vdash t : \sigma$ are interpreted as a morphism in $\mathbf{QBS}([\![\Gamma]\!], [\![\sigma]\!])$. The interpretation of monadic types can be found in Figure 5; the interpretation of the non-monadic fragment is standard and deferred to the appendix. In the Figure, $(-)^{\#_{\mathcal{PS}}}$ denotes the Kleisli lifting of $\mathcal{PS}$; $Kl_{X,Y} : (X \Rightarrow \mathcal{PS}Y) \to (\mathcal{PS}X \Rightarrow \mathcal{PS}Y)$ denotes the internal Kleisli lifting; $\vartheta^{\mathcal{P}}_{X,Y} : \mathcal{P}(X) \times Y \to \mathcal{P}(X \times Y)$ is the co-strength, a transformation analogous to the strength but with swapped arguments; and $fold_{K\cdot 1,X} : K \cdot 1 \times X \times (X \Rightarrow X) \to X$ denotes the iterator over the natural numbers up to K. We assume that every distribution ν with arity $\tau_{\nu,1} \times \cdots \times \tau_{\nu,|\nu|} \to \sigma_\nu$ has an interpretation $[\![\nu]\!]$ of the proper type $[\![\tau_{\nu,1}]\!] \times \cdots \times [\![\tau_{\nu,|\nu|}]\!] \to \mathcal{P}[\![\sigma_\nu]\!]$. This semantics is sound in the following sense:

Theorem 6.1. *Let $\Xi \mid \Delta \mid \Gamma \vdash t : \sigma$ be a well-typed term and $\emptyset \vdash t_i : \Delta(\mathcal{A}_i)$ be closed terms given for each $\mathcal{A}_i \in \mathrm{dom}(\Delta)$. Then $[\![\Gamma \vdash t[t_i/\mathcal{A}_i]_{\alpha\in\mathrm{dom}\,\Delta} : \sigma]\!] \in \mathbf{QBS}([\![\Gamma]\!], [\![\sigma]\!])$.*

6.4 Heyting-Valued Predicates over QBSs

Formulas and assertions are interpreted in the same way as *predicates over QBSs*. Recall that a complete Heyting algebra is a complete lattice $\Omega = (\Omega, \sqsubseteq)$ (whose meet and join are denoted by $\sqcap$ and $\sqcup$ respectively) together with a *pseudo-complement* operator $\Rrightarrow$.

Definition 2. *Let $\Omega = (\Omega, \sqsubseteq)$ be a complete Heyting algebra. An Ω-valued predicate on a QBS X is a function of type $|X| \to \Omega$. Define $\mathbf{UP}^\Omega_X \triangleq \mathbf{Set}(|X|, \Omega)$ to mean the set of Ω-valued predicates on X.*

By the pointwise order, $\mathbf{UP}^\Omega_X$ is again a complete Heyting algebra. We define $\mathbf{2} \triangleq \{\bot \sqsubseteq \top\}$ to mean the Sierpinski space complete Heyting algebra. For $x \in |X|$ and $P \in \mathbf{UP}^{\mathbf{2}}_X$, we say that x satisfies P, denoted by $x \models P$, if and only if $P(x) = \top$.

Every $\mathbf{2}$-valued predicate can be converted into a Ω-valued predicate. Define $I : \mathbf{UP}^{\mathbf{2}}_X \to \mathbf{UP}^\Omega_X$ by $I(P)(x) = \top_\Omega$ if $P(x) = \top_2$ and $I(P)(x) = \bot_\Omega$ if $P(x) = \bot_2$. This is a complete Heyting algebra homomorphism, that is, a function preserving all joins, all meets and pseudo-complements.

We introduce a generalized inverse image operation for Ω-valued predicates. For a QBS-morphism $f : Y \to X$, define $f^* : \mathbf{UP}^\Omega_X \to \mathbf{UP}^\Omega_Y$ by $f^*P = P \circ |f|$. This is also a complete Heyting algebra homomorphism. We also introduce a notation: for a QBS morphism $f : X \to Y$ and $P \in \mathbf{UP}^\Omega_X$ and

$Q \in \mathbf{UP}^{\Omega}_{Y}$, we write $f : P \dot{\to} Q$ to mean the inequality $P \sqsubseteq f^* Q$ in $\mathbf{UP}^{\Omega}_{X}$. When $\Omega = \mathbf{2}$, $f : P \dot{\to} Q$ is equivalent to stating that for any γ, $\gamma \models P$ implies $|f|(\gamma) \models Q$.

We also define the partial application of an Ω-valued predicate with an environment. Let $\Gamma, x_1 : \tau_1, \cdots, x_n : \tau_n$ be a context. For a predicate $P \in \mathbf{UP}^{\Omega}_{[\![\Gamma, x_1:\tau_1, \cdots, x_n:\tau_n]\!]}$ and $\gamma \in |[\![\Gamma]\!]|$, by $P_\gamma \in \mathbf{UP}^{\Omega}_{[\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]}$ we mean the predicate $P_\gamma(p) = P \circ |m_{\Gamma, x_1:\tau_1, \cdots, x_n:\tau_n}|(\gamma, p)$. For $P \in \mathbf{UP}^{\Omega}_{X}$ and $Q \in \mathbf{UP}^{\Omega}_{Y}$, we define $P \dot{\times} Q = \pi_1^* P \sqcap \pi_2^* Q$.

Note that we have chosen predicates to be morphisms in **Set**, rather than morphisms in **QBS**. This allows us to avoid reasoning about measurability when defining predicates and writing specifications, while still having a model that works as intended when the predicates are measurable.

6.5 Strong Graded Liftings of the Probability Measure Monad

We introduce a concept called *strong graded lifting* of strong monads. The following definition is a specialization of the one in [Katsumata 2014] to Ω-valued predicates.

Definition 3 (Heyting-valued strong graded lifting of strong monad). *Let $(E, \leq, 0, +)$ be a partially ordered monoid. An* Ω-valued strong E-graded lifting *of $\mathcal{P}$ is a family of functions $\dot{\mathcal{P}}_X : E \to (\mathbf{UP}^{\Omega}_{X} \Rightarrow \mathbf{UP}^{\Omega}_{\mathcal{P}X})$, implicitly indexed by $X \in \mathbf{QBS}$, satisfying:*

$$e \leq e' \implies \dot{\mathcal{P}}(e)(P) \sqsubseteq \dot{\mathcal{P}}(e')(P) \qquad \eta^{\mathcal{P}}_X : P \dot{\to} \dot{\mathcal{P}}(0)(P)$$
$$\mu^{\mathcal{P}}_X : \dot{\mathcal{P}}(e)(\dot{\mathcal{P}}(e')(P)) \dot{\to} \dot{\mathcal{P}}(e + e')(P) \qquad \theta^{\mathcal{P}}_X : P \dot{\times} \dot{\mathcal{P}}(e)(Q) \dot{\to} \dot{\mathcal{P}}(e)(P \dot{\times} Q).$$

The following is an informal explanation of liftings, ignoring for the moment the gradings. The second and third conditions specify how the unit and multiplication of the lifting interact with the unit and multiplication of the base monad: for a predicate P over X and $x \in P$, then $\eta(x) \in \dot{\mathcal{P}}(0)(P)$, and if $x \in \dot{\mathcal{P}}(e)(\dot{\mathcal{P}}(e')(P))$, then $\mu(x) \in \dot{\mathcal{P}}(e + e')(Q)$. The fourth condition specifies a similar interaction with the strength. These properties are used in proving the soundness of the rules of our logics. At the level of liftings, gradings can be seen as some additional specification, or as adding quantitative information to the specification. For instance, in HO-UBL use the grading on a lifting to specify the probability with which a computation may fail to satisfy the specification. This is the intuition behind the first condition in the definition: it allows weakenings of the grading of a lifted predicate, i.e., if $e \leq e'$ then $\dot{\mathcal{P}}(e)(Q) \Rightarrow \dot{\mathcal{P}}(e')(Q)$. We now present some examples of liftings that we will use in our soundness proofs:

Example 6.2 (Lifting for union bounds). Inspired from the lifting for the union bound introduced in [Sato et al. 2019, Section 9.1], we give a $\mathbf{2}$-valued strong $([0, \infty], \leq, +, 0)$-graded lifting $\dot{\mathcal{P}}^{\mathrm{ub}}$ of $\mathcal{P}$:

$$\dot{\mathcal{P}}^{\mathrm{ub}}_X(\delta)(P)(\nu) = \top \iff \forall f \in \mathbf{QBS}(X, \{0,1\} \cdot 1), P \sqsubseteq |f| \,.\, \Pr_{x \sim \nu}[f(x) = 1] \geq 1 - \delta.$$

This can be constructed by the graded $\top\top$-lifting [Katsumata 2014]. Morally, we want $\dot{\mathcal{P}}^{\mathrm{ub}}_X(\delta)(P)(\nu)$ to hold if the probability of sampling a value from μ that satisfies P is at least $1 - \delta$. However, we cannot compute this probability directly because P may not be measurable. Instead, we need to quantify over all the measurable f above P. We can then show that if P is indeed measurable (i.e. $P = |P_0|$ for some QBS-morphism P_0), $\Pr_{x \sim \nu}[P(x) = 0] \leq \delta$ holds for every $\nu \in \dot{\mathcal{P}}^{\mathrm{ub}}(\delta)(P)$. That is, for measurable predicates, the lifting behaves as intended.

Example 6.3 (Lifting for expectations). We introduce a $([0, \infty], \geq)$-valued strong $([0, \infty], \leq, +, 0)$-graded lifting $\dot{\mathcal{P}}^{\mathrm{Exp}}$ of $\mathcal{P}$. Note that the predicates take values in the Heyting algebra $[0, \infty]$ *with reversed order* (i.e. $x \sqsubseteq y$ iff $y \leq x$), which will be used to reason about upper bounds. We define:

$$\dot{\mathcal{P}}^{\exp}_X(\delta)(P)(\nu) = \sup\{S_{\delta+\delta'}(f^\sharp \nu) \mid f : X \to \mathcal{P}[0, \infty], \sup\{S_{\delta'}(f(i)) \mid P(i) < S_{\delta'}(f(i))\} < S_{\delta+\delta'}(f^\sharp \nu)\}$$
$$\text{where } S_\delta(\mu) = \max(0, \mathbb{E}_{r \sim \mu}[r] - \delta).$$

This can be constructed by the graded $\top\top$-lifting [Katsumata 2014] since $\delta \leq \delta' \implies S_\delta \sqsubseteq S_{\delta'}$. Intuitively, the lifting $\dot{\mathcal{P}}^{\mathrm{Exp}}$ gives an upper bound of expected value of P under a distribution ν with margin of error δ. In the general case, where P is not measurable, we get instead an upper bound on the expected value of any $f : X \dot{\to} [0,\infty]$ measurable in **QBS** such that $|f| \leq P$ (i.e. $P \sqsubseteq |f|$). That, is we obtain $\dot{\mathcal{P}}^{\mathrm{Exp}}_X(\delta)(P)(\nu) \sqsubseteq \mathbb{E}_{x\in\nu}[f(x)] - \delta$.

6.6 Combining Liftings and State Transformer Monads

The material from the previous section allows us to model predicates over the monad $\mathcal{P}$, but we need to extend it to model predicates over the probabilistic state monad $\mathcal{PS}$ that models computations in our language. The same approach of finding a lifting of $\mathcal{PS}$ does not work directly because it would not allow us to include the specification about states. Such a lifting would map a Ω-valued predicate over X to a 2-valued predicate over $\mathcal{PS}X = M \Rightarrow \mathcal{P}(X \times M)$, but this does not match the shape of triples in our logics. We actually need to lift a pair of Ω-valued predicates over M (the precondition) and over $X \times M$ (the postcondition) into a 2-valued predicate over $\mathcal{PS}X$.

Therefore, we need to find a different construction. Assume there exists a Ω-valued strong $(E, \leq, 0, +)$-graded lifting $\dot{\mathcal{P}}$ of $\mathcal{P}$. For each QBS X, we define a function $\dot{\mathcal{P}}\dot{\mathrm{S}}_X(-)(-,-) : E \to (\mathbf{UP}^\Omega_M \times \mathbf{UP}^\Omega_{X\times M} \Rightarrow \mathbf{UP}^2_{\mathcal{P}SX})$ by $f \models \dot{\mathcal{P}}\dot{\mathrm{S}}_X(e)(P,Q) \iff f : P \dot{\to} \dot{\mathcal{P}}(\delta)(Q)$. Recall that $|\mathcal{PS}X| = \mathbf{QBS}(M, \mathcal{P}(X \times M))$. We call $\dot{\mathcal{P}}\dot{\mathrm{S}}$ a stateful lifting. This can be seen as a transformer that takes a Ω-valued strong E-graded lifting $\dot{\mathcal{P}}$ of $\mathcal{P}$ and returns a stateful lifting of the probabilistic state transformer monad $\mathcal{PS}$. In plain words, $\dot{\mathcal{P}}\dot{\mathrm{S}}$ maps an Ω-valued precondition $P \in \mathbf{UP}^\Omega_M$ and an Ω-valued postcondition $Q \in \mathbf{UP}^\Omega_{X\times M}$ to the computations in $\mathcal{PS}(X)$ that send initial memories in P to distributions over $X \times M$ satisfying the lifted predicate $\dot{\mathcal{P}}(e)(Q)$. In a way, this can be seen as the set of computations $f : \mathsf{T}_{\Sigma,e}(\sigma)$ satisfying the generalized Hoare triple $\{P\} f : \mathsf{T}_{\Sigma,k}(\sigma)\{\{Q\}\}$.

This operator is not an $E-$graded lifting, because it does not have the appropriate type. However, properties of Ω-valued strong E-graded liftings can be extended to $\dot{\mathcal{P}}\dot{\mathrm{S}}$ as stated below:

Lemma 1. *Let $\dot{\mathcal{P}}$ be an Ω-valued strong $(E, \leq, 1, \cdot)$-graded lifting of $\mathcal{P}$. Let $f \in \mathbf{QBS}(X \times M, \mathcal{P}(Y \times M))$, and $P \in \mathbf{UP}^2_X$, $Q \in \mathbf{UP}^\Omega_M$, $R \in \mathbf{UP}^\Omega_{X\times M}$ and $S \in \mathbf{UP}^\Omega_{Y\times M}$ be predicates. The following holds:*

$$\eta^{\mathcal{PS}}_X : P \dot{\to} \dot{\mathcal{P}}\dot{\mathrm{S}}_X(0)(Q, IP \dot{\times} Q)$$
$$f : R \dot{\to} \dot{\mathcal{P}}(e)(S) \implies (\lambda(f))^{\#_{\mathcal{PS}}} : \dot{\mathcal{P}}\dot{\mathrm{S}}_X(e')(Q,R) \dot{\to} \dot{\mathcal{P}}\dot{\mathrm{S}}_Y(e' + e)(Q,S)$$
$$\theta^{\mathcal{PS}}_{X,Y} : P \dot{\times} \dot{\mathcal{P}}\dot{\mathrm{S}}_Y(e)(Q,R) \dot{\to} \dot{\mathcal{P}}\dot{\mathrm{S}}_{X\times Y}(e)(Q, \pi_1^* IP \sqcap R)$$

These consequences can be explained informally by using the language of Hoare logic:

- The first consequence states that if $x \in \phi$, then $\{Q\}\ \eta(x)\ \{\{\phi \sqcap Q\}\}_0$ is a valid generalized Hoare triple for any Q.
- The second consequence gives us a way to sequence computations as in Hoare logic. It states that if f satisfies $\{Q(x,-)\}\ f(x)\ \{\{R\}\}_e$ for every argument x, and t satisfies $\{P\}\ t\ \{\{Q\}\}_{e'}$, then $\{P\}$ let $x = t$ in $f\ \{\{R\}\}_{e+e'}$ is a valid generalized Hoare triple. Here $Q(x,-)$ is the set of memories m such that $(x,m) \in Q$.
- The third consequence states that if $x \in \phi$ and we have t such that $\{P\}\ t\ \{\{Q\}\}_e$ then $\{P\}\ \theta(x,t)\ \{\{\phi \sqcap Q\}\}_e$ is a valid generalized Hoare triple.

6.7 Soundness of the Unary Logics

We interpret formulas, assertions and entailment relations in the logic by complete Heyting algebras over QBSs. We first develop the semantics of HOL judgements of the form $\bullet \mid \bullet \mid \Gamma \mid \Psi \vdash \phi$, which is simply denoted by $\Gamma \mid \Psi \vdash \phi$. We then interpret an open judgement $J = \Xi \mid \Delta \mid \Gamma \mid \Psi \vdash \phi$ as the conjunction of all closed instantiations $\bullet \mid \bullet \mid \Gamma' \vdash \Psi' \vdash \Phi'$ of J. Here, each $\alpha \in \Xi$ is

instanciated with some subset of Loc, and $\mathcal{A} \in \Delta$ is instanciated with a closed term of type $\Delta(\mathcal{A})$. The semantics of open judgements of UHOL and HO-UBL are similarly defined. This interpretation is well-behaved with respect to substitution. In particular, the substitution $\phi[t/x]$ of x by a term t of appropriate type can be interpreted by the inverse image $[\![\Gamma \vdash \phi[t/x]]\!] = (\mathrm{sub}_x^{\Gamma \vdash t:\tau})^*[\![\Gamma, x\colon \tau \vdash \phi]\!]$. The soundness results of Aguirre et al. [2017] for the base logics HOL and UHOL can be recovered in this setting, but we defer it to the appendix.

We interpret HO-UBL using the lifting $\dot{\mathcal{P}}^{\mathrm{ub}}\dot{\mathrm{S}}$ of the probabilistic state monad constructed from the lifting for the union bound logic $\dot{\mathcal{P}}^{\mathrm{ub}}$ as in Example 6.2. The soundness result is stated as:

Proposition 6.4. *Let $\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta$ be a derivable HO-UBL judgment without the adversary rule. Then, for any $\gamma \in |[\![\Gamma]\!]|$, $\gamma \models [\![\Gamma \vdash \bigwedge \Psi]\!]$ implies*

$$[\![\Gamma \vdash t : \mathsf{T}_{\Sigma,k}(\tau)]\!](\gamma) \models \dot{\mathcal{P}}^{\mathrm{ub}}\dot{\mathrm{S}}(\delta)([\![\Gamma, \mathbf{s} : \mathbb{M} \vdash P]\!]_\gamma, [\![\Gamma, \mathbf{v} : \tau, \mathbf{s} : \mathbb{M} \vdash Q]\!]_\gamma).$$

Analogously, we interpret HO-EXP using the lifting $\dot{\mathcal{P}}^{\mathrm{exp}}\dot{\mathrm{S}}$ of the probabilistic state monad constructed from the lifting for expectations bound logic $\dot{\mathcal{P}}^{\mathrm{exp}}$ as in Example 6.3. Most of the proof of the previous result can be reused, and only the rules for sampling and linearity need separate proofs. The soundness result is stated as:

Proposition 6.5. *Let $\Gamma \mid \Psi \vdash \{P\}t : \mathsf{T}_{\Sigma,k}(\tau)\{\{Q\}\}_\delta$ be a derivable HO-UBL judgment without the adversary rule. Then, for any $\gamma \in |[\![\Gamma]\!]|$, $\gamma \models [\![\Gamma \vdash \bigwedge \Psi]\!]$ implies*

$$[\![\Gamma \vdash t : \mathsf{T}_{\Sigma,k}(\tau)]\!](\gamma) \models \dot{\mathcal{P}}^{\mathrm{exp}}\dot{\mathrm{S}}(\delta)([\![\Gamma, \mathbf{s} : \mathbb{M} \vdash P]\!]_\gamma, [\![\Gamma, \mathbf{v} : \tau, \mathbf{s} : \mathbb{M} \vdash Q]\!]_\gamma).$$

6.8 Semantics for the Relational Logics

Let $\Omega = (\Omega, \sqsubseteq)$ be a complete Heyting algebra. To interpret relational logics, we first define the concept of Ω-valued binary relation between two QBSs X, Y. They are simply Ω-valued predicates over product QBS $X \times Y$. We thus define $\mathbf{BR}^{\Omega}_{X,Y} \triangleq \mathbf{UP}^{\Omega}_{X\times Y}$. For 2-valued binary relation $P \in \mathbf{BR}^{2}_{X,Y}$ and $(x, y) \in |X \times Y| = |X| \times |Y|$, we say that (x, y) satisfies P (denoted by $(x, y) \models P$) if $P(x, y) = \top$.

We routinely extend the development in the previous section to Ω-valued binary relations. For QBS-morphisms $f : X \to Y$ and $f' : X' \to Y'$, we define the pullback operation $(f, f')^* : \mathbf{BR}^{\Omega}_{X',Y'} \to \mathbf{BR}^{\Omega}_{X,Y}$ to be $(f \times f')^*$. We write $(f, f') : P \dot{\to} Q$ to mean $P \sqsubseteq (f, f')^*Q$.

We introduce the concept of *Heyting-algebra valued strong graded relational lifting*.

Definition 4. *Let $(E, \leq, 0, +)$ be a partially ordered monoid. An* Ω-valued strong E-graded relational lifting *of $\mathcal{P}$ is a family of functions $\dot{\mathcal{P}}_{X,Y}(-)(-) : E \to (\mathbf{BR}^{\Omega}_{X,Y} \Rightarrow \mathbf{BR}^{\Omega}_{\mathcal{P}X,\mathcal{P}Y})$, implicitly indexed by $X, Y \in \mathbf{QBS}$, satisfying:*

$$e \leq e' \implies \dot{\mathcal{P}}(e)(P) \sqsubseteq \dot{\mathcal{P}}(e')(P) \qquad (\eta^{\mathcal{P}}_X, \eta^{\mathcal{P}}_Y) : P \dot{\to} \dot{\mathcal{P}}(0)(P)$$

$$(\mu^{\mathcal{P}}_X, \mu^{\mathcal{P}}_Y) : \dot{\mathcal{P}}(e)(\dot{\mathcal{P}}(e')(P)) \dot{\to} \dot{\mathcal{P}}(e + e')(P) \qquad (\theta^{\mathcal{P}}_X, \theta^{\mathcal{P}}_Y) : P \mathbin{\dot{\times}} \dot{\mathcal{P}}(e)(Q) \dot{\to} \dot{\mathcal{P}}(e)(P \mathbin{\dot{\times}} Q).$$

Example 6.6 (Relational lifting for differential privacy). Inspired from [Sato 2016], we construct a 2-valued strong graded relational lifting for *differential privacy* by a graded analogue of the codensity lifting [Katsumata et al. 2018]. The grading monoid is the product partially ordered monoid $([0, \infty], \leq, 0, +)^2$.

$$\dot{\mathcal{P}}^{\mathrm{dp}}_{X,Y}(\epsilon, \delta)(P)(\nu_1, \nu_2) = \top \iff \forall (f, g)\colon P \dot{\to} S(\epsilon', \delta')\ .\ (f^\sharp \nu_1, g^\sharp \nu_2) \models S(\epsilon + \epsilon', \delta + \delta')$$
$$\text{where } (\nu_1, \nu_2) \models S(\epsilon, \delta) \iff \Pr_{b\sim\nu_1}[b = 0] \leq e^{\epsilon} \Pr_{b\sim\nu_2}[b = 0] + \delta.$$

Any morphism $\chi_S\colon X \to \{0, 1\}$ in **QBS** standing for the "measurable" subset S of X, we have $\dot{\mathcal{P}}^{\mathrm{dp}}_{X,X}(\epsilon, \delta)(\mathrm{Eq}_X)(\mu_1, \mu_2) = \top \implies \Pr_{x\sim\mu_1}[x \in S] \leq e^{\epsilon} \Pr_{x\sim\mu_2}[x \in S] + \delta$ since $S(0, 0) \models (\chi_S, \chi_S)$. A strong $([0, \infty], \leq, 0, +)$-graded lifting describing ϵ-differential privacy can be given by $\dot{\mathcal{P}}^{\mathrm{dp}}_{X,Y}(\epsilon, 0)$.

We next introduce the stateful relational lifting, based on a Ω-valued $(E, \leq, +, 0)$-graded relational lifting $\dot{\mathcal{P}}$ of $\mathcal{P}$. It is a function $\dot{\mathcal{P}}\dot{\mathsf{S}}_{X,Y}(-)(-,-) : E \to (\mathbf{BR}^{\Omega}_{X,Y} \times \mathbf{BR}^{\Omega}_{X\times M, Y\times M} \Rightarrow \mathbf{BR}^{2}_{\mathcal{P}\mathcal{S}X, \mathcal{P}\mathcal{S}Y})$ defined for each $X, Y \in \mathbf{QBS}$ by:

$$(f, f') \models \dot{\mathcal{P}}\dot{\mathsf{S}}_{X,Y}(e)(P, Q) \iff (f, f') : P \dot{\to} \dot{\mathcal{P}}(e)(Q).$$

Lemma 1 can then be generalized accordingly. We omit the details.

6.9 Soundness of the Relational Logics

The semantics of the relational logics are a generalization of the semantics of the unary logics. We defer soundness of RHOL to the appendix. We interpret the monadic rules in the category of relations over QBS, with the 2-valued strong $[0,\infty]^2$-graded relational lifting $\dot{\mathcal{P}}^{\mathrm{dp}}$, which induces $\dot{\mathcal{P}}^{\mathrm{dp}}\dot{\mathsf{S}}$ as in the previous section. The soundness result is stated below. Its proof is by induction on the derivation and is largely independent of the choice of a specific lifting:

Proposition 6.7. *Let* $\Gamma \mid \Psi \vdash \{P\}t_1 : \mathsf{T}_{\Sigma,k}(\tau_1) \sim t_1 : \mathsf{T}_{\Sigma,k}(\tau_2)\{\{Q\}\}_\delta$ *be a derivable HO-PRL judgment without the* [ADV − R] *rule. Then for any* $\gamma \in |[\![\Gamma]\!]|$, $\gamma \models [\![\Gamma \vdash \bigwedge \Psi]\!]$ *implies*

$$([\![\Gamma \vdash t_1 : \mathsf{T}_{\Sigma,k}(\tau_1)]\!](\gamma), [\![\Gamma \vdash t_2 : \mathsf{T}_{\Sigma,k}(\tau_2)]\!](\gamma)) \models \dot{\mathcal{P}}\dot{\mathsf{S}}(0,\delta)([\![P]\!]_\gamma, [\![Q]\!]_\gamma),$$

where $[\![P]\!]_\gamma \triangleq [\![\Gamma, \mathbf{s}_1 : \mathbb{M}, \mathbf{s}_2 : \mathbb{M} \vdash P]\!]_\gamma$ *and* $Q_\gamma \triangleq [\![\Gamma, \mathbf{s}_1 : \mathbb{M}, \mathbf{v}_1 : \sigma, \mathbf{s}_2 : \mathbb{M}, \mathbf{v}_2 : \sigma \vdash Q]\!]_\gamma$.

6.10 Soundness of the Adversary Rules

To prove soundness of the adversary rules we will use the technique of logical relations. Logical predicates and relations [Plotkin 1973] are a technique used in programming language theory to prove properties such as strong normalization or contextual equivalence. The idea of logical relations (or predicates) is that they allow us to prove that all inhabitants of a certain type τ satisfy a particular property $\mathcal{L}(\tau)$ that is defined inductively on the structure of types, rather than terms.

For instance, in the unary case, we define a logical predicate $\mathcal{L}_{\phi,\delta}(\cdot)$ indexed by an assertion ϕ over memories and a real $\delta \geq 0$. For every type σ, $\mathcal{L}_{\phi,\delta}(\sigma)$ corresponds to a set of closed terms. We defer the details of this definition to the appendix, here it suffices to know that $\mathcal{L}_{\phi,\delta}(\mathsf{T}_{\Sigma,k}(\tau))$ is the set of computations that preserve the invariant ϕ with error probability $k \cdot \delta$ and that return a result in $\mathcal{L}_{\phi,\delta}(\tau)$ (i.e., they satisfy the triple $\{\phi\}\ \_\ \{\{\phi \wedge \mathbf{v} \in \mathcal{L}_{\phi,\delta}(\tau)\}\}_{k\cdot\delta}$), and as usual, if $t \in \mathcal{L}_{\phi,\delta}(\sigma \to \tau)$ and $u \in \mathcal{L}_{\phi,\delta}(\sigma)$ then $t\ u \in \mathcal{L}_{\phi,\delta}(\tau)$. Then we prove a Basic Lemma: any closed term t of type τ inhabits the predicate $\mathcal{L}_{\phi,\delta}(\tau)$ if $\phi \in \mathsf{Safe}(\mathbf{Eff}(\tau))$. This has a rather natural interpretation: if ϕ does not depend on any location in $\mathbf{Eff}(\tau)$, then it must be preserved after running t.

The adversary rule [ADV − U] can then be proven sound from the Basic Lemma. By inspecting its premises, we know that $\mathcal{A}$ inhabits the logical relation $\mathcal{L}_{P,\delta}(\forall\alpha.(\sigma \to \mathsf{T}_{\alpha,1}(\tau)) \to \mathsf{T}_{\Sigma\cup\alpha,k}(\tau'))$, because $P \in \mathsf{Safe}(\Sigma)$. We also have that $\lambda x.t$ inhabits the logical relation $\mathcal{L}_{P,\delta}(\sigma \to \mathsf{T}_{\Sigma',1}(\tau))$, because we have a derivation of this fact (note that the Basic Lemma cannot be applied, because P may not be safe for Σ'). Then, we can conclude that running $\mathcal{A}$ with $\lambda x.t$ as argument inhabits the logical relation $\mathcal{L}_{P,\delta}(\mathsf{T}_{\Sigma\cup\alpha,k}(\tau'))$, and therefore must preserve P. The techniques then generalize to the relational case, where we define a logical relation for every type, and then we prove a Basic Lemma for it. Soundness of [ADV − R] is a consequence of this Lemma.

Proposition 6.8. *The* [ADV − U] *and* [ADV − R] *rules are sound.*

7 RELATED WORK

Reasoning about adversaries. Garg et al. [2010]; Jia et al. [2015] develop first- and higher-order program logics to reason about safety properties of first-order concurrent and stateful programs interacting with adversaries. Both provide rules to reason about adversaries, morally similar to ours.

Our context of adversary variables representing closed programs traces lineage to a similar idea based on comonads in Jia et al. [2015]. Devriese et al. [2016] develop semantic principles to reason about adversaries, cast in terms of parametricity properties of side-effects, an idea they call "effect parametricity". They use these principles to verify code that uses object capabilities. No syntactic proof rules are developed. These works cover only the boolean, deterministic, unary setting.

Closer to our work, Barthe et al. [2009] define a probabilistic relational Hoare logic (pRHL) for reasoning about the security of cryptographic constructions. Their logic applies to a probabilistic imperative language with adversarial calls and features a proof rule for adversaries. Our rule for the relational, non-quantitative setting closely matches their rule. Barbosa et al. [2021] formalize a resource-aware module system used in EasyCrypt to reason about adversaries. There are commonalities between their approach and ours: they view an adversary as a functor, whereas we view an adversary as an expression of second-order type. However, the technicalities are very different, since they build their system on top of an imperative language. A further difference is that they account for the computational cost of adversaries, which we left aside in this work. Other similar approaches for reasoning about adversaries include Computational Indistinguishability Logic [Barthe et al. 2010], and state-separating proofs [Brzuska et al. 2018]. However, these approaches are developed in an abstract mathematical setting, not in the context of a programming language.

Barthe et al. [2016a] define an adversary rule for reasoning about differential privacy in a quantitative variant of pRHL; their rule uses bounds on the number of oracle queries to derive privacy bounds of adversarial computations from privacy bounds of oracles. Barthe et al. [2016b] define a Union Bound logic to reason about accuracy of adversarial computations for a similar language. However, their proof rule is restricted to adversaries without oracles. We are not aware of any prior work on adversarial computations in the quantitative setting.

Program logics for probabilistic computations. We relate our program logics to existing approaches for reasoning about probabilistic computations. For brevity, we only discuss approaches not discussed before. Kozen [1985] introduces expectation-based reasoning for a core probabilistic programming language. Morgan et al. [1996] define a weakest pre-expectation calculus. Aguirre et al. [2021] develop a variant of the calculus for relational properties. Kaminski et al. [2016] show how similar ideas can be used for reasoning about expected cost. All these works share the setting of a probabilistic imperative language. Aguirre and Katsumata [2020] show that expectation-based reasoning remains sound in a higher-order setting, but their semantics is based on set theory, not Quasi-Borel spaces, so they cannot model continuous distributions. They also do not provide proof systems.

There exist adaptations of (approximate) probabilistic relational Hoare logic in the higher-order setting, starting from [Barthe et al. 2014a]. However, these adaptations have a set-theoretical or topos of trees semantics and only support distributions over discrete base types. Sato et al. [2019] introduce an expressive logic for a language similar to ours but without state and adversary. Their model is also based on QBS. Tassarotti and Harper [2019] develop a logic to prove relational properties of higher-order programs that combine probabilities and non-determinism. They do not support all the kinds of reasoning we do, and the relations they can prove are between a program and an specification, rather than between two programs. Maillard et al. [2020] define a framework, embedded in a relational dependent type theory, for defining and reasoning about program logics for general monadic effects. While their work is based on Dijkstra monads, ours is more closely related to Hoare monads [Nanevski et al. 2013, 2008]. Our work extends Hoare monads to support Heyting-valued predicates, probabilistic programs, grading and adversarial reasoning.

Program equivalence. There is a very large body of methods for proving program equivalence, and in particular contextual equivalence, in higher-order languages with state, probabilities, and effects; see e.g. [Benton et al. 2014; Bizjak and Birkedal 2015; Crubillé and Lago 2015; Jung et al. 2015;

Matache and Staton 2019; Pitts and Stark 1998]. Many of these methods have been applied to reason about security and privacy, using the natural view of adversaries as contexts. These methods are not comparable with ours: our relational logic can prove a richer set of specifications (for instance, the postcondition needs not be an equivalence relation). However, they cannot establish some basic equivalences, e.g. swapping of two sampling instructions, due to the specific way the logic constructs couplings. We also conjecture that our logics are easier to extend to richer settings, such as multi-stage and multi-adversary security notions (see e.g. [Ristenpart et al. 2011]). Finally, these methods cannot be used to reason about unary properties.

8 CONCLUDING REMARKS

We conclude the paper with a discussion of additional examples that can be handled by the three logics we have presented (and by small extensions to the logics), and a discussion of how we can extend our framework with unbounded recursion.

Other examples. HO-UBL can verify the *accuracy* of differentially private mechanisms such as the Sparse Vector algorithm [Dwork and Roth 2014], since accuracy can be formulated as the probability that the noisy answer is close to the actual answer. We have already worked out this example but, for reasons of space, we defer it to the appendix.

The bounded leakage model is a model of leakage-resilient cryptography in which the adversary is given access to a leakage oracle which takes as input a function with a small codomain and returns the output of this function applied to the secret state. A (partially formalized) proof of security of a pseudo-random generator in the bounded leakage model is given in [Barthe et al. 2014b]. HO-PRL can be used to verify this proof, using either a first- or a higher-order representation of leakage.

Other examples can be verified with extensions to our logics that can also be proved sound in our framework. For instance, we can support a slightly different relational logic in which the Hoare quadruple is indexed by a pair (ϵ, δ), and interpreted using the lifting from Example 6.6. With this logic, we can study differential privacy of mechanisms such as the exponential mechanism on non-numerical queries [Dwork and Roth 2014], which uses a scoring function that assigns positive values to all possible input/output pairs. Prior work [Barthe et al. 2012], has verified this mechanism using a first-order representation of scoring. However, we can verify a higher-order representation of this mechanism, where the scoring function is passed as an argument to the mechanism.

We can also use HO-EXP to verify examples based on the weakest pre-expectation calculus [Morgan et al. 1996]. One caveat is that many of these examples use arbitrary while loops, which our language does not currently support. This extension would require extending the model as discussed at the end of this section. Other examples, e.g., stability of machine learning algorithms, would require developing a logic for relational pre-expectations. Yet others, e.g. cryptography, would require enriching our logics with additional proof principles that embed notions of cryptographic reductions. In the long run, it would be interesting to support these formalisms with an implementation to mechanize examples.

Unbounded recursion. Our language provides bounded recursion via the monadic fold. An interesting follow-up would be to extend our language with unbounded monadic recursion. For this, we would also need to change the semantic model. One possibility is to use the recently proposed category $\omega\mathbf{QBS}$ [Vákár et al. 2019] to interpret types.

ACKNOWLEDGMENTS

S.K. was supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), Japan Science and Technology Agency. T.S. was supported by JSPS KAKENHI Grant Number 20K19775, Japan. M.G. was supported by NSF awards CCF-2040222 and CCF-1718220.

REFERENCES

Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2017. A relational logic for higher-order programs. *PACMPL* 1, ICFP (2017), 21:1–21:29. https://doi.org/10.1145/3110265

Alejandro Aguirre, Gilles Barthe, Justin Hsu, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2021. A Pre-Expectation Calculus for Probabilistic Sensitivity. *Proc. ACM Program. Lang.* 5, POPL, Article 52 (Jan. 2021), 28 pages. https://doi.org/10.1145/3434333

Alejandro Aguirre and Shin-ya Katsumata. 2020. Weakest Preconditions in Fibrations. In *36th Mathematical Foundations of Programming Semantics Conference, MFPS 2020 (Electronic Notes in Theoretical Computer Science, Vol. 352)*. 5–27. https://doi.org/10.1016/j.entcs.2020.09.002

Robert J Aumann et al. 1961. Borel structures for function spaces. *Illinois Journal of Mathematics* 5, 4 (1961), 614–630. https://doi.org/10.1215/ijm/1255631584

Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, and Pierre-Yves Strub. 2021. Mechanized Proofs of Adversarial Complexity and Application to Universal Composability. Cryptology ePrint Archive, Report 2021/156. https://eprint.iacr.org/2021/156.

Gilles Barthe, Marion Daubignard, Bruce M. Kapron, and Yassine Lakhnech. 2010. Computational indistinguishability logic. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov (Eds.). ACM, 375–386. https://doi.org/10.1145/1866307.1866350

Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. Proving expected sensitivity of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 57:1–57:29. https://doi.org/10.1145/3158145

Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016a. Advanced Probabilistic Couplings for Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 55–67. https://doi.org/10.1145/2976749.2978391

Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella Béguelin. 2014a. Probabilistic relational verification for cryptographic implementations. In *POPL 2014*, Suresh Jagannathan and Peter Sewell (Eds.). https://doi.org/10.1145/2578855.2535847

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016b. A Program Logic for Union Bounds. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy (LIPIcs, Vol. 55)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 107:1–107:15. https://doi.org/10.4230/LIPIcs.ICALP.2016.107

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *POPL 2009, Savannah, GA, USA, January 21-23, 2009*. https://doi.org/10.1145/1480881.1480894

Gilles Barthe, Boris Köpf, Laurent Mauborgne, and Martín Ochoa. 2014b. Leakage Resilience against Concurrent Cache Attacks. In *Proc. 3rd Conference on Principles of Security and Trust (POST '14)*. Springer. https://doi.org/10.1007/978-3-642-54792-8_8

Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012. Probabilistic relational reasoning for differential privacy. In *POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*. https://doi.org/10.1145/2103656.2103670

Nick Benton, Martin Hofmann, and Vivek Nigam. 2014. Abstract effects and proof-relevant logical relations. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 619–632. https://doi.org/10.1145/2535838.2535869

Ales Bizjak and Lars Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *FoSSaCS 2015, London, UK, April 11-18, 2015. Proceedings*. https://doi.org/10.1007/978-3-662-46678-0_18

Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426. https://doi.org/10.1145/362686.362692

Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. 2018. State Separation for Code-Based Game-Playing Proofs. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11274)*, Thomas Peyrin and Steven D. Galbraith (Eds.). Springer, 222–249. https://doi.org/10.1007/978-3-030-03332-3_9

David Clayton, Christopher Patton, and Thomas Shrimpton. 2019. Probabilistic Data Structures in Adversarial Environments. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 1317–1334. https://doi.org/10.1145/3319535.3354235

Raphaëlle Crubillé and Ugo Dal Lago. 2015. Metric Reasoning about λ-Terms: The Affine Case. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 633–644.

https://doi.org/10.1109/LICS.2015.64

Vincent Danos and Thomas Ehrhard. 2011. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.* 209, 6 (2011), 966–991. https://doi.org/10.1016/j.ic.2011.02.001

Dominique Devriese, Lars Birkedal, and Frank Piessens. 2016. Reasoning about Object Capabilities with Logical Relations and Effect Parametricity. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. 147–162. https://doi.org/10.1109/EuroSP.2016.22

Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407. http://dx.doi.org/10.1561/0400000042

Deepak Garg, Jason Franklin, Dilsun Kirli Kaynar, and Anupam Datta. 2010. Compositional System Security with Interface-Confined Adversaries. *Electr. Notes Theor. Comput. Sci.* 265 (2010), 49–71. https://doi.org/10.1016/j.entcs.2010.08.005

Thomas Gerbet, Amrit Kumar, and Cédric Lauradoux. 2015. The Power of Evil Choices in Bloom Filters. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*. IEEE Computer Society, 101–112. https://doi.org/10.1109/DSN.2015.21

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005137

R. Impagliazzo and S. Rudich. 1989. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing, 1989*. ACM, New York, 44–61. https://doi.org/10.1145/73007.73012

Kenneth E. Iverson. 1962. *A Programming Language*. John Wiley & Sons, Inc., USA.

Limin Jia, Shayak Sen, Deepak Garg, and Anupam Datta. 2015. A Logic of Programs with Interface-Confined Code. In *IEEE 28th Computer Security Foundations Symposium (CSF)*. 512–525. https://doi.org/10.1109/CSF.2015.38

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 637–650. https://doi.org/10.1145/2676726.2676980

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs, Vol. 9632. 364–389. https://doi.org/10.1007/978-3-662-49498-1_15

Shin-ya Katsumata. 2014. Parametric Effect Monads and Semantics of Effect Systems. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 633–645. https://doi.org/10.1145/2535838.2535846

Shin-ya Katsumata, Tetsuya Sato, and Tarmo Uustalu. 2018. Codensity Lifting of Monads and its Dual. *Log. Methods Comput. Sci.* 14, 4 (2018). https://doi.org/10.23638/LMCS-14(4:6)2018

Dexter Kozen. 1985. A Probabilistic PDL. 30, 2 (1985), 162–178. https://doi.org/10.1145/800061.808758

Saunders MacLane. 1971. *Categories for the Working Mathematician*. Springer-Verlag, New York. Graduate Texts in Mathematics, Vol. 5.

Kenji Maillard, Catalin Hritcu, Exequiel Rivas, and Antoine Van Muylder. 2020. The next 700 relational program logics. *Proc. ACM Program. Lang.* 4, POPL (2020), 4:1–4:33. https://doi.org/10.1145/3371072

Cristina Matache and Sam Staton. 2019. A Sound and Complete Logic for Algebraic Effects. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11425)*, Mikolaj Bojanczyk and Alex Simpson (Eds.). Springer, 382–399. https://doi.org/10.1007/978-3-030-17127-8_22

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. 18, 3 (1996), 325–353. https://doi.org/10.1145/229542.229547

Aleksandar Nanevski, Anindya Banerjee, and Deepak Garg. 2013. Dependent Type Theory for Verification of Information Flow and Access Control Policies. *ACM Trans. Program. Lang. Syst.* 35, 2 (2013), 6:1–6:41. https://doi.org/10.1145/2491522.2491523

Aleksandar Nanevski, J. Gregory Morrisett, and Lars Birkedal. 2008. Hoare type theory, polymorphism and separation. *J. Funct. Program.* 18, 5-6 (2008), 865–911. https://doi.org/10.1017/S0956796808006953

Moni Naor and Eylon Yogev. 2019. Bloom Filters in Adversarial Environments. *ACM Trans. Algorithms* 15, 3 (2019), 35:1–35:30. https://doi.org/10.1145/3306193

Andrew Pitts and Ian Stark. 1998. Operational Reasoning for Functions with Local State. In *Higher Order Operational Techniques in Semantics*, Andrew Gordon and Andrew Pitts (Eds.). Publications of the Newton Institute, Cambridge University Press, 227–273. http://www.inf.ed.ac.uk/~stark/operfl.html

Gordon Plotkin. 1973. Lambda-definability and logical relations.

Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. 2011. Careful with Composition: Limitations of the Indifferentiability Framework. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6632)*, Kenneth G. Paterson (Ed.). Springer, 487–506. https://doi.org/10.1007/978-3-642-20465-4_27

Tetsuya Sato. 2016. Approximate Relational Hoare Logic for Continuous Random Samplings. In *The Thirty-second Conference on the Mathematical Foundations of Programming Semantics, MFPS 2016, Carnegie Mellon University, Pittsburgh, PA, USA, May 23-26, 2016 (Electronic Notes in Theoretical Computer Science, Vol. 325)*, Lars Birkedal (Ed.). Elsevier, 277–298. https://doi.org/10.1016/j.entcs.2016.09.043

Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. 2019. Formal verification of higher-order probabilistic programs: reasoning about approximation, convergence, Bayesian inference, and optimization. *PACMPL* 3, POPL (2019), 38:1–38:30. https://doi.org/10.1145/3290351

Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. 2017. Denotational Validation of Higher-order Bayesian Inference. *Proc. ACM Program. Lang.* 2, POPL, Article 60 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158148

Joseph Tassarotti and Robert Harper. 2019. A Separation Logic for Concurrent Randomized Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 64 (Jan. 2019), 30 pages. https://doi.org/10.1145/3290377

Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *PACMPL* 3, POPL (2019), 36:1–36:29. https://doi.org/10.1145/3290349