Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System

SATOSHI KURA, Waseda University, Japan

HIROSHI UNNO, Tohoku University, Japan

Verification of higher-order probabilistic programs is a challenging problem. We present a verification method that supports several quantitative properties of higher-order probabilistic programs. Usually, extending verification methods to handle the quantitative aspects of probabilistic programs often entails extensive modifications to existing tools, reducing compatibility with advanced techniques developed for qualitative verification. In contrast, our approach necessitates only small amounts of modification, facilitating the reuse of existing techniques and implementations. On the theoretical side, we propose a dependent refinement type system for a generalised higher-order fixed point logic (HFL). Combined with continuation-passing style encodings of properties into HFL, our dependent refinement type system enables reasoning about several quantitative properties, including weakest pre-expectations, expected costs, moments of cost, and conditional weakest pre-expectations for higher-order probabilistic programs with continuous distributions and conditioning. The soundness of our approach is proved in a general setting using a framework of categorical semantics so that we don't have to repeat similar proofs for each individual problem. On the empirical side, we implement a type checker for our dependent refinement type system that reduces the problem of type checking to constraint solving. We introduce *admissible predicate variables* and *integrable predicate variables* to constrained Horn clauses (CHC) so that we can soundly reason about the least fixed points and samplings from probability distributions. Our implementation demonstrates that existing CHC solvers developed for non-probabilistic programs can be extended to a solver for the extended CHC with only small efforts. We also demonstrate the ability of our type checker to verify various concrete examples.

CCS Concepts: • **Theory of computation** → **Program verification**; **Probabilistic computation**.

Additional Key Words and Phrases: dependent refinement type system, higher-order program

1 INTRODUCTION

Probabilistic programs have been studied from two perspectives: for writing randomised algorithms and for describing probabilistic models and Bayesian inferences. For these purposes, many probabilistic programming languages have been proposed and developed, including Anglican [Wood et al. 2014], LazyPPL [Dash et al. 2023], Church [Goodman et al. 2008], and Hakaru [Narayanan et al. 2016]. To facilitate programming, these languages have several features such as sampling from continuous distributions, conditioning for modelling posterior distributions, and sometimes higher-order functions. On the other hand, these features pose challenges to the verification of probabilistic programs, invalidating many existing verification methods for non-probabilistic programs. Although many studies have been devoted to this challenge [Avanzini et al. 2021; McIver and Morgan 2005; Olmedo et al. 2018; Sato et al. 2019], *automated* verification is still a challenging problem. Furthermore, there are relatively few studies on automated verification of *higher-order* probabilistic programs compared to imperative probabilistic programs.

In this paper, we propose a dependent refinement type system for verifying various properties of higher-order probabilistic programs with continuous distributions (Fig. 1). We consider a generalised

---

Higher-order probabilistic programs
with continuous distributions

CPS encodings of properties [Avanzini et al. 2021; Kura 2023]

Terms of HFL (Section 2)   +   Specifications by refinement types (Section 3)

Our dependent refinement type system (Section 4)
& A type checking algorithm (Section 5,6)

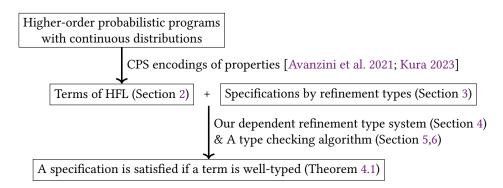A specification is satisfied if a term is well-typed (Theorem 4.1)

Fig. 1. Outline of our approach.

higher-order fixed point logic (HFL). Given a probabilistic program, we encode properties of the program as terms of the HFL using CPS (Continuation-Passing Style) transformations. Such encodings are available for several properties, including weakest pre-expectations, expected costs, moments of cost, and conditional weakest pre-expectations [Avanzini et al. 2021; Kura 2023]. Then, we give a specification of a probabilistic program as a refinement type. Finally, we type-check the term against the specification using our dependent refinement type system to verify whether the specification is satisfied. In our approach, only top-level declarations require annotations of refinement types, and such annotations need *not* be inductive because our type checker can automatically synthesise inductive invariants.

Our main contribution is the *automation* of the CPS-based verification methods [Avanzini et al. 2021; Kura 2023] for higher-order probabilistic programs, which is achieved by introducing a new dependent refinement type system for HFL. The only method that we are aware of for reasoning about CPS-transformed probabilistic programs is a program logic EHOL proposed by [Avanzini et al. 2021]. However, it is not obvious how to automate reasoning by EHOL since logical formulas considered in EHOL contain higher-order terms. Therefore, we developed a more automatable method, namely, a refinement type system for CPS-transformed programs. Using our refinement type system, we just need to solve constraints that contain only first-order terms. Moreover, our refinement type system is more general than EHOL in the sense that it can reason about properties beyond the expected cost. Our work also supports continuous distributions, which are not supported by [Avanzini et al. 2021]. The CPS-based approach has the following advantages over other approaches for higher-order probabilistic programs [Avanzini et al. 2023; Beutner and Ong 2021]: (1) it provides a unified framework for various verification problems, and (2) it allows a reduction to general verification frameworks developed for non-probabilistic programs (e.g. CHC solving [Bjørner et al. 2015]).

Our approach enhances *reusability* of both theory and implementation. From a theoretical point of view, our design of the dependent refinement type system (Section 4) is guided by categorical semantics, which naturally yields a soundness theorem (Theorem 4.1) that holds for various situations. This general design leads to novel automated verification methods for several properties (including the expected cost, the weakest pre-expectation, the cost moment, and the conditional weakest pre-expectation of probabilistic programs) while minimising the need for individual proofs for each property. Although our paper specifically concentrates on probabilistic programs, our theoretical framework also goes beyond the verification of probabilistic programs (e.g. HFL model checking of event sequences [Kobayashi et al. 2018; Kura 2023]). From the point of view of implementation, our approach can seamlessly integrate with existing type checkers for dependent refinement type

systems designed for non-probabilistic programs. This extension to probabilistic programs involves only two key modifications to constraint solvers: supporting admissible predicates for reasoning about fixed points (Section 6.2) and supporting integration operators for reasoning about continuous distributions (Section 6.3). These two modifications can be implemented as simple extensions of existing solvers, as we will explain in Section 5, and the remaining components of type checkers can be reused from those for non-probabilistic programs. This simplicity has the advantage that we can easily incorporate advanced techniques and sophisticated constraint solvers developed for non-probabilistic programs.

1.1 Refinement Types After CPS Transformations

Our dependent refinement type system is intended to be used in combination with CPS-based methods to translate a given program to a term that describes a property of the program. In the context of verification of probabilistic programs, [Avanzini et al. 2021] proposed a CPS transformation that translates a higher-order probabilistic program to a term of pure lambda calculus with fixed points such that the denotation of the term is equal to the expected cost of the given probabilistic program.

Example 1.1 (the expected cost of coin flipping). For example, consider the following program that keeps flipping a fair coin until we get a head while counting the number of tails.

$$\textbf{let rec } \text{coin } x = (\text{coin } ())^{\checkmark} \oplus_{1/2} () \textbf{ in } \text{coin } ()$$

Here, $\oplus_{1/2}$ means probabilistic branching that takes each branch with probability $1/2$, and we consider the expected number of occurrences of $(-)^{\checkmark}$ in execution traces. Following [Avanzini et al. 2021], we apply a CPS transformation and pass $\lambda y.0$ as a continuation. Then, we obtain the following term, whose denotation is the expected cost of the probabilistic program above.

$$\textbf{let fix } \text{coin } x \; k = 1/2 \cdot (1 + \text{coin } () \; k) + 1/2 \cdot k \; () \textbf{ in } \text{coin } () \; (\lambda y.0) \tag{1}$$

Note that coin has type $\text{coin} : \textbf{unit} \rightarrow (\textbf{unit} \rightarrow \textbf{real}^{\infty}_{\geq 0}) \rightarrow \textbf{real}^{\infty}_{\geq 0}$ where $\textbf{real}^{\infty}_{\geq 0}$ is a type of non-negative extended real numbers in $[0, \infty]$. Note also that in the above CPS transformation, $\oplus_p$ is mapped to $p \cdot (-) + (1 - p) \cdot (-)$, and $(-)^{\checkmark}$ is mapped to $1 + (-)$. Intuitively, the term (1) represents the limit of the converging sequence below.

$$c_0 = 0, \quad c_1 = 1/2 \cdot (1 + c_0) + 1/2 \cdot 0 = 1/2, \quad c_2 = 1/2 \cdot (1 + c_1) + 1/2 \cdot 0 = 3/4, \quad \cdots \rightarrow 1$$

Technically, we get the converging sequence above because the interpretation of **let fix** is different from the standard interpretation of recursion: **let fix** is the least fixed point with respect to the standard order on $[0, \infty]$. □

Recently, [Kura 2023] generalised this CPS translation to a wide class of weakest precondition transformers: we can translate a given higher-order program into a term of a generalised higher-order fixed point logic (HFL) so that the term describes a certain kind of properties of the given program. In terms of verification of probabilistic programs, this generalisation specifically subsumes four properties: weakest pre-expectations, expected costs, moments of cost, and conditional weakest pre-expectations. However, how to reason about HFL terms was out of the scope of these studies.

To automatically reason about HFL terms, we propose a dependent refinement type system for HFL. For example, consider the term (1) in Example 1.1. As a specification, we may consider the following refinement type, which reads "the expected cost r satisfies $r \leq 1$" or more simply "1 is an upper bound of the expected cost", assuming that $\lambda y.0 : \textbf{unit} \rightarrow \{r : \textbf{real}^{\infty}_{\geq 0} \mid r = 0\}$ is passed as the second argument.

$$\text{coin} : (x : \textbf{unit}) \rightarrow (k : \textbf{unit} \rightarrow \{r : \textbf{real}^{\infty}_{\geq 0} \mid r = 0\}) \rightarrow \{r : \textbf{real}^{\infty}_{\geq 0} \mid r \leq 1\} \tag{2}$$

To verify that (2) is a correct refinement type for the term (1), we provide a set of typing rules for dependent refinement types. We also prove that typing rules are sound. By utilising mathematical abstraction via categorical semantics, the soundness theorem is proved in a general situation, subsuming any instance supported by the framework of [Kura 2023]. Therefore, we obtain verification methods for higher-order probabilistic programs simply as special cases.

Our mathematical abstraction makes it easier to extend the verification method limited to discrete distributions [Avanzini et al. 2021] to a method that allows continuous distributions too. When we apply our approach to continuous distributions, we only need to check if a model for continuous distributions satisfies a certain set of axioms that make our soundness theorem hold. In fact, there exists such a model: the category ω**QBS** of ω-quasi-Borel spaces [Vákár et al. 2019]. This process is much easier than repeating the whole soundness proof for the case of continuous distributions.

1.2 Key Modifications of Type Checkers and Constraints Solvers

We implement a type checker for our dependent refinement type system as an extension of an existing type checker RCaml[1] for non-probabilistic programs. There are two key modifications involved in this extension.

1.2.1 Admissible Predicates. As we explained in Example 1.1, the semantics of *fixed points* let fix in our HFL is given by a different order relation from the standard semantics of *recursion*. In the following, we distinguish *fixed points* and *recursion* as different concepts. Because of the difference above, standard typing rules for recursion are unsound for fixed points (as we will explain in Section 4.3). For example, the standard typing rule for partial correctness of recursion can derive the following type for Example 1.1.

$$\text{coin} : (x : \textbf{unit}) \rightarrow (k : \textbf{unit} \rightarrow \{r : \textbf{real}^\infty_{\geq 0} \mid r = 0\}) \rightarrow \{r : \textbf{real}^\infty_{\geq 0} \mid r < 1\}$$

However, this is incorrect because the true expected cost of Example 1.1 is $r = 1$.

Following EHOL [Avanzini et al. 2021], we adapt a standard typing rule for recursion by imposing the admissibility condition on a predicate on the result type of a fixed point. This adaptation is essential for obtaining a sound typing rule, especially when reasoning about quantitative properties expressed as the least fixed point. Technically, a predicate $P \subseteq [0, \infty]$ is *admissible* if the bottom element $0 \in [0, \infty]$ satisfies P and the predicate is closed under the supremum $\sup_n x_n$ of any ω-chain $x_0 \leq x_1 \leq \dots$ such that $x_n \in P$ for any n. For example, (2) is a correct type for Example 1.1 because $r \leq 1$ is admissible, but we cannot replace $r \leq 1$ with $r < 1$ because $r < 1$ is not admissible (not closed under sup). Note that $r = 1$ is (unfortunately) not admissible either and thus cannot be used here. In general, it is difficult to reason about the exact value or lower bounds in probabilistic program verification [Beutner and Ong 2021; Feng et al. 2023; Hark et al. 2020; McIver and Morgan 2005], which is why most of the studies (including ours) focus on upper bounds instead of the exact value of the expected cost.

To implement fixed points in our type checker, we extend our type checker and the constraint solver used in our type checker so that only admissible predicates are synthesised for fixed points. This extension doesn't break the basic design of the existing implementation: what we need is (1) in the type checker, to implement the typing rule for fixed points and (2) in the constraint solver (specifically, a solver for constrained Horn clauses), to support a new type of predicate variables for admissible predicates, which we call *admissible predicate variables*. The former is rather straightforward. For the latter, we first consider syntactic criteria to check whether a predicate is admissible or not. Then, we restrict the search space for admissible predicates by the syntactic criteria.

---

[1]https://github.com/hiroshi-unno/coar

The novelties here are (1) incorporating admissible predicates explicitly into a refinement type system and (2) supporting admissible predicates in a type checker. Technically, ensuring admissibility is important for *any* refinement type system when proving the soundness of typing rules for recursion or fixed points. However, there hasn't been any refinement type system that requires admissibility explicitly because it is typically straightforward to ensure admissibility when reasoning about recursion, as we explain in Section 4. For reasoning about fixed points, we propose a novel type checker that supports admissible predicates. To the best of our knowledge, no other type checker of this nature exists.

1.2.2   Integration Operators. Although our dependent refinement type system is proved sound in a general situation, it is, to some extent, inevitable that we have to consider how to deal with basic (or primitive) operators, which may vary depending on programming languages targeted for verification. Specifically, this is the case for integration operators for continuous distributions. Since we consider probabilistic programs with continuous distributions (e.g. the uniform distribution over the unit interval $[0, 1]$), we have to include corresponding integration operators (e.g. $\int_0^1 (-) \, \mathrm{d}x$) in our HFL so that we can reason about properties defined by expected values (see Section 3.4). However, integration operators are usually not supported by constraints solvers nor even considered in dependent refinement type systems.

To support integration operators, we provide a typing rule for an integration operator in Section 4.3.3. Then, we introduce another new type of predicate variables, which we call *integrable predicate variables* so that constraints for integration operators can be expressed appropriately. Finally, we extend our constraint solver to support integrable predicate variables. Our extension is rather simple: we restrict templates for integrands to affine expressions so that we can easily compute integrals by linearity of integration.

1.3   Summary of Contributions

- We provide a dependent refinement type system for a generalised HFL. We proved its soundness for a general class of models. Combined with CPS-based methods, our approach can verify the following properties of higher-order probabilistic programs with continuous distributions: weakest pre-expectations, expected costs, moments of cost, and conditional weakest pre-expectations.
- By introducing two new types of predicate variables (i.e. admissible and integrable predicate variables) to CHC, we present a reduction from the type-checking problem for our dependent refinement type system to constraint solving for the extended CHC. We also propose an extension of a CHC solver that supports admissible and integrable predicate variables by considering appropriate templates. A notable advantage of our approach lies in the ability to reuse most of the basic design of implementations except for the two types of predicate variables. This facilitates the seamless integration of advanced techniques developed for the verification of non-probabilistic programs.
- We implement a type checker and demonstrate its ability to verify various properties of benchmark programs. Our implementation solved 11 out of 14 benchmarks ranging over four kinds of verification problems for higher-order probabilistic programs. No other tool even supports as many kinds of benchmarks as ours. Since our approach does not depend on a choice of constraint solvers, there is considerable freedom in the choice of generic constraint solvers. Moreover, such constraint solvers need *not* be tailored for probabilistic programs thanks to the CPS transformation. Therefore, future developments of generic constraint solvers will potentially improve experimental results.

Table 1. Truth values for reasoning about probabilistic programs. Truth values are equipped with partial orders so that we can define least fixed points. The superscript $(-)^{\mathrm{op}}$ indicates that we consider the opposite partial order (lfp w.r.t. the opposite order = gfp w.r.t. the original order).

Property	Truth values (**Prop**)
Weakest pre-expectation [McIver and Morgan 2005]	$[0, \infty]$
Expected cost [Kaminski et al. 2018]	$[0, \infty]$
Cost moment [Kura et al. 2019]	$[0, \infty]^n$
Conditional weakest pre-expectation [Olmedo et al. 2018]	$[0, \infty] \times [0, 1]^{\mathrm{op}}$

$$\sigma, \tau := \mathbf{Prop} \mid b \mid 1 \mid \sigma \times \tau \mid 0 \mid \sigma + \tau \mid \sigma \to \mathbf{Prop} \qquad \Gamma := \cdot \mid \Gamma, x : \tau$$

$M, N := x$	variable	$\mid \mathbf{op}\ M$	basic operators
$\mid ()\quad \mid (M, N)$	0-tuple, 2-tuple	$\mid \pi_i\ M$	projection ($i = 1, 2$)
$\mid \delta(M)$	nullary case analysis	$\mid \iota_i\ M$	injection ($i = 1, 2$)
$\mid \delta(M, x_1 : \sigma_1.N_1, x_2 : \sigma_2.N_2)$	binary case analysis	$\mid \lambda x : \sigma.M$	lambda abstraction
$\mid \mathbf{fix}\ (f : \sigma \to \mathbf{Prop}).M$	fixed point	$\mid M\ N$	function application

Fig. 2. Simple types σ, τ; simple contexts Γ; and terms M, N. In the definition above, x ranges over variables, b over base types in **Base**, and **op** over basic operators in **BO**. We assume variables in a context are mutually distinct.

S-BasicOp
$$\frac{\Gamma \vdash M : \mathrm{ar}(\mathbf{op})}{\Gamma \vdash \mathbf{op}\ M : \mathrm{car}(\mathbf{op})}$$

S-Abs
$$\frac{\Gamma, x : \sigma \vdash M : \mathbf{Prop}}{\Gamma \vdash \lambda x : \sigma.M : \sigma \to \mathbf{Prop}}$$

S-Fix
$$\frac{\Gamma, f : \sigma \to \mathbf{Prop}, \vdash M : \sigma \to \mathbf{Prop}}{\Gamma \vdash \mathbf{fix}\ (f : \sigma \to \mathbf{Prop}).M : \sigma \to \mathbf{Prop}}$$

Fig. 3. Selected typing rules.

1.4 Organization of the Paper

In Section 2, we review the definition of the generalised higher-order fixed point logic λ_{HFL} [Kura 2023]. In Section 3, we explain that CPS transformations (also proposed in [Kura 2023]) from probabilistic programs to λ_{HFL} can capture various interesting properties. We also present our idea of using refinement types as specifications of probabilistic programs. In Section 4, we present our dependent refinement type system and its soundness. In Section 5, we explain a reduction from the type-checking problem to constraint solving where we introduce an extension of CHC. In Section 6, we show how to implement a constraint solver by modifying an existing CHC solver and present experimental results. We discuss related work in Section 7. Section 8 contains conclusions and future work.

2 A GENERALISED HIGHER ORDER FIXED POINT LOGIC

In this section, we review the definition of a generalised higher-order fixed point logic λ_{HFL} [Kura 2023] that can describe properties of higher-order programs. We will explain later in Section 3 how to encode properties of a given program into a term of λ_{HFL}.

2.1 Syntax

Following [Kura 2023], λ_{HFL} is defined as a simply typed lambda calculus with fixed points. We define *simple types*, *simple contexts*, and *terms* of λ_{HFL} as in Fig. 2. In the definition of simple types, **Prop** is a type of truth values, which need not be boolean values. The meaning of **Prop** depends on the verification problems we consider. When verifying probabilistic programs, it is common to use quantitative values as "truth values" (see Table 1 and Section 3). For example, we use non-negative extended real numbers $[0, \infty]$ as truth values for expected cost analyses, in which case, λ_{HFL} is essentially the same as the target language of [Avanzini et al. 2021]. The unit type 1 (or sometimes written as **unit**) is inhabited by only one value, and the empty type 0 is inhabited by no value. We use the sum type $1 + 1$ as a type of boolean values where **true** $:= \iota_1$ () and **false** $:= \iota_2$ (). Although λ_{HFL} is designed as a higher-order logic, we *never* call a term of λ_{HFL} a *formula* to distinguish terms from first-order formulas defined in Section 4.

We have two parameters for syntax: a set **Base** of base types and a set **BO** of basic operators. Each element in **BO** is a triple of the form $(\mathbf{op} : \text{ar}(\mathbf{op}) \rightarrow \text{car}(\mathbf{op})) \in \mathbf{BO}$ where $\text{ar}(\mathbf{op})$ and $\text{car}(\mathbf{op})$ are types called an *arity* and a *coarity* of **op**, respectively. For example, we often include type **int** of integers and type **real** of real numbers as base types in **Base**. We can introduce constants of type **int** to λ_{HFL} by adding a basic operator $n : 1 \rightarrow \mathbf{int}$ for any $n \in \mathbb{Z}$, and similarly for constants of other types. Basic arithmetic operators like $(+_{\mathbf{int}}) : \mathbf{int} \times \mathbf{int} \rightarrow \mathbf{int}$ and comparison operator like $(\leq_{\mathbf{int}}) : \mathbf{int} \times \mathbf{int} \rightarrow 1 + 1$ are also examples of basic operators.

The type **Prop** plays a special role in the typing system of λ_{HFL}. *Well-typed term* $\Gamma \vdash M : \tau$ are defined by rather standard typing rules except that the codomain of function types are restricted to **Prop** (see Fig. 3), which leads to a restriction of lambda abstractions and fixed points. This is not too restrictive because we always get this form of lambda abstractions and fixed points by applying CPS transformations with **Prop** as an answer type.

We introduce several syntactic sugars as follows. We define if expressions **if** M **then** N_1 **else** $N_2 := \delta(M, x_1.N_1, x_2.N_2)$ by case analyses where x_1 and x_2 are fresh variables since $1 + 1$ is used as the type of boolean values. For any constant $\mathbf{op} : 1 \rightarrow \sigma$, we usually omit argument () and write $\mathbf{op} := \mathbf{op}$ (). We often use infix notation for arithmetic operators and comparison operators: more formally, if $\mathbf{op} : \sigma_1 \times \sigma_2 \rightarrow \tau$ is a basic operator where $\sigma_1, \sigma_2 \in \mathbf{Base} \cup \{\mathbf{Prop}\}$, we define $M \mathbf{\ op\ } N := \mathbf{op} \ (M, N)$. We define let-fix expressions by **let fix** $f \ x = M$ **in** $N := (\lambda f.N) \ (\mathbf{fix} \ f.\lambda x.M)$.

The restriction of function types forces us to use uncurried functions like $f : \mathbf{int} \times \mathbf{int} \rightarrow \mathbf{Prop}$ instead of curried one $f : \mathbf{int} \rightarrow (\mathbf{int} \rightarrow \mathbf{Prop})$. For convenience, we think of a curried function as a syntactic sugar for the corresponding uncurried function: $\lambda x.\lambda y.M := \lambda(x, y).M := \lambda z.M[\pi_1 \ z/x, \pi_2 \ z/y]$ where M is a term of type **Prop** (use η-expansion to M if needed). Correspondingly, a partial application of $M : \sigma \times \tau \rightarrow \mathbf{Prop}$ to $N : \sigma$ is understood as $\lambda y : \tau.M \ (N, y) : \tau \rightarrow \mathbf{Prop}$. We use a similar syntactic sugar for fixed points too.

2.2 Denotational Semantics

Technically, the denotational semantics for λ_{HFL} is defined in a general way for a certain class of categorical models, which we call λ_{HFL}-*models*. Although this gives flexibility of choosing an appropriate λ_{HFL}-model for each problem of program verification, explaining λ_{HFL}-models in the most general form requires a certain level of familiarity with category theory. For the sake of non-category theorists, we restrict our description in this section to a specific λ_{HFL}-model, namely, the category $\omega\mathbf{CPO}$ of ωcpos and Scott-continuous functions, using as few terminology of category theory as possible. This restriction to $\omega\mathbf{CPO}$ is just for simplicity of explanation: our soundness theorem in Section 4 holds for any λ_{HFL}-model, including $\omega\mathbf{QBS}$ models used for

continuous distributions in Section 3. Denotational semantics in general λ_{HFL}-models can be found in Appendix A.

We recall basic definitions of ωcpos and Scott-continuous functions. An ω-*chain* in a partially ordered set $(X, \leq)$ is a sequence $\{x_n \in X\}_n$ of elements in X such that $x_n \leq x_{n+1}$ for any n. An *ωcpo* is a pair $(X, \leq_X)$ of a set X and a partial order $\leq_X$ such that the supremum of any ω-chain in X exists. Note that we often leave $\leq_X$ implicit and write $X = (X, \leq_X)$. A *Scott-continuous function* between two ωcpos $(X, \leq_X)$, $(Y, \leq_Y)$ is a monotone function $f : X \to Y$ that preserves supremum of ω-chains. Let $\omega\mathbf{CPO}$ be the category of ωcpos and Scott-continuous functions. Recall that for any ωcpo $(X, \leq_X)$, $(Y, \leq_Y)$, we have their product $(X, \leq_X) \times (Y, \leq_Y) = (X \times Y, \leq_{X \times Y})$, coproduct (or sum) $(X, \leq_X) + (Y, \leq_Y) = (X + Y, \leq_{X+Y})$, and function space $(X, \leq_X) \Rightarrow (Y, \leq_Y) = (\{f : X \to Y \mid f \text{ is Scott-continuous}\}, \leq_{X \Rightarrow Y})$ in $\omega\mathbf{CPO}$ where partial orders are defined by $(x, y) \leq_{X \times Y} (x', y')$ iff $x \leq_X x' \wedge y \leq_Y y'$; $x \leq_{X+Y} x'$ iff $x \leq_X x' \vee x \leq_Y x'$; and $f \leq_{X \Rightarrow Y} g$ iff $\forall x \in X, f(x) \leq_Y g(x)$.

Now, we define the interpretation of types. Suppose that interpretations of each base type $b \in \mathbf{Base}$ and the type $\mathbf{Prop}$ of truth values are given: let $[\![b]\!]$ and $[\![\mathbf{Prop}]\!]$ be ωcpos that give an interpretation of b and $\mathbf{Prop}$, respectively. For example, we usually interpret $\mathbf{int}$ and $\mathbf{real}$ by ωcpos $(\mathbb{Z}, =)$ and $(\mathbb{R}, =)$ equipped with discrete orders. Then, we extend $[\![-]\!]$ to the interpretation of types as follows.

$$[\![b]\!] := [\![b]\!] \qquad [\![\mathbf{Prop}]\!] := [\![\mathbf{Prop}]\!] \qquad [\![0]\!] := (\emptyset, \emptyset) \qquad [\![\sigma + \tau]\!] := [\![\sigma]\!] + [\![\tau]\!]$$
$$[\![1]\!] := (\{\star\}, =) \qquad [\![\sigma \times \tau]\!] := [\![\sigma]\!] \times [\![\tau]\!] \qquad [\![\sigma \to \mathbf{Prop}]\!] := [\![\sigma]\!] \Rightarrow [\![\mathbf{Prop}]\!]$$

A context is interpreted by $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n]\!] := \prod_{i=1}^{n} [\![\sigma_i]\!]$, or equivalently, by the set of mappings defined below. Note that the interpretation $[\![\Gamma]\!]$ of a context is an ωcpo whose partial order is given by the pointwise order: $\gamma \leq \gamma'$ if and only if $\gamma(x) \leq \gamma'(x)$ for any x.

$$[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n]\!] := \{\gamma \mid \mathrm{dom}(\gamma) = \{x_1, \ldots, x_n\} \wedge \forall (x_i : \sigma_i), \gamma(x_i) \in [\![\sigma_i]\!]\}$$

The empty mapping for the empty context is denoted by $\emptyset$.

We define the interpretation of terms as follows. Suppose that an interpretation of each basic operator in $\mathbf{BO}$ is given: for each basic operator $\mathbf{op} : \sigma \rightharpoonup \tau$ in $\mathbf{BO}$, let $[\![\mathbf{op}]\!] : [\![\sigma]\!] \to [\![\tau]\!]$ be a Scott-continuous function that gives an interpretation of $\mathbf{op}$. For any well-typed term $\Gamma \vdash M : \sigma$, the interpretation is defined as a Scott-continuous function $[\![\Gamma \vdash M : \sigma]\!] : [\![\Gamma]\!] \to [\![\sigma]\!]$ (we often write $[\![M]\!] = [\![\Gamma \vdash M : \sigma]\!]$ for simplicity). Most parts of the definition of $[\![M]\!]$ are standard (see Appendix A for details). The interpretation of $\Gamma \vdash \mathbf{fix}\ f.M : \sigma \to \mathbf{Prop}$ is defined by the least fixed point of $[\![M]\!](\gamma[f \mapsto (-)]) : [\![\sigma]\!] \Rightarrow [\![\mathbf{Prop}]\!] \to [\![\sigma]\!] \Rightarrow [\![\mathbf{Prop}]\!]$ where $\gamma \in [\![\Gamma]\!]$.

$$[\![\mathbf{fix}\ (f : \sigma \to \mathbf{Prop}).M]\!](\gamma) = \mathrm{lfp}^{[\![\sigma]\!] \Rightarrow [\![\mathbf{Prop}]\!]}[\![M]\!](\gamma[f \mapsto (-)])$$

For example, if $[\![\mathbf{Prop}]\!] = [0, 1]^{\mathrm{op}} = ([0, 1], \geq)$, then a fixed point $\mathbf{fix}\ f.M$ is interpreted by the lfp with respect to $[0, 1]^{\mathrm{op}} = ([0, 1], \geq)$, which is the gfp with respect to $[0, 1] = ([0, 1], \leq)$.

Note that how we interpret the type $\mathbf{Prop}$, base types in $\mathbf{Base}$, and basic operators in $\mathbf{BO}$ is considered as a part of a λ_{HFL}-model. Technically, a λ_{HFL}-model is defined as a combination of (1) a category $\mathbb{C}$ with sufficient structures (product / coproduct / function space constructions and fixed points) and (2) an assignment of an interpretation to each base type, each basic operator, and $\mathbf{Prop}$ (see Appendix A).

3 ENCODING VARIOUS PROBLEMS TO HFL TERMS

In this section, we show several concrete verification problems for probabilistic programs. We review CPS-based encodings proposed in [Kura 2023] and exemplify how various properties of probabilistic programs can be translated to λ_{HFL}. We also explain λ_{HFL}-models used for each verification problem and specifically how $\mathbf{Prop}$ is interpreted in each λ_{HFL}-model. To motivate the dependent refinement

type system in Section 4, we describe how refinement types are used to give specifications of probabilistic programs.

3.1 A Brief Review of the CPS-Based Encodings

The correctness of the CPS-based encodings below is guaranteed by the general framework proposed by [Kura 2023], which we review here to improve self-containedness. The purpose of the framework is to obtain the weakest precondition of a given effectful program as a term of HFL. This is achieved by considering a CPS transformation that translates an effectful program M into a term of HFL $M^\sharp$. As a language for effectful programs, they consider a functional programming language with recursion and algebraic effects. Intuitively, algebraic effects are primitive operations for causing computational effects. Examples of algebraic effects include a probabilistic branching operation, an operation for sampling from a probability distribution, and the tick operator for increasing cost. Note that although this paper focuses on probabilistic programs, the CPS-based encodings can be applied to more general algebraic effects, as detailed in [Kura 2023].

On the semantic side, a program M is interpreted as a morphism $[\![M]\!] : X \to TY$ in a category $\mathbb{C}$ where T is a strong monad that represents computational effects. Terms of HFL are interpreted in the same category $\mathbb{C}$ as a pure simply typed lambda calculus as we explained in Section 2. The weakest precondition for $f : X \to TY$ is defined as follows. Let $\Omega \in \mathbb{C}$ be an object that represents a set of truth values (e.g. $\Omega = \{0, 1\}$ or $\Omega = [0, \infty]$). Then, the postcondition is given by $p : Y \to \Omega$. Assuming that we have an Eilenberg–Moore algebra $\zeta : T\Omega \to \Omega$, the weakest precondition is defined by $\mathrm{wp}[f](p) = \zeta \circ Tp \circ f : X \to \Omega$. It is known that many properties of programs can be expressed using weakest preconditions if we choose $\zeta : T\Omega \to \Omega$ appropriately [Aguirre et al. 2022]. For example, the weakest pre-expectation of a probabilistic program M is given by $\mathrm{wp}[[\![M]\!]](p)$ where T is the probability distribution monad, $\Omega = [0, \infty]$, and $\zeta : T\Omega \to \Omega$ is the function defined by the expected value $\zeta(\mu) \coloneqq \mathbb{E}[\mu]$ of $\mu \in T[0, \infty]$.

Now, the CPS transformation gives the weakest precondition in the following sense.

THEOREM 3.1 (FROM [KURA 2023]). *Assume that $\mathbb{C}$ is a "good" model (which is almost the same as requiring $\mathbb{C}$ to be a λ_{HFL}-model, but see [Kura 2023] for details). For a well typed program $\Gamma \vdash M : \rho$ and a postcondition (a term of HFL) $x : \rho \vdash P : \mathbf{Prop}$, we have $\mathrm{wp}^\zeta[[\![M]\!]]([\![P]\!]) = [\![M^\sharp (\lambda x : \rho.P)]\!]$ if types in Γ and ρ are constructed without $\to$. Here, $\Gamma \vdash M^\sharp : (\rho \to \mathbf{Prop}) \to \mathbf{Prop}$ is the term obtained by applying the CPS transformation to M.*

Note that the restriction on Γ and ρ does *not* exclude the use of lambda abstraction or recursion in M.

Due to the nature of the CPS-based encoding, our method is inherently constrained to specifications presented as refinement types on the *weakest precondition transformer* (WPT) of a given program. Some properties are difficult to specify using WPTs although we can express many important properties using WPTs, as we will explain in the rest of this section. For example, it is difficult to express non-functional, temporal liveness properties as WPTs, and thus, such properties are out of the scope of our approach.

3.2 A Model for Higher Order Probabilistic Programs with Continuous Distributions

Technically, the category $\omega\mathbf{CPO}$ of ωcpos is not sufficient to model continuous distributions like uniform distributions and Gaussian distributions. To interpret continuous distribution, we use the category $\omega\mathbf{QBS}$ [Vákár et al. 2019] of ωqbses as a λ_{HFL}-model. Intuitively, an ωqbs is a set X equipped with both an ωcpo structure $(X, \leq_X)$ and a quasi-Borel-space structure (X, M_X) in a compatible way (see [Heunen et al. 2017; Vákár et al. 2019] for what M_X means). Quasi-Borel spaces are a generalisation of measurable spaces [Heunen et al. 2017]. Compared to the traditional

measure-theoretic treatment of continuous distributions, quasi-Borel spaces have the advantage that quasi-Borel spaces are closed under the construction of function spaces: the function space $X \Rightarrow Y$ between two ωqbses X, Y is again an ωqbs. Since ω**QBS** has sufficient categorical structures as a λ_{HFL}-model, denotational semantics of λ_{HFL} in ω**QBS** is defined in the same way as the case of the ω**CPO** model explained in Section 2, and more importantly, our soundness theorem in Section 4 is also true for ω**QBS**. To keep our explanation simple, we often leave quasi-Borel-space structures implicit in this section and just write ωcpo structures even when we use ω**QBS** as a model (see Appendix A for details).

3.3 Expected Cost

We consider the problem of estimating the expected cost of higher-order probabilistic programs. This problem is studied in [Avanzini et al. 2021]. Our cost model is given by explicit *tick operators* $(-)^{\checkmark}$, which increment the accumulated cost and do nothing else. We consider the expected number of tick operators used until a given program terminates. Given a probabilistic program, we can translate it to a term M of λ_{HFL} whose interpretation $[\![M]\!]$ is the expected cost [Avanzini et al. 2021]. Specifically, we use a CPS transformation that maps each occurrence of the tick operator $(-)^{\checkmark}$ to $1 + (-)$ and each probabilistic branching $\oplus_p$ to $p \cdot (-) + (1 - p) \cdot (-)$ (see Example 1.1).

Terms of λ_{HFL} obtained from the CPS transformation above is interpreted in ω**QBS**. The type **Prop** of truth values is interpreted by $[0, \infty]$ with the standard order whereas base types are interpreted by sets with discrete orders (and with the standard σ-algebra structures).

$$[\![\textbf{Prop}]\!] = ([0, \infty], \leq) \qquad [\![\textbf{int}]\!] = (\mathbb{Z}, =) \qquad [\![\textbf{real}]\!] = (\mathbb{R}, =)$$

The remaining problem here is how to reason about the term of λ_{HFL} obtained from the CPS transformation above. We can give a specification about the expected cost as a refinement type for the term of λ_{HFL}. For example, the specification for Example 1.1 can be given as (2), which reads "an upper bound of the expected cost is 1". To verify the correctness of such refinement types, we will provide a dependent refinement type system in Section 4.

3.4 Continuous Distributions

Consider the expected cost analysis for a probabilistic program with continuous distributions. To reason about continuous distribution, we add integration operators to λ_{HFL} as a basic operator. For simplicity, we focus on the uniform distribution over the unit interval $[0, 1]$. Then, the integration operator for the uniform distribution is given as follows.

$$\textbf{unif} : (\textbf{real} \to \textbf{Prop}) \to \textbf{Prop} \qquad [\![\textbf{unif}]\!](f) = \int_0^1 f(x) \, dx \qquad (3)$$

Example 3.2. Consider the expected cost of a continuous variant of random walks.

$$\textbf{let rec } \text{rw } x = \textbf{if } x \geq 0 \textbf{ then } (y \leftarrow \textbf{uniform}; (\text{rw } (x + 3 \cdot y - 2))^{\checkmark}) \textbf{ else } () \textbf{ in } \text{rw } 1$$

Here, **uniform** is the uniform distribution over $[0, 1]$ The argument x of the function rw : **real** $\to$ **unit** is the current position, and $x = 1$ is the initial position. For each step, the program samples y from the uniform distribution over $[0, 1]$, transforms y into a sample $y' = 3 \cdot y - 2$ from the uniform distribution over $[-2, 1]$, and then adds y' to the current position x until x becomes negative. To reason about the expected cost, we apply a CPS transformation and obtain the following.

$$\textbf{let fix } \text{rw } x \, k = \textbf{if } x \geq 0 \textbf{ then unif } (\lambda y.1 + \text{rw } (x + 3 \cdot y - 2) \, k) \textbf{ else } k () \textbf{ in } \text{rw } 1 \, (\lambda r.0) \quad (4)$$

Here, the uniform distribution **uniform** is translated to the corresponding integration operator **unif**. The interpretation of (4) gives the expected cost of the random walk. $\qquad\qquad\square$

The expected cost of "rw x" in Example 3.2 is upper-bounded by $2 \cdot x + 4$. Using refinement types, we can specify this as follows.

$$\text{rw} : (x : \{x : \textbf{real} \mid x \geq -2\}) \rightarrow (k : \textbf{unit} \rightarrow \{r : \textbf{Prop} \mid r = 0\}) \rightarrow \{r : \textbf{Prop} \mid r \leq |2 \cdot x + 4|\} \quad (5)$$

Here, $|-| : \textbf{real} \rightharpoonup \textbf{Prop}$ is a basic operator that returns the absolute value of a real number, which is used here just to ensure that the type of $|2 \cdot x + 4|$ is **Prop**.

3.5 Weakest Pre-Expectation

The weakest pre-expectation for probabilistic programs [McIver and Morgan 2005] is a quantitative generalisation of the weakest precondition for non-probabilistic programs. Instead of using boolean predicates as pre-/post-conditions, the weakest pre-expectation transformer uses $[0, \infty]$-valued predicates as *pre-/post-expectations*. Intuitively, the weakest pre-expectation wp[prog](post) is the expected value of the given post-expectation post with respect to the probability distribution of outputs of the program prog. For example, termination probability is a special case of the weakest pre-expectation: if the post-expectation is the constant 1, then the weakest pre-expectation is the termination probability of a given program.

The weakest pre-expectation can be generalised to higher-order probabilistic programs. In fact, we obtain the weakest pre-expectation transformer by just ignoring tick operators when applying the CPS transformation for expected cost analyses in Section 3.3. The term obtained from the CPS transformation takes a post-expectation as a continuation and returns the value of the weakest pre-expectation. We use the same λ_{HFL}-model as expected cost analyses (Section 3.3). Specifically, we use $[\![\textbf{Prop}]\!] = ([0, \infty], \leq)$ as a set of quantitative truth values.

Example 3.3. Consider the termination probability of the following program [Olmedo et al. 2016].

$$\textbf{let rec } \text{rec3 } x = () \oplus_{1/2} (\text{rec3 } (); \text{rec3 } (); \text{rec3 } ()) \textbf{ in } \text{rec3 } ()$$

By applying the CPS transformation and passing $\lambda y.1$ as a continuation, we obtain the following term in λ_{HFL} whose interpretation is the termination probability $p \in [\![\textbf{Prop}]\!]$.

$$\textbf{let fix } \text{rec3 } x\ k = 1/2 \cdot k\ () + 1/2 \cdot \text{rec3 } ()\ (\lambda y.\text{rec3 } ()\ (\lambda y.\text{rec3 } ()\ k)) \textbf{ in } \text{rec3 } ()\ (\lambda y.1)$$

Here, we assume that we have as basic operations binary arithmetic operators $(+), (\cdot) : \textbf{Prop} \times \textbf{Prop} \rightharpoonup \textbf{Prop}$ and constants $1/2, 1 : \textbf{unit} \rightharpoonup \textbf{Prop}$. □

It is known that $(\sqrt{5} - 1)/2$ gives an upper bound of the termination probability of Example 3.3. Using refinement types, we can specify this as follows.

$$\text{rec3} : (x : \textbf{unit}) \rightarrow (k : \textbf{unit} \rightarrow \{v : \textbf{Prop} \mid r = 1\}) \rightarrow \{v : \textbf{Prop} \mid r \leq (\sqrt{5} - 1)/2\}$$

3.6 Cost Moment

The cost moment analysis [Kura et al. 2019] of a probabilistic program is a generalisation of the expected cost analysis. In expected cost analyses, the aim is to estimate the expected value (i.e. the first moment $\mathbb{E}[C]$) of the cost C of a probabilistic program. On the other hand, the aim of cost moment analyses is to estimate higher moments $\mathbb{E}[C^n]$ of the cost C.

Even when we are interested only in the n-th moment $\mathbb{E}[C^n]$ of the cost, it is convenient to compute all the moments from the first moment to the n-th moment simultaneously ($\mathbb{E}[C], \ldots, \mathbb{E}[C^n]$) as pointed out in [Kura et al. 2019]. Thus, we define a λ_{HFL}-model for cost moment analyses by the category $\omega \textbf{QBS}$ together with the following interpretation of **Prop**.

$$[\![\textbf{Prop}]\!] = ([0, \infty]^n, \leq) \qquad \text{where } \leq \text{ is the component-wise order.}$$

Intuitively, a value $(v_1, \ldots, v_n) \in [\![\mathbf{Prop}]\!]$ represents a tuple of moments of cost: v_1 is the first moment (i.e. the expected cost), and v_2 is the second moment, and so on.

A CPS transformation for cost moment analyses is given by a simple modification to expected cost analyses [Kura 2023]. The tick operator $(-)^{\checkmark}$ is mapped to the *elapse function* $1 \oplus (-) : \mathbf{Prop} \rightharpoonup \mathbf{Prop}$ instead of $1 + (-) : \mathbf{Prop} \rightharpoonup \mathbf{Prop}$. Intuitively, the elapse function $1 \oplus (\mathbb{E}[C], \ldots, \mathbb{E}[C^n])$ computes the binomial expansion of $(\mathbb{E}[1 + C], \ldots, \mathbb{E}[(1 + C)^n])$, which motivates the following definition: the k-th component of $[\![1 \oplus (-)]\!](v_1, \ldots, v_n)$ is defined by $1 + \sum_{i=1}^{k} \binom{k}{i} v_i$

Example 3.4. Consider estimating the second moment of the cost of the coin flipping in Example 1.1. We obtain the following term by the CPS transformation:

$$\mathbf{let\ fix}\ \mathrm{coin}_2\ x\ k = 1/2 \cdot (1 \oplus \mathrm{coin}_2\ ()\ k) + 1/2 \cdot k\ ()\ \mathbf{in}\ \mathrm{coin}_2\ ()\ (\lambda r.(0,0)) \tag{6}$$

where $1/2 \cdot (-) : \mathbf{Prop} \rightharpoonup \mathbf{Prop}$ is the component-wise multiplication. The interpretation of (6) is the pair of the expected cost and the second moment of the cost. The meaning of the elapse function $1 \oplus (-)$ becomes clearer if we (informally) decompose $\mathbf{Prop}$ to $\mathbf{real}_{\geq 0}^{\infty} \times \mathbf{real}_{\geq 0}^{\infty}$ as follows.

$$\mathbf{let\ fix}\ \mathrm{coin}_2\ x\ k = \mathbf{let}\ (r_{11}, r_{12}) = \mathrm{coin}_2\ ()\ k\ \mathbf{in} \quad \mathbf{let}\ (r_{21}, r_{22}) = k\ ()\ \mathbf{in}$$
$$(1/2 \cdot (1 + r_{11}) + 1/2 \cdot r_{21}, \quad 1/2 \cdot (1 + 2 \cdot r_{11} + r_{12}) + 1/2 \cdot r_{22})\ \mathbf{in} \quad \mathrm{coin}_2\ ()\ (\lambda r.(0,0)) \qquad \square$$

To verify that 3 is an upper bound of the second moment of the cost of Example 3.4, we can write the following specification using refinement types.

$$\mathrm{coin}_2 : (x : \mathbf{unit}) \rightarrow (k : \mathbf{unit} \rightarrow \{r : \mathbf{Prop} \mid r = (0,0)\}) \rightarrow \{r : \mathbf{Prop} \mid r \leq (\infty, 3)\}$$

3.7 Conditional Weakest Pre-Expectation

We consider reasoning about higher-order probabilistic programs with conditioning. Conditioning is used in probabilistic programs to model posterior probabilities. Here, we focus on *hard conditioning* $\mathrm{observe}(b)$ for boolean values b (as opposed to *soft conditioning* $\mathrm{score}(r)$ for non-negative real weight r). Intuitively, a probabilistic program with conditioning $\mathrm{observe}(b)$ is interpreted as a sub-probability distribution from which all runs that violate the condition b are excluded. To reason about probabilistic programs with conditioning, the conditional weakest pre-expectation transformer is introduced in [Olmedo et al. 2018] for the probabilistic guarded command language (pGCL) with conditioning. The conditional weakest pre-expectation is defined by the weakest pre-expectation normalised (divided) by the probability of all valid runs.

We can extend conditional weakest pre-expectations to higher-order programs by a CPS transformation as follows. First note that without loss of generality, we can assume that the form of conditioning in a probabilistic program is either $\mathrm{observe}(\mathbf{true})$ or $\mathrm{observe}(\mathbf{false})$ because we can rewrite $\mathrm{observe}(P)$ to $\mathbf{if}\ P\ \mathbf{then}\ \mathrm{observe}(\mathbf{true})\ \mathbf{else}\ \mathrm{observe}(\mathbf{false})$. Then, the CPS transformation for conditional weakest pre-expectation maps $\mathrm{observe}(\mathbf{true})$ and $\mathrm{observe}(\mathbf{false})$ to $1 \cdot (-)$ and $0 \cdot (-)$, respectively. We will show an example in Example 3.5.

As a λ_{HFL}-model, we use $\omega\mathbf{QBS}$ together with the following interpretation of $\mathbf{Prop}$.

$$[\![\mathbf{Prop}]\!] = [0, \infty] \times [0, 1]^{\mathrm{op}} = ([0, \infty], \leq) \times ([0, 1], \geq)$$

The definition of "truth values" above follows [Olmedo et al. 2018]. When computing conditional weakest pre-expectation, it is convenient to separately compute unnormalised weakest pre-expectations for valid runs and the probability of all valid runs (i.e. runs such that b in $\mathrm{observe}(b)$ is always evaluated as true). The first component $[0, \infty]$ represents the unnormalised weakest pre-expectation for valid runs, and the second component $[0, 1]^{\mathrm{op}}$ represents the probability of all valid runs. We use the opposite order for the second component $[0, 1]^{\mathrm{op}}$ because the probability of

all valid runs is defined using greatest fixed points. Thus, the conditional weakest pre-expectation is given by normalisation v_1/v_2 where $(v_1, v_2) \in \llbracket \mathbf{Prop} \rrbracket$.

Example 3.5. Consider the conditional weakest pre-expectation of the following program [Olmedo et al. 2018, Example 4.4], which flips three coins repeatedly.

let $\mathrm{prob}_{1/2}$ $x = \mathbf{true} \oplus_{1/2} \mathbf{false}$ **in**

let rec tc $m = $ **let** $b_1 = \mathrm{prob}_{1/2}$ () **in let** $b_2 = \mathrm{prob}_{1/2}$ () **in let** $b_3 = \mathrm{prob}_{1/2}$ () **in**

$\qquad$ **observe**$(\neg b_1 \vee \neg b_2 \vee \neg b_3)$; $\quad$ **if** $b_1 \vee b_2 \vee b_3$ **then** tc $(m + 1)$ **else** $m + 1$ **in** $\qquad$ tc 0

By applying a CPS transformation and simplifying the result, we obtain the following term of λ_{HFL}. Here, we use $\lambda m. [m = N]$ as a post-expectation where N is an integer and $[m = N] := $ **if** $m = N$ **then** 1 **else** 0 is the Iverson bracket.

$M_{\mathrm{cwp}} \quad := \quad$ **let fix** tc m $k = 1/8 \cdot 0 + 6/8 \cdot$ tc $(m + 1)$ $k + 1/8 \cdot k$ $(m + 1)$ **in** tc 0 $(\lambda m. ([m = N], 1))$

The conditional weakest pre-expectation is obtained as $\llbracket M_{\mathrm{cwp}} \rrbracket_1 / \llbracket M_{\mathrm{cwp}} \rrbracket_2$ where $\llbracket M_{\mathrm{cwp}} \rrbracket_i$ is the i-th component of $\llbracket M_{\mathrm{cwp}} \rrbracket$. $\qquad\qquad\qquad$ □

Using refinement types, we can give the following specification for Example 3.5.

tc : $(m : \mathbf{int}) \rightarrow (k : (m' : \mathbf{int}) \rightarrow \{r : \mathbf{Prop} \mid r = ([m' = N], 1)\})$

$\qquad\qquad \rightarrow \{(r_1, r_2) : \mathbf{Prop} \mid r_1 \leq$ **if** $m \geq N$ **then** 0 **else** $1/6 \cdot (3/4)^{N-m} \wedge r_2 \geq 1/2\}$

Note that this specification implies that the conditional weakest pre-expectation is upper-bounded by $r_1/r_2 \leq 1/3 \cdot (3/4)^{N-m}$.

4 DEPENDENT REFINEMENT TYPE SYSTEM

In this section, we propose a novel refinement type system for λ_{HFL}.

4.1 Formulas

4.1.1 Syntax. Let **AP** be a set of atomic predicates and $\mathrm{ar}(\mathbf{ap})$ be a type, called an *arity*, for each $\mathbf{ap} \in \mathbf{AP}$. We assume that the arity $\mathrm{ar}(\mathbf{ap})$ is a finite product of base types or **Prop**, that is, there exists n and $\sigma_1, \ldots \sigma_n \in \mathbf{Base} \cup \{\mathbf{Prop}\}$ such that $\mathrm{ar}(\mathbf{ap}) = \sigma_1 \times \cdots \times \sigma_n$. We often write $\mathbf{ap} : \sigma$ to indicate that the arity of $\mathbf{ap}$ is σ. We assume that **AP** always contains the order relation $(\leq_{\mathbf{Prop}}) : \mathbf{Prop} \times \mathbf{Prop}$ for **Prop** and the equality relation $(=_b) : b \times b$ for each base type $b \in \mathbf{Base}$.

Formulas are defined by the following syntax.

$$\psi, \phi := \mathbf{ap}(M) \mid \top \mid \bot \mid \psi \implies \phi \mid \psi \wedge \phi \mid \psi \vee \phi \qquad \text{where } \mathbf{ap} \in \mathbf{AP}$$

We define a *well-formed formula* $\Gamma \vdash \phi$ inductively: for atomic predicates, $\Gamma \vdash \mathbf{ap}(M)$ is well-formed if $\Gamma \vdash M : \mathrm{ar}(\mathbf{ap})$ is well-typed, and for other cases, $\Gamma \vdash \phi$ is well-formed if each sub-formula is well-formed. For example, if $\Gamma \vdash M : b$ and $\Gamma \vdash N : b$ are well-typed term, then $\Gamma \vdash (=_b) (M, N)$ is a well-formed formula. For better readability, we often use infix notation and omit type annotations, like $M = N$ instead of $(=_b) (M, N)$ and $M \leq N$ instead of $(\leq_{\mathbf{Prop}}) (M, N)$.

4.1.2 Semantics. Suppose that a λ_{HFL}-model and an interpretation of atomic predicates in **AP** are given: for any $\mathbf{ap} : \sigma$, let $\llbracket \mathbf{ap} \rrbracket \subseteq \llbracket \sigma \rrbracket$ be a subset that gives an interpretation of $\mathbf{ap}$. We assume that we always interpret $\leq_{\mathbf{Prop}}$ by $\llbracket \leq_{\mathbf{Prop}} \rrbracket := \{(x, y) \in \llbracket \mathbf{Prop} \rrbracket \times \llbracket \mathbf{Prop} \rrbracket \mid x \leq_{\llbracket \mathbf{Prop} \rrbracket} y\} \subseteq \llbracket \mathbf{Prop} \times \mathbf{Prop} \rrbracket$ and $=_b$ by $\llbracket =_b \rrbracket := \{(x, x) \in \llbracket b \rrbracket \times \llbracket b \rrbracket \mid x \in \llbracket b \rrbracket\} \subseteq \llbracket b \times b \rrbracket$. A well-formed formula $\Gamma \vdash \phi$ is interpreted as the subset $\llbracket \phi \rrbracket \subseteq \llbracket \Gamma \rrbracket$ defined below. We say $\gamma \in \llbracket \Gamma \rrbracket$ *satisfies* ϕ if $\gamma \in \llbracket \phi \rrbracket$.

$\llbracket \top \rrbracket := \llbracket \Gamma \rrbracket \qquad \llbracket \psi \wedge \phi \rrbracket := \llbracket \psi \rrbracket \cap \llbracket \phi \rrbracket \qquad \llbracket \mathbf{ar}(M) \rrbracket := \{\gamma \in \llbracket \Gamma \rrbracket \mid \llbracket M \rrbracket(\gamma) \in \llbracket \mathbf{ap} \rrbracket\}$

$\llbracket \bot \rrbracket := \emptyset \qquad \llbracket \psi \vee \phi \rrbracket := \llbracket \psi \rrbracket \cup \llbracket \phi \rrbracket \qquad \llbracket \psi \implies \phi \rrbracket := \{\gamma \in \llbracket \Gamma \rrbracket \mid \gamma \in \llbracket \psi \rrbracket \implies \gamma \in \llbracket \phi \rrbracket\}$

4.2 Refinement Types

4.2.1 Syntax. Refinement types $\dot{\sigma}, \dot{\tau}$ *are defined as follows.*

$$\dot{\sigma}, \dot{\tau} \coloneqq \{v : b \mid \phi\} \mid \{v : \mathbf{Prop} \mid \phi\} \mid \{v : 1 \mid \phi\} \mid (x : \dot{\sigma}) \times \dot{\tau} \mid 0 \mid \dot{\sigma} + \dot{\tau} \mid (x : \dot{\sigma}) \rightarrow \{v : \mathbf{Prop} \mid \phi\}$$

The *underlying type* $|\dot{\sigma}|$ of a refinement type $\dot{\sigma}$ is a simple type obtained by erasing formulas. For example, we have $|\{v : \sigma \mid \phi\}| = \sigma$ and $|(x : \dot{\sigma}) \rightarrow \dot{\tau}| = |\dot{\sigma}| \rightarrow |\dot{\tau}|$. The *underlying context* $|\dot{\Gamma}|$ is defined by $|x_1 : \dot{\sigma}_1, \ldots, x_n : \dot{\sigma}_n| \coloneqq x_1 : |\dot{\sigma}_1|, \ldots, x_n : |\dot{\sigma}_n|$. For any $\sigma \in \mathbf{Base} \cup \{1, \mathbf{Prop}\}$, we often write $\sigma \coloneqq \{v : \sigma \mid \top\}$. For any $\sigma_1, \sigma_2 \in \mathbf{Base} \cup \{1, \mathbf{Prop}\}$, we define $\{(v_1, v_2) : \sigma_1 \times \sigma_2 \mid \phi\} \coloneqq (v_1 : \sigma_1) \times \{v_2 : \sigma_2 \mid \phi\}$. For $n > 2$, we define $\{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \phi\}$ in the same way.

Refinement contexts $\dot{\Gamma}$ are defined by a list of pairs of a variable and a refinement type: $\dot{\Gamma} \coloneqq \cdot \mid \dot{\Gamma}, x : \dot{\sigma}$. We assume variables in a context are mutually distinct. We define *well-formed contexts* $\vdash \dot{\Gamma}$ and *well-formed types* $\dot{\Gamma} \vdash \dot{\sigma}$ inductively so that any formula in $\dot{\Gamma}$ and $\dot{\sigma}$ is well-formed (Appendix B). Note that we write a dot on top of each meta-variable for types σ, τ and contexts Γ to distinguish refinement types from simple types in Section 2.

4.2.2 Semantics. Intuitively, a refinement type represents a subset of its underlying type. Taking this into account, we define the interpretation of refinement contexts and refinement types as follows. For each well-formed context $\vdash \dot{\Gamma}$, the interpretation $[\![\dot{\Gamma}]\!]$ is given by a pair (X, P) where $X = [\![|\dot{\Gamma}|]\!]$ is the interpretation of the underlying context of $\dot{\Gamma}$, and $P \subseteq X$ is a subset. Intuitively, P is the set of all $\gamma \in [\![|\dot{\Gamma}|]\!]$ that satisfy all formulas in $\dot{\Gamma}$. For each well-formed type $\dot{\Gamma} \vdash \dot{\sigma}$, the interpretation $[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$ is given by a quadruple (X, Y, P, Q) where $(X, P) = [\![\dot{\Gamma}]\!]$ is the interpretation of $\dot{\Gamma}$, $Y = [\![|\dot{\sigma}|]\!]$ is the interpretation of the underlying type of $\dot{\sigma}$, and $Q \subseteq X \times Y$ is a subset such that $(x, y) \in Q$ implies $x \in P$. Examples of the interpretation are given below.

Let $\quad \dot{\Gamma}_1 \coloneqq x : \{x : \mathbf{int} \mid 0 \leq x\} \quad$ and $\quad \dot{\Gamma}_2 \coloneqq x : \{x : \mathbf{int} \mid 0 \leq x\}, y : \{y : \mathbf{int} \mid x \leq y\}$.

$$
\begin{aligned}
[\![\dot{\Gamma}_1]\!] \quad &= \quad (\mathbb{Z}, \quad \{x \in \mathbb{Z} \mid 0 \leq x\}) \\
[\![\dot{\Gamma}_1 \vdash \{y : \mathbf{int} \mid x \leq y\}]\!] \quad &= \quad (\mathbb{Z}, \quad \mathbb{Z}, \quad \{x \in \mathbb{Z} \mid 0 \leq x\}, \quad \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid 0 \leq x \wedge x \leq y\}) \\
[\![\dot{\Gamma}_2]\!] \quad &= \quad (\mathbb{Z} \times \mathbb{Z}, \quad \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid 0 \leq x \wedge x \leq y\})
\end{aligned}
$$

We often write $[\![-]\!]_i$ for the i-th component of $[\![-]\!]$. Technically, this interpretation is based on a categorical construction of models of dependent refinement type systems [Kura 2021] (see Appendix B for details). Note that if $(X, Y, P, Q) = [\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$, then we have $[\![\dot{\Gamma}, x : \dot{\sigma}]\!] = (X \times Y, Q)$.

Using our denotational semantics, a (semantic) subtyping relation can be defined as follows. A type $\dot{\Gamma} \vdash \dot{\sigma}$ is a *semantic subtype* of $\dot{\Gamma} \vdash \dot{\tau}$ if $[\![\dot{\Gamma}, x : \dot{\sigma}]\!]_1 = [\![\dot{\Gamma}, x : \dot{\tau}]\!]_1$ and $[\![\dot{\Gamma}, x : \dot{\sigma}]\!]_2 \subseteq [\![\dot{\Gamma}, x : \dot{\tau}]\!]_2$.

4.3 Typing Rules

Fig. 4 shows typing rules for dependent refinement types. Here, a judgement for well-typed-ness has the form $\dot{\Gamma} \vdash M : \dot{\sigma}$. We can use ordinary typing rules for dependent refinement types in most cases except for R-Fix and R-BasicOp. R-Fix and R-BasicOp contain semantic conditions in their premises, which will be explained in Section 4.3.2 and Section 4.3.3, respectively. In R-Sub, we use a subtyping relation $\dot{\Gamma} \vdash \dot{\sigma} <: \dot{\tau}$. Since derivation rules for the subtyping relation are standard, they are omitted here (see Appendix B for details). The soundness of the typing rules is stated as follows.

THEOREM 4.1 (SOUNDNESS). *For any* λ_{HFL}*-model, typing rules in Fig. 4 are sound: if* $\dot{\Gamma} \vdash M : \dot{\sigma}$ *is well-typed, then for any* $\gamma \in [\![\dot{\Gamma}]\!]_2$*, we have* $(\gamma, [\![M]\!](\gamma)) \in [\![\dot{\Gamma}, r : \dot{\sigma}]\!]_2$ *where* $[\![\dot{\Gamma}]\!]_2$ *is the second component of* $[\![\dot{\Gamma}]\!]$*.*

PROOF. By induction of derivation of $\dot{\Gamma} \vdash M : \dot{\sigma}$. We use a general categorical construction of [Kura 2021]. For the case of R-Fix, see Lemma 4.10. Other cases are proved in Appendix C. □

R-App
$$\frac{\dot{\Gamma} \vdash M : (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\} \qquad \dot{\Gamma} \vdash N : \dot{\sigma}}{\dot{\Gamma} \vdash M\,N : \{v : \mathbf{Prop} \mid \phi[N/x]\}}$$

R-Abs
$$\frac{\dot{\Gamma}, x : \dot{\sigma} \vdash M : \{v : \mathbf{Prop} \mid \phi\}}{\dot{\Gamma} \vdash \lambda x : \dot{\sigma}.M : (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}}$$

R-Unit
$$\frac{\vdash \dot{\Gamma}}{\dot{\Gamma} \vdash () : \{v : 1 \mid \top\}}$$

R-Pair
$$\frac{\dot{\Gamma} \vdash M : \dot{\sigma} \qquad \dot{\Gamma} \vdash N : \dot{\tau}[M/x]}{\dot{\Gamma} \vdash (M, N) : (x : \dot{\sigma}) \times \dot{\tau}}$$

R-Fst
$$\frac{\dot{\Gamma} \vdash M : (x : \dot{\sigma}) \times \dot{\tau}}{\dot{\Gamma} \vdash \pi_1\,M : \dot{\sigma}}$$

R-Snd
$$\frac{\dot{\Gamma} \vdash M : (x : \dot{\sigma}) \times \dot{\tau}}{\dot{\Gamma} \vdash \pi_2\,M : \dot{\tau}[\pi_1\,M/x]}$$

R-Case0
$$\frac{\dot{\Gamma} \vdash M : 0 \qquad \dot{\Gamma} \vdash \dot{\tau}}{\dot{\Gamma} \vdash \delta(M) : \dot{\tau}}$$

R-Inj
$$\frac{i \in \{1, 2\} \qquad \dot{\Gamma} \vdash M : \dot{\sigma}_i \qquad \dot{\Gamma} \vdash \dot{\sigma}_{3-i}}{\dot{\Gamma} \vdash \iota_i\,M : \dot{\sigma}_1 + \dot{\sigma}_2}$$

R-Case2
$$\frac{\dot{\Gamma}, z : \dot{\sigma}_1 + \dot{\sigma}_2 \vdash \dot{\tau} \qquad \dot{\Gamma} \vdash M : \dot{\sigma}_1 + \dot{\sigma}_2 \qquad \dot{\Gamma}, x_1 : \dot{\sigma}_1 \vdash N_1 : \dot{\tau}[\iota_1\,x_1/z] \qquad \dot{\Gamma}, x_2 : \dot{\sigma}_2 \vdash N_2 : \dot{\tau}[\iota_2\,x_2/z]}{\dot{\Gamma} \vdash \delta(M, x_1.N_1, x_2.N_2) : \dot{\tau}[M/z]}$$

R-VarRefine
$$\frac{\vdash \dot{\Gamma} \qquad (x : \{v : b \mid \phi\}) \in \dot{\Gamma}}{\dot{\Gamma} \vdash x : \{v : b \mid v = x\}}$$

R-Var
$$\frac{\vdash \dot{\Gamma} \qquad (x : \dot{\sigma}) \in \dot{\Gamma}}{\dot{\Gamma} \vdash x : \dot{\sigma}}$$

R-Sub
$$\frac{\dot{\Gamma} \vdash M : \dot{\sigma} \qquad \dot{\Gamma} \vdash \dot{\sigma} <: \dot{\tau}}{\dot{\Gamma} \vdash M : \dot{\tau}}$$

R-Fix
$$\frac{\dot{\Gamma}, f : (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\} \vdash M : (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\} \qquad |\dot{\Gamma}|, x : |\dot{\sigma}|, v : \mathbf{Prop} \vdash \phi \qquad \phi \text{ is admissible at } v}{\dot{\Gamma} \vdash \mathbf{fix}\,f.M : (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}}$$

R-BasicOp
$$\frac{\dot{\Gamma} \vdash \dot{\tau} \qquad \mathrm{ar}(\mathbf{op}) = |\dot{\sigma}| \qquad \mathrm{car}(\mathbf{op}) = |\dot{\tau}| \qquad \dot{\Gamma} \vdash M : \dot{\sigma} \qquad \forall \gamma \in [\![\dot{\Gamma}, v : \dot{\sigma}]\!]_2, \gamma[v \mapsto a(\mathbf{op})(\gamma(v))] \in [\![\dot{\Gamma}, v : \dot{\tau}]\!]_2}{\dot{\Gamma} \vdash \mathbf{op}(M) : \dot{\tau}}$$

Fig. 4. Typing rules for $\dot{\Gamma} \vdash M : \dot{\sigma}$.

The following corollary states the soundness in a more intuitive way.

COROLLARY 4.2. *For any λ_{HFL}-model, if $\vdash M : \{v : \sigma \mid \phi\}$ is well-typed, then $\vdash \phi[M/v]$ is true.* □

Since Corollary 4.2 hold for any λ_{HFL}-model, we obtain soundness for each instance in Section 3.

COROLLARY 4.3 (SOUNDNESS OF WEAKEST PRE-EXPECTATIONS AND EXPECTED COSTS). *Consider the λ_{HFL}-model for weakest pre-expectations and expected costs. For any $u \in [\![\mathbf{Prop}]\!] = ([0, \infty], \leq)$,*

$$\text{if} \quad \vdash M : \{v : \mathbf{Prop} \mid v \leq u\} \quad \text{is well-typed, then} \quad [\![M]\!](\emptyset) \leq u. \qquad \square$$

COROLLARY 4.4 (SOUNDNESS OF COST MOMENT ANALYSES). *Consider the λ_{HFL}-model for cost moment analyses. For any $(u_1, \ldots, u_n) \in [\![\mathbf{Prop}]\!] = ([0, \infty]^n, \leq)$,*

$$\text{if} \quad \vdash M : \{v : \mathbf{Prop} \mid v \leq (u_1, \ldots, u_n)\} \quad \text{is well-typed, then} \quad [\![M]\!](\emptyset) \leq (u_1, \ldots, u_n). \qquad \square$$

COROLLARY 4.5 (SOUNDNESS OF THE CONDITIONAL WEAKEST PRE-EXPECTATIONS). *Consider the λ_{HFL}-model for conditional weakest pre-expectations. For any $(u_1, u_2) \in [0, \infty] \times [0, 1]^{\mathrm{op}}$,*

$$\text{if} \quad \vdash M : \{v : \mathbf{Prop} \mid v \leq_{\mathbf{Prop}} (u_1, u_2)\} \quad \text{is well-typed, then} \quad [\![M]\!]_1(\emptyset) \leq u_1 \wedge [\![M]\!]_2(\emptyset) \geq u_2$$

where $[\![M]\!]_i(\emptyset)$ is the i-th component of $[\![M]\!](\emptyset) \in [\![\mathbf{Prop}]\!] = [0, \infty] \times [0, 1]^{\mathrm{op}}$. □

FIX-UNSOUND
$$\frac{\Gamma, f : (x : \sigma) \to \{v : \mathbf{real}_{\geq 0}^{\infty} \mid \phi\} \vdash M : (x : \sigma) \to \{v : \mathbf{real}_{\geq 0}^{\infty} \mid \phi\}}{\Gamma \vdash \mathbf{fix}\ f.M : (x : \sigma) \to \{v : \mathbf{real}_{\geq 0}^{\infty} \mid \phi\}}$$

Fig. 5. The standard typing rule for (the partial correctness) of recursion is unsound for fixed points.

4.3.1 Unsoundness of the Standard Rule for Recursion. The standard typing rule for recursion (FIX-UNSOUND in Fig. 5) is unsound for fixed points. To give a counterexample, consider the term (1) in Example 1.1. We consider the following type where $\phi(r)$ is a predicate on the expected cost r.

$$\text{coin} : (x : \mathbf{unit}) \to (k : \mathbf{unit} \to \{r : \mathbf{real}_{\geq 0}^{\infty} \mid r = 0\}) \to \{r : \mathbf{real}_{\geq 0}^{\infty} \mid \phi(r)\} \tag{7}$$

If we use FIX-UNSOUND, we can derive $\phi(r) = r < 1$ and $\phi(r) = \perp$ since $\phi(r)$ implies $\phi(1/2 \cdot (1 + r) + 1/2 \cdot 0)$ in both cases. However, both of them are incorrect because the true expected cost $r = 1$ doesn't satisfy neither $r < 1$ nor $\perp$.

Technically, the unsoundness of FIX-UNSOUND is because the semantics of **fix** is different from usual semantics of recursion: **fix** is defined by the least fixed point with respect to the standard order on $[0, \infty]$ whereas usual recursion is the least fixed point with respect to $X_{\perp} = X \cup \{\perp\}$ for some X where $\perp$ represents divergence and the partial order on $X_{\perp}$ is defined by (1) $\perp$ is the least element and (2) restriction of the partial order on X is the discrete order. Therefore, the typing rule for the latter semantics is unsound for the former.

4.3.2 A Sound Typing Rule for Fixed Points. To fix the issue of FIX-UNSOUND, we take into account *admissible subsets* (*admissible predicates*) of ωcpos and consider the typing rule R-FIX. This approach is an adaptation of EHOL [Avanzini et al. 2021] to a more automatable method, namely, a dependent refinement type system. Furthermore, we prove that R-FIX is sound for any λ_{HFL}-models. Thus, our dependent refinement type system is applicable to all verification problems listed in Section 3.

Admissible subsets have a closure property with respect to least fixed points. Recall that the least fixed point lfp f in ωcpos is given by the supremum of an ω-chain: assuming that an ωcpo X has a least element $\perp_X \in X$, we have lfp $f = \sup_n f^n(\perp_X)$ for any Scott-continuous function $f : X \to X$. Suppose that $S \subseteq X$ is a subset such that $f : X \to X$ maps elements in S to elements in S. Even in this situation, lfp $f \in S$ is not necessarily true in general. To ensure lfp $f \in S$, we need the notion of admissible subsets.

Definition 4.6 (admissible subset). Let X be an ωcpo with a least element $\perp_X \in X$. A subset $S \subseteq X$ is *admissible* if (i) $\perp_X \in S$ and (ii) S is closed under supremum of ω-chains.

Example 4.7. Consider the ωcpo $([0, \infty], \leq)$. For any $a \in [0, \infty]$, $\{x \in [0, \infty] \mid x \leq a\} \subseteq [0, \infty]$ is admissible. On the other hand, $\{x \in [0, \infty] \mid x < a\} \subseteq [0, \infty]$ is *not* admissible because this subset is not closed under supremum of ω-chains. For any $a > 0$, $\{x \in [0, \infty] \mid a \leq x\} \subseteq [0, \infty]$ is *not* admissible because the least element $0 \in [0, \infty]$ is not in this subset. □

Example 4.7 suggests why $\phi(r) = r < 1$ should *not* be derived for Example 1.1 and (7). Recall that the interpretation of the fixed point in (1) is given by the supremum of the ω-chain $0 \leq 1/2 \leq 3/4 \leq \cdots \to 1$. Although each element of the ω-chain satisfies $\phi(r) = r < 1$, their supremum doesn't because $\phi(r) = r < 1$ is not admissible.

Example 4.8. Consider the ωcpo $X_{\perp} = (X \cup \{\perp\}, \leq)$ where we have $x \leq y$ if and only if $x = \perp$. For any subset $S \subseteq X$, $S \cup \{\perp\} \subseteq X_{\perp}$ is admissible. □

R-BasicConst
$$\dfrac{\mathbf{op} : 1 \to \tau \qquad \tau \in \mathbf{Base} \cup \{\mathbf{Prop}\}}{\dot{\Gamma} \vdash \mathbf{op} : \{v : \tau \mid v =_\tau \mathbf{op}\}}$$

R-BasicSimp
$$\dfrac{\mathbf{op} : \sigma_1 \times \cdots \times \sigma_n \to \tau \qquad \sigma_1, \ldots, \sigma_n, \tau \in \mathbf{Base} \cup \{\mathbf{Prop}\} \qquad \dot{\Gamma} \vdash M : \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \phi[\mathbf{op}(v_1, \ldots, v_n)/v]\}}{\dot{\Gamma} \vdash \mathbf{op}(M) : \{v : \tau \mid \phi\}}$$

R-BasicBool
$$\dfrac{\begin{array}{c} \mathbf{op} : \sigma_1 \times \cdots \times \sigma_n \to 1 + 1 \qquad \sigma_1, \ldots, \sigma_n \in \mathbf{Base} \cup \{\mathbf{Prop}\} \\ v_1 : \sigma_1, \ldots, v_n : \sigma_n \vdash \psi_t \qquad \forall \gamma \in [\![(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \psi_t]\!], a(\mathbf{op})(\gamma) = \iota_1 \, () \\ v_1 : \sigma_1, \ldots, v_n : \sigma_n \vdash \psi_f \qquad \forall \gamma \in [\![(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \psi_f]\!], a(\mathbf{op})(\gamma) = \iota_2 \, () \\ \dot{\Gamma} \vdash M : \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \psi_t \wedge \phi_t[()/v] \vee \psi_f \wedge \phi_f[()/v]\} \end{array}}{\dot{\Gamma} \vdash \mathbf{op}(M) : \{v : 1 \mid \phi_t\} + \{v : 1 \mid \phi_f\}}$$

Fig. 6. Three common patterns of typing rules for basic operators.

Example 4.8 is why the standard rule for partial correctness (Fix-Unsound) of recursion does not impose admissibility explicitly. The semantics of recursion is usually defined using the ωcpo $X_\perp$, in which case, the admissibility of predicates is guaranteed almost for free.

Recall that a formula $\Gamma \vdash \phi$ is interpreted as a subset of $[\![\Gamma]\!] = \prod_i [\![\sigma_i]\!]$. For each variable $(x_i : \sigma_i) \in \Gamma$, $[\![\phi]\!] \subseteq [\![\Gamma]\!]$ induces a subset of $[\![\sigma_i]\!]$ by fixing values assigned to other variables in Γ. We define the admissibility of formulas by considering the admissibility of such subsets.

Definition 4.9 (admissible formula). A well-formed formula $\Gamma \vdash \phi$ is *admissible at variable x* if $(x : \mathbf{Prop}) \in \Gamma$ and for any $\gamma \in [\![\Gamma]\!]$, $\{v \in [\![\mathbf{Prop}]\!] \mid \gamma[x \mapsto v] \in [\![\Gamma]\!]\} \subseteq [\![\mathbf{Prop}]\!]$ is admissible. Here, $\gamma[x \mapsto v]$ is the mapping such that $\gamma[x \mapsto v](x) = v$ and $\gamma[x \mapsto v](y) = \gamma(y)$ if $x \neq y$.

LEMMA 4.10. *R-Fix is sound in the following sense. Let $\dot{\tau} = (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}$. If M satisfies $(\gamma, [\![M]\!](\gamma)) \in [\![\dot{\Gamma}, f : \dot{\tau}, f' : \dot{\tau}]\!]_2$ for any $\gamma \in [\![\dot{\Gamma}, f : \dot{\tau}]\!]_2$ and $|\dot{\Gamma}|, x : |\dot{\sigma}|, v : \mathbf{Prop} \vdash \phi$ is admissible at v, then we have $(\gamma, [\![\mathbf{fix}\ f.M]\!](\gamma)) \in [\![\dot{\Gamma}, f : \dot{\tau}]\!]_2$ for any $\gamma \in [\![\dot{\Gamma}]\!]_2$.*

PROOF. This is a part of the induction in Theorem 4.1. We sketch the proof (see Appendix C for details). Since $|\dot{\Gamma}|, x : |\dot{\sigma}|, v : \mathbf{Prop} \vdash \phi$ is admissible at v, we can prove that the forth component of $[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}]\!]$ is also "admissible", which precisely means that for any $\gamma \in [\![\dot{\Gamma}]\!]_2$,

$$\{v : [\![|\dot{\sigma}|]\!] \to [\![\mathbf{Prop}]\!] \mid \gamma[f \mapsto v] \in [\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}]\!]_4\} \subseteq [\![|\dot{\sigma}|]\!] \Rightarrow [\![\mathbf{Prop}]\!]$$

is admissible where $[\![-]\!]_i$ is the i-th component of $[\![-]\!]$. By definition of the interpretation, we have $[\![\mathbf{fix}\ f.M]\!](\gamma) = \sup_n F^n(\perp)$ where $F := [\![M]\!](\gamma[f \mapsto (-)])$. By the induction hypothesis and by the admissibility of $[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}]\!]$, we conclude $(\gamma, [\![\mathbf{fix}\ f.M]\!](\gamma)) \in [\![\dot{\Gamma}, f : (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}]\!]_2$ for any $\gamma \in [\![\dot{\Gamma}]\!]_2$. □

To implement R-Fix in a type checker, we need to check the admissibility of formulas *syntactically*, which will be discussed later in Section 6.2.

4.3.3 Typing Rules for Basic Operators. R-BasicOp gives a general typing rule for basic operators. One of its premises is a semantic condition $\forall \gamma \in [\![\dot{\Gamma}, v : \dot{\sigma}]\!]_2, \gamma[v \mapsto a(\mathbf{op})(\gamma(v))] \in [\![\dot{\Gamma}, v : \dot{\tau}]\!]_2$, which intuitively means that if an argument of $a(\mathbf{op})$ satisfies the precondition in $\dot{\sigma}$, then $a(\mathbf{op})$ returns a value that satisfies the postcondition in $\dot{\tau}$. Although R-BasicOp is a natural and general rule, it is usually convenient to specialise it so that we can avoid the semantic condition in an implementation of a type checker. Fortunately, rather common typing rules shown in Fig. 6 can be proved sound in any λ_{HFL}-model, and they cover most of basic operators that we need for verifying probabilistic programs.

R-LeqInt
$$\dot{\Gamma} \vdash (M, N) : \{(v_1, v_2) : \textbf{int} \times \textbf{int} \mid v_1 \leq v_2 \land \phi_t[()/v] \lor (v_1 \leq v_2 \implies \bot) \land \phi_f[()/v]\}$$
$$\overline{\dot{\Gamma} \vdash M \leq_{\textbf{int}} N : \{v : 1 \mid \phi_t\} + \{v : 1 \mid \phi_f\}}$$

R-Unif
$$\frac{|\dot{\Gamma}| \vdash N : \textbf{real} \to \textbf{Prop} \qquad \dot{\Gamma} \vdash M : (x : \{x : \textbf{real} \mid 0 \leq x \land x \leq 1\}) \to \{v : \textbf{Prop} \mid v \leq N\,x\}}{\dot{\Gamma} \vdash \textbf{unif}(M) : \{v : \textbf{Prop} \mid v \leq \textbf{unif}(N)\}}$$

Fig. 7. Specialised typing rules for basic operators.

In Fig. 6, the first rule R-BasicConst is a specialised rule for constants $\textbf{op} : 1 \rightharpoonup \tau$ in **BO** where $\tau \in \textbf{Base} \cup \{\textbf{Prop}\}$. The second rule R-BasicSimp gives a typing rule for basic operators whose arity and coarity are given by $\textbf{op} : \sigma_1 \times \cdots \times \sigma_n \rightharpoonup \tau$ where $\sigma_1, \ldots, \sigma_n, \tau \in \textbf{Base} \cup \{\textbf{Prop}\}$. Note that many binary arithmetic operators conform to this pattern. The last rule R-BasicBool is a specialised rule for basic operators that return boolean values of type $1 + 1$. R-BasicBool still contains semantic conditions, but they are much easier to handle than the general rule R-BasicOp. The semantic conditions for R-BasicBool means that ψ_t and ψ_f are conditions on $\text{ar}(\textbf{op})$ under which $\textbf{op}$ returns **true** and **false**, respectively. For example, R-LeqInt in Fig. 7 is a more specialised rule for a basic operator $((\leq_{\textbf{int}}) : \textbf{int} \times \textbf{int} \rightharpoonup 1 + 1) \in \textbf{BO}$, in which we use $\psi_t(v_1, v_2) \coloneqq v_1 \leq v_2$ and $\psi_f(v_1, v_2) \coloneqq v_1 \leq v_2 \implies \bot$, assuming that we have a corresponding atomic predicate $(\leq_{\textbf{int}}) : \textbf{int} \times \textbf{int}$ in **AP**.

PROPOSITION 4.11. *Typing rules in Fig. 6 are sound in any λ_{HFL}-model.* ☐

A notable exception to three common patterns in Fig. 6 is the integration operator for a continuous distribution. For example, to reason about uniform distribution over the unit interval $[0, 1]$, we consider an integration operator $\textbf{unif} : (\textbf{real} \to \textbf{Prop}) \rightharpoonup \textbf{Prop}$ defined in (3) whose semantics is given by $f \mapsto \int_0^1 f(x)\,dx$. Since the arity of $\textbf{unif}$ is a function type $\textbf{real} \to \textbf{Prop}$, we cannot apply any of Fig. 6. In such a situation, we need to design a typing rule on a case-by-case basis. The general rule R-BasicOp remains sound for all basic operators including $\textbf{unif}$, but R-BasicOp is not very convenient in practice. We provide a typing rule R-Unif for $\textbf{unif}$ in Fig. 7. R-Unif asserts: if a term N of type $\textbf{real} \to \textbf{Prop}$ is an upper bound of M, then $\textbf{unif}(M)$ is upper bounded by $\textbf{unif}(N)$. This is sound by the monotonicity of integration. Here, note that we don't have to restrict ourselves to the uniform distribution. It is easy to consider similar typing rules for other probability distributions and to prove soundness.

PROPOSITION 4.12. *R-Unif is sound for the λ_{HFL}-model for weakest pre-expectations, expected costs, cost moments, and conditional weakest pre-expectations.*

PROOF. By the monotonicity of $[\![\textbf{unif}]\!]$. See Appendix C for details. ☐

5 TYPE CHECKING ALGORITHM

As we have seen in Section 3, many verification problems for probabilistic programs can be expressed as type-checking problems for our dependent refinement type system. Given a term M of λ_{HFL}, a refinement context $\dot{\Gamma}$, and a refinement type $\dot{\sigma}$ for the term, the *type-checking problem* is the problem of deciding whether $\dot{\Gamma} \vdash M : \dot{\sigma}$ is well-typed.

In this section, we explain a reduction from type-checking problems to constraint solving, which is based on the standard type checking algorithm [Rondon et al. 2008; Unno and Kobayashi 2009] for refinement type systems. The main difference from the standard one is that we consider Constrained

Horn Clauses (CHC) constraints extended with two new types of predicate variables: one is for admissible predicates, and the other is for integration operators. We call this **CHC**[adm, $\int$]. We generate a set of constraints whose satisfiability implies the well-typedness of a given term $\dot{\Gamma} \vdash M : \dot{\sigma}$. Note that specifications by refinement types are only required for top-level declarations because our type checker can synthesise inductive invariants automatically. Implementing a constraint solver for the extended CHC will be explained later in Section 6.

5.1 Reduction to CHC Constraints

The workflow of type checking is given as follows. Given a term of λ_{HFL}, we first apply the Hindley–Milner type inference algorithm [Damas and Milner 1982] to obtain simple types for each sub-terms of M. Then, we generate templates of refinement types by replacing each occurrence of $\sigma \in$ **Base**$\cup\{1, \textbf{Prop}\}$ in simple types with the refinement type $\{x : \sigma \mid P(\tilde{y})\}$ where P is a (fresh) predicate variable and $\tilde{y}$ is a list of variables visible from the current scope. We generate **CHC**[adm, $\int$]-constraints on predicate variables by applying typing rules of the dependent refinement type system. Those **CHC**[adm, $\int$]-constraints have three types of predicate variables: ordinary predicate variables, *admissible predicate variables*, and *integrable predicate variables*. We use ordinary predicate variables for most of the typing rules, just like the standard type-checking algorithm [Rondon et al. 2008; Unno and Kobayashi 2009]. Admissible predicate variables and integrable predicate variables are used for R-Fix and R-Unif. Once we obtain CHC constraints, we solve them using a CHC solver extended for **CHC**[adm, $\int$]. If a solution is found, then $\vdash M : \dot{\sigma}$ is well-typed. This reduction to **CHC**[adm, $\int$]-constraints generates a set of CHC constraints whose size grows linearly with respect to the size of a given program. [2]

We explain more about the new types of predicate variables. A predicate variable in CHC is denoted by $P(\tilde{x})$ where $\tilde{x}$ is a set of variables on which P depends. An *admissible predicate variable* $P(v; \tilde{x})$ is defined as a predicate variable $P(v, \tilde{x})$ that must be instantiated by a predicate $\phi(y, \tilde{x})$ (i.e. a formula with free variables $y, \tilde{x}$) that is admissible at y. Since R-Fix requires the admissibility of the predicate on the codomain of a fixed point **fix** $f.M : \sigma \rightarrow \textbf{Prop}$, we use an admissible predicate variable when generating constraints using R-Fix. An *integrable predicate variable* $P(v; y; \tilde{x})$ is a predicate variable $P(v, y, \tilde{x})$ that must be instantiated by a formula of the form $v \leq N y$ (recall R-Unif) where free variables in N must be in $\tilde{x}$. For each integrable predicate variable $P(v; y; \tilde{x})$, we have an associated predicate variable **Integ**$_{\textbf{unif}}(P)(v; \tilde{x})$. Whenever $P(v; y; \tilde{x})$ is instantiated to $v \leq N y$, **Integ**$_{\textbf{unif}}(P)(v; \tilde{x})$ is instantiated to $v \leq \textbf{unif}(N)$ at the same time. We use an integrable predicate variable when generating CHC constraints using R-Unif.

Example 5.1. We explain the CHC constraint generation using a simple example. Consider type-checking the following term.[3]

$$(\lambda x.x + 1)\ 42 \quad : \quad \{y : \textbf{int} \mid y \geq 0\} \tag{8}$$

First, we apply the Hindley–Milner type inference, which yields the simple type for each subtem of (8). For example, $(\lambda x.x + 1)\ 42$ has type **int** and $\lambda x.x + 1$ has type **int** $\rightarrow$ **int**. Then, we use a procedure **ConstGen** for constraint generation, which takes a refinement context $\dot{\Gamma}$, a term M, and a type annotation $\dot{\sigma}$; and returns CHC constraints. The procedure **ConstGen** recursively

---

[2]Our current implementation of the constraint generator performs a preprocessing step that eliminates as many redundant predicate variables as possible to assist the backend CHC solver. Although the original CHC constraints are of linear size, this preprocessing can cause an exponential blowup of CHC constraints in the worst case. This issue is left for future work because the experiments (Section 6) showed that we can still solve many benchmarks.

[3]This term has a subterm of type **int** $\rightarrow$ **int**, which is, strictly speaking, not allowed in our refinement type system. However, we are allowing such function types only in this example for illustrative purposes.

R-Fix-ConstGen

$$\tau \to \textbf{Prop} = \text{SType}(\textbf{fix } f.M) \qquad \text{Let } \dot{\tau} \text{ be a refinement-type template obtained from } \tau.$$

Let P^{adm} be a fresh *admissible* predicate variable.

$$\dot{\Gamma}, f : (x : \dot{\tau}) \to \{v : \textbf{Prop} \mid P^{\text{adm}}(v; \text{vars}(\dot{\Gamma}), x)\} \vdash M : (x : \dot{\tau}) \to \{v : \textbf{Prop} \mid P^{\text{adm}}(v; \text{vars}(\dot{\Gamma}), x)\}$$

$$\dfrac{\Gamma \vdash (x : \dot{\tau}) \to \{v : \textbf{Prop} \mid P^{\text{adm}}(v; \text{vars}(\dot{\Gamma}), x)\} <: (x : \dot{\sigma}) \to \{v : \textbf{Prop} \mid \phi\}}{\dot{\Gamma} \vdash \textbf{fix } f.M : (x : \dot{\sigma}) \to \{v : \textbf{Prop} \mid \phi\}}$$

R-Unif-ConstGen

Let P^{int} be a fresh integration predicate variable.

$$\dot{\Gamma} \vdash M : (x : \{x : \textbf{real} \mid 0 \le x \land x \le 1\}) \to \{v : \textbf{Prop} \mid P^{\text{int}}(v; x; \text{vars}(\dot{\Gamma}))\}$$

$$\dfrac{\dot{\Gamma} \vdash \{v : \textbf{Prop} \mid \textbf{Integ}_{\textbf{unif}}(P^{\text{int}})(v; \text{vars}(\dot{\Gamma}))\} <: \{v : \textbf{Prop} \mid \phi\}}{\dot{\Gamma} \vdash \textbf{unif}(M) : \{v : \textbf{Prop} \mid \phi\}}$$

Fig. 8. Selected rules for constraint generation. We assume that the arguments of predicate variables are of type **int**, **real**, or **Prop**, and that variables that are not typed by those types are implicitly removed from the arguments of predicate variables.

applies typing rules for our refinement type system. In this example, we invoke **ConstGen** with the following arguments.

$$\textbf{ConstGen}\big(\cdot, \quad (\lambda x.x + 1)\ 42, \quad \{y : \textbf{int} \mid y \ge 0\}\big) \qquad \text{where } \cdot \text{ is the empty (refinement) context.}$$

To compute this, R-App should be applied first. Since **ConstGen** needs to guess a refinement type of $\lambda x.x + 1$, **ConstGen** generates fresh predicate variables and builds a refinement-type template $(x : \{x : \textbf{int} \mid P_1(x)\}) \to \{y : \textbf{int} \mid P_2(x, y)\}$. For the function application to be well-typed, the following constraint (9) should be satisfied where **SubType**$(\dot{\Gamma}, \dot{\sigma}, \dot{\tau})$ is a procedure for generating constraints for the subtyping relation $\dot{\Gamma} \vdash \dot{\sigma} <: \dot{\tau}$.

$$\begin{aligned} &\textbf{ConstGen}\big(\cdot, \lambda x.x + 1, (x : \{x : \textbf{int} \mid P_1(x)\}) \to \{y : \textbf{int} \mid P_2(x, y)\}\big) \\ &\cup \textbf{ConstGen}\big(\cdot, 42, \{x : \textbf{int} \mid P_1(x)\}\big) \cup \textbf{SubType}\big(\cdot, \{y : \textbf{int} \mid P_2(42, y)\}, \{y : \textbf{int} \mid y \ge 0\}\big) \end{aligned} \tag{9}$$

In other words, **ConstGen** computes constraints using R-App-ConstGen below.

R-App-ConstGen

Let $\dot{\sigma} \to \dot{\tau}_1$ be a refinement-type template obtained from the (simple) type of M.

$$\dfrac{\dot{\Gamma} \vdash M : \dot{\sigma} \to \dot{\tau}_1 \qquad \dot{\Gamma} \vdash N : \dot{\sigma} \qquad \dot{\Gamma} \vdash \dot{\tau}_1[N/x] <: \dot{\tau}_2}{\dot{\Gamma} \vdash M\ N : \dot{\tau}_2}$$

The rest of the constraint generation proceeds as follows. The first component of (9) is computed by applying R-Abs and then applying R-BasicSimp for $+ : \textbf{int} \times \textbf{int} \to \textbf{int}$, which yields $P_1(x) \implies P_2(x, x + 1)$. By R-BasicConst, the second component of (9) is $x = 42 \implies P_1(x)$. By the definition of subtyping relation, the third component of (9) is $P_2(42, y) \implies y \ge 0$. As a result, we get the following CHC constraints.

$$P_1(x) \implies P_2(x, x + 1) \qquad x = 42 \implies P_1(x) \qquad P_2(42, y) \implies y \ge 0$$

Since the CHC constraints are satisfiable (for example, let $P_1(x) = (x = 42)$ and $P_2(x, y) = (y \ge 0)$), we conclude that (8) is well-typed. □

Example 5.2. We explain how constraints are generated for R-Fix and R-Unif. Consider type-checking the following term, which is taken from the random walk example in (4).

$$\textbf{fix } \text{rw}.(\lambda\, x\, k.\textbf{if } x \geq 0 \textbf{ then unif } (\lambda y.1 + \text{rw } (x + 3 \cdot y - 2)\, k) \textbf{ else } k\, ()) \tag{10}$$

The type annotation is given as follows, which is essentially the same as (5).

$$\text{rw} : (x : \{x : \textbf{real} \mid x \geq -2\}) \to (k : (u : \textbf{unit}) \to \{r : \textbf{Prop} \mid r = 0\}) \to \{r : \textbf{Prop} \mid r \leq |2 \cdot x + 4|\}$$

Similarly to Example 5.1, we apply the Hindley–Milner type inference to infer simple types and then apply the procedure **ConstGen** for constraint generation to (10). Let's take a closer look at the constraint generation. **ConstGen** first applies R-Fix-ConstGen in Fig. 8 to handle the fixed point. R-Fix-ConstGen introduces the refinement-type template (11) for rw where $P_{\text{rw}}^{\text{adm}}$ is an admissible predicate variable.

$$(x : \{x : \textbf{real} \mid P_x(x)\}) \to (k : (u : \{u : \textbf{unit} \mid P_u(x)\}) \to \{r : \textbf{Prop} \mid P_k(x, r)\})$$
$$\to \{r : \textbf{Prop} \mid P_{\text{rw}}^{\text{adm}}(r; x)\} \tag{11}$$

R-Fix-ConstGen also requires that (11) must be a subtype of (5). After handling the fixed point, **ConstGen** handles lambda abstraction and if-then-else, which is standard and straightforward. In the then clause, we have an integration operator, to which **ConstGen** applies R-Unif-ConstGen (Fig. 8). R-Unif-ConstGen introduces a fresh integration predicate variable P^{int} for the codomain type of the integrand. After finishing the constraint generation and simplification, we get the following CHC constraints.

$$\left. \begin{array}{rcl} x \geq -2 & \implies & P_x(x) \\ x \geq -2 \wedge P_u(x) \wedge r = 0 & \implies & P_k(x, r) \\ x \geq -2 \wedge P_{\text{rw}}^{\text{adm}}(r; x) & \implies & r \leq |2 \cdot x + 4| \end{array} \right\} \quad \text{(11) is a subtype of (5)}$$

$$\left. \begin{array}{rcl} P_x(x) \wedge x \geq 0 \wedge \textbf{Integ}_{\textbf{unif}}(P^{\text{int}})(r; x) & \implies & P_{\text{rw}}^{\text{adm}}(r; x) \\ P_x(x) \wedge x \geq 0 \wedge 0 \leq y \wedge y \leq 1 & \implies & P_x(x + 3 \cdot y - 2) \\ P_x(x) \wedge x \geq 0 \wedge 0 \leq y \wedge y \leq 1 \wedge P_{\text{rw}}^{\text{adm}}(r; x + 3 \cdot y - 2) & \implies & P^{\text{int}}(r + 1; y; x) \end{array} \right\} \quad \text{then clause}$$

$$\left. \begin{array}{rcl} P_x(x) \wedge x < 0 & \implies & P_u(x) \\ P_x(x) \wedge x < 0 \wedge P_k(x, r) & \implies & P_{\text{rw}}^{\text{adm}}(r; x) \end{array} \right\} \quad \text{else clause}$$

These constraints are satisfiable: Let (11) be equal to (5) and $P^{\text{int}}(r; y; x) = (r \leq 2 \cdot x + 6 \cdot y + 1)$. Note that under this assignment, we have $\textbf{Integ}_{\textbf{unif}}(P^{\text{int}})(r; x) = (r \leq \int_0^1 2 \cdot x + 6 \cdot y + 1 \, \mathrm{d}y) = (r \leq 2 \cdot x + 4)$. □

5.2 General Syntactic Conditions for Admissible Formulas

We provide general syntactic conditions for admissible formulas in Fig. 9. These rules give us hints when we implement a constraint solver for CHC with admissible predicates. We consider a judgement of the form $\Gamma \vdash \text{adm}(v, \phi)$, which means ϕ is admissible at v, and provide derivation rules in Fig. 9. Those rules are sound, as stated below.

THEOREM 5.3. *If $\Gamma \vdash \text{adm}(v, \phi)$, then $\Gamma \vdash \phi$ is well-formed is admissible at v for any λ_{HFL}-model.* □

Note that for $v, M : \textbf{Prop}$, $\Gamma \vdash \text{adm}(v, v < M)$ and $\Gamma \vdash \text{adm}(v, M \leq v)$ are not derivable because $v < M$ and $M \leq v$ are not admissible in general (see Example 4.7 for counterexamples). This poses a difficulty in reasoning about lower bounds of the least fixed point in our dependent refinement type system. In general, reasoning about lower bounds for probabilistic programs is more difficult

$$(\Gamma, v : \tau) \setminus v \coloneqq \Gamma \qquad (\Gamma, x : \tau) \setminus v \coloneqq (\Gamma \setminus v), x : \tau$$

Adm-Leq
$$\frac{(v : \textbf{Prop}) \in \Gamma \qquad \Gamma \vdash M : \textbf{Prop}}{\Gamma \vdash \mathrm{adm}(v, v \leq_{\textbf{Prop}} M)}$$

Adm-Imp
$$\frac{(\Gamma \setminus v) \vdash \phi \qquad \Gamma \vdash \mathrm{adm}(v, \psi)}{\Gamma \vdash \mathrm{adm}(v, \phi \implies \psi)}$$

Adm-And
$$\frac{\Gamma \vdash \mathrm{adm}(v, \phi) \qquad \Gamma \vdash \mathrm{adm}(v, \psi)}{\Gamma \vdash \mathrm{adm}(v, \phi \wedge \psi)}$$

Adm-Or
$$\frac{\Gamma \vdash \mathrm{adm}(v, \phi) \qquad \Gamma \vdash \mathrm{adm}(v, \psi)}{\Gamma \vdash \mathrm{adm}(v, \phi \vee \psi)}$$

Fig. 9. Syntactic rules to guarantee admissibility of formulas.

than upper bounds. There are a few papers [Beutner and Ong 2021; Feng et al. 2023; Hark et al. 2020; McIver and Morgan 2005] on reasoning about lower bounds, but it is not straightforward to combine these methods with our general framework. We would like to tackle this in future work. Note that we can reason about lower bounds if an HFL term does not contain least fixed points.

6 IMPLEMENTATION AND EXPERIMENTS

Following Section 5, our type checker is implemented as an extension of RCaml, in which PCSat is used as a CHC solver. The CHC solver is based on a method called counterexample-guided inductive synthesis (CEGIS) with linear templates and extended to $\textbf{CHC}[\mathrm{adm}, \int]$-constraints. First, we will explain how we extend the constraint solver to admissible predicate variables and integrable predicate variables. Then, we will present and discuss experimental results.

6.1 CEGIS-Based CHC Solving

CEGIS [Solar-Lezama et al. 2006] is a method for constraint solving and consists of two components: a synthesiser and a validator. The synthesiser guesses a solution, and the validator checks whether a candidate solution is a genuine solution. Given a set of constraints (e.g. CHC constraints), the synthesiser and the validator iteratively interact with each other and gradually improve candidate solutions by accumulating counterexamples found by the validator.

The synthesiser of our CHC solver guesses a candidate solution using (affine) templates. The coefficients of templates are determined using counterexamples. The validator of our CHC solver checks a candidate solution using an SMT solver. If a candidate solution is not a genuine solution, the SMT solver gives new counterexamples, which are used in the next iteration of CEGIS.

6.2 Supporting Admissible Predicate Variables

The basic strategy for supporting new types of predicate variables is to appropriately restrict the search space of solutions. In our CHC solver, a search space for predicate variables is defined by a *template*, which is a set of formulas that contain unknown parameters. To support admissible predicate variables, we define *admissible templates* such that any instantiation of unknown parameters gives a formula that is syntactically admissible. Since the interpretation of **Prop** varies for each problem in Section 3, different problems need different admissible templates. We design and implement admissible templates for the problems considered in this paper using the syntactic rules for admissible predicates in Fig. 9 as a guide because those syntactic rules are sound for any λ_{HFL}-model (i.e., for any interpretation of **Prop**).

6.2.1 An Admissible Template for Weakest Pre-Expectations and Expected Costs. Let $P_{\mathrm{adm}}(v; \tilde{x})$ be an admissible predicate variable where v is a variable at which $P_{\mathrm{adm}}(v; \tilde{x})$ is expected to be admissible

and $\tilde{x} = \{x_1, \ldots, x_m\}$ is a set of variables. We assume that each variable is assigned a type and that the type of each variable belongs to $\{\textbf{int}, \textbf{real}, \textbf{Prop}\}$. This ensures that templates can be handled by SMT-based constraint solvers. In the implementation, this assumption can be satisfied by simply ignoring variables that are not typed as $\textbf{int}$, $\textbf{real}$, or $\textbf{Prop}$.

For weakest pre-expectations and expected costs ($[\![\textbf{Prop}]\!] = [0, \infty]$), the search space for $P_{\mathrm{adm}}(v; \tilde{x})$ is defined by the following admissible template.

$$d \cdot v \leq e \qquad \text{where} \qquad e \quad := \quad \textbf{if } \phi \textbf{ then } e_1 \textbf{ else } e_2 \quad | \quad |c_0 + \sum_{i=1}^{m} c_i \cdot x_i| \qquad (12)$$

Here, $d \in \mathbb{Z}$ ($d \geq 0$) and $c_i \in \mathbb{Z}$ ($i = 0, \ldots, m$) are unknown parameters; and ϕ is a conjunction of affine inequations over $\tilde{x}$. Note that we are essentially using rational numbers as unknown parameters since d serves as the denominator. Furthermore, $d = 0$ corresponds to the case of $v \leq \infty$. We assume that we have type conversion operators like $\textbf{int} \rightarrow \textbf{real}$ and the absolute-value operator $|-| : \textbf{real} \rightarrow \textbf{Prop}$ as basic operators; and type conversions are implicitly used in $c_0 + \sum_{i=1}^{m} c_i \cdot x_i$. Precisely speaking, ϕ is not a formula defined in Section 4 but a term of λ_{HFL} of type $1 + 1$. However, we don't have to worry too much about it because such a term is easily definable in λ_{HFL}.

LEMMA 6.1. *Any instantiation of the admissible template* (12) *is a formula admissible at* v.

PROOF. By induction on the definition of the admissible template. For the base case, $d \cdot v \leq |c_0 + \sum_{i=1}^{m} c_i \cdot x_i|$ is admissible by ADM-LEQ in Fig. 9. For the step case, $d \cdot v \leq \textbf{if } \phi \textbf{ then } e_1 \textbf{ else } e_2$ is admissible because the equivalent formula $(\phi \implies d \cdot v \leq e_1) \wedge (\neg \phi \implies d \cdot v \leq e_2)$ is admissible by ADM-AND and ADM-IMP. □

6.2.2 An Admissible Template for Cost Moment. Consider the case of cost moment analyses ($[\![\textbf{Prop}]\!] = [0, \infty]^n$). Similarly to the case of expected cost analyses, we define the following admissible template for $P_{\mathrm{adm}}(v; \tilde{x}) = P_{\mathrm{adm}}((v_1, \ldots, v_n); \tilde{x})$.

$$\bigwedge_{i=1}^{n} d_i \cdot v_i \leq e_i \qquad \text{where} \qquad e \quad := \quad \textbf{if } \phi \textbf{ then } e_1 \textbf{ else } e_2 \quad | \quad |c_0 + \sum_{i=1}^{m} c_i \cdot x_i|$$

Here, we decompose $v : \textbf{Prop}$ to a tuple of non-negative extended real numbers $(v_1, \ldots, v_n) : (\textbf{real}_{\geq 0}^{\infty})^n$. Similarly to Lemma 6.1, any instantiation of the admissible template above is an admissible formula at v.

6.2.3 An Admissible Template for Conditional Weakest Pre-Expectation. Consider the case of conditional weakest pre-expectation ($[\![\textbf{Prop}]\!] = [0, \infty] \times [0, 1]^{\mathrm{op}}$). We define an admissible template for $P_{\mathrm{adm}}((v_1, v_2); \tilde{x})$ as follows.

$$d_1 \cdot v_1 \leq e_1 \wedge d_2 \cdot v_2 \geq \min\{d_2, e_2\} \qquad \text{where} \qquad e \quad := \quad \textbf{if } \phi \textbf{ then } e_1 \textbf{ else } e_2 \quad | \quad |c_0 + \sum_{i=1}^{m} c_i \cdot x_i|$$

Note that $d_2 \cdot v_2 \geq \min\{d_2, e_2\}$ gives a *lower bound* of v_2 because $(v_1, v_2) \leq_{\textbf{Prop}} (u_1, u_2)$ is defined by $v_1 \leq u_1$ and $v_2 \geq u_2$. Note also that the right-hand side of $d_2 \cdot v_2 \geq \min\{d_2, e_2\}$ ensures the range of $\min\{1, e_2/d_2\}$ is subsumed by $[0, 1]$ if $d_2 > 0$. In other words, the admissible template is designed so that we have $(e_1/d_1, \min\{1, e_2/d_2\}) : \textbf{Prop}$.

6.3 Supporting Integrable Predicate Variables

Our implementation uses the following templates for integrable predicate variables. Recall that an integrable variable $P(v; y; \tilde{x})$ has to be instantiated as $v \leq N y$. We consider restricting the form of $N : \textbf{real} \rightarrow \textbf{Prop}$ to the affine expression $N = \lambda y.c \cdot y + N'$ where N' is an affine expression over $\tilde{x}$. That is, the template for $P(v; y; \tilde{x})$ is given by $v \leq c \cdot y + N'$. Here, note that y does not occur in N'.

Table 2. Experimental results. Benchmarks are listed in Appendix D. The timeout is set to 300 seconds.

Problem	Benchmark	Time (sec)
Weakest pre-expectation	`lics16_rec3` (Example 3.3)	timeout
	`lics16_rec3_ghost`	1.270
	`lics16_coins`	3.110
Expected cost analysis	`random_walk`	2.761
	`random_walk_unif` (Example 3.2)	7.508
	`coin_flip` (Example 1.1)	0.718
	`coin_flip_unif`	0.884
	`icfp21_walk`	3.532
	`icfp21_coupons`	timeout
	`lics16_fact`	3.383
Cost moment analysis	`coin_flip_ord2` (Example 3.4)	1.135
	`coin_flip_ord3`	4.040
Conditional weakest pre-expectation	`toplas18_ex4.4` (Example 3.5)	timeout
	`two_coin_conditioning`	1.079

Since $\mathbf{unif}(N)$ can be computed as $1/2 \cdot c + N'$ by the linearity of integration, we can instantiate $\mathbf{Integ_{unif}}(P)(v; \tilde{x})$ as $v \leq 1/2 \cdot c + N'$.

6.4 Results and Discussion

Table 2 shows the result of our experiments. Most of the benchmarks are collected from past papers on verification of probabilistic programs [Avanzini et al. 2021; Olmedo et al. 2018, 2016]. Specifications for benchmarks are annotated by hand as refinement types. Note that annotations are required for only top-level declarations, and safe inductive invariants for the least fixed points of λ_{HFL} are automatically synthesised by the constraint solver. Note also that our analysis is modular rather than whole-program because our implementation analyses a benchmark program function by function. Our benchmarks contain four verification problems for higher-order probabilistic programs discussed in this paper. The results were obtained on 12th Gen Intel(R) Core(TM) i7-1270P 2.20 GHz with 32 GB of memory.

The aim of the evaluation is to demonstrate that by instantiating our general framework with a specific refinement type checker and a CHC solver, we can indeed build a verifier that is effective for (a) 4 problem instances on cost moment analysis and conditional weakest pre-expectation of recursive probabilistic programs that cannot be handled by existing methods and (b) 10 problem instances on weakest pre-expectation and expected cost analysis of (higher-order) recursive probabilistic programs that few existing methods can handle. These problem instances shown in Appendix D are rather small, but they were selected because they pose challenges for existing automated methods, and our tool has been able to verify most of them fully automatically.

The experimental result (Table 2) shows that our implementation successfully solved most of the benchmarks even though we implemented it as a simple extension of the existing type checker for non-probabilistic programs. While some benchmarks could be solved by the current implementation, it is worth noting that this is not because we consider the verification of probabilistic programs. Rather, this is due to common issues with ordinary dependent refinement type systems for non-probabilistic programs. Therefore, by incorporating advanced techniques for ordinary dependent refinement type systems, our implementation can potentially overcome these issues.

For example, our implementation could not solve `lics16_rec3`. When reasoning about rec3 : **unit** → (**unit** → **Prop**) → **Prop**, the predicate on the result type **Prop** should be able to depend

on the result of the second argument of rec3, which has a function type **unit → Prop**. However, most implementations of dependent refinement type systems struggle to handle such situations effectively. One possible way to overcome this problem is to add an extra ghost parameter and define rec3 : **unit** → $[a : \textbf{Prop}]$ → (**unit** → $\{v : \textbf{Prop} \mid r \leq a\}$) → $\{v : \textbf{Prop} \mid r \leq c \cdot a\}$ as follows.

$$\textbf{let fix } \text{rec3 } x \, [a] \, k = 1/2 \cdot k \, () + 1/2 \cdot \text{rec3 } () \, (\lambda y.\text{rec3 } () \, [c^2 \cdot a] \, (\lambda y.\text{rec3 } () \, [c \cdot a] \, k)) \textbf{ in}$$
$$\text{rec3 } () \, [a] \, (\lambda y.1) \tag{13}$$

Here, we indicate the extra parameter a by square brackets $[-]$, and c is a constant such that $(\sqrt{5} - 1)/2 \leq c \leq 1$. Our implementation can verify that (13) is well-typed (`lics16_rec3_ghost` in Table 2). This idea is proposed in [Unno et al. 2013], in which they proved that the relative completeness of the refinement-type-based verification for *non-probabilistic* higher-order programs can be achieved by inserting extra ghost parameters at appropriate positions. We conjecture that a similar result holds for the probabilistic case, but we leave it for future work.

Our implementation also suffered from the limitations about affine templates for predicate variables (`toplas18_ex4.4`). A possible remedy for this issue is to use other constraint solvers that support more expressive templates, e.g., CEGIS for polynomial invariants [Sharma et al. 2013], a synthesis method based on polynomial templates [Chatterjee et al. 2020], and a method based on recurrence relations [Kincaid et al. 2017].

Remark 6.2. The restriction of integrable predicate variables to affine templates above arises from the following two requirements. (1) The integration of templates must be computable. (2) The class of templates must be solvable in constraint solvers. Affine templates satisfy both requirements: We can easily integrate affine templates by substituting the expected value, and affine constraints are easy to solve. If we consider piecewise affine templates, it would not be difficult to handle them in our constraint solver, but computing integral would be difficult. On the other hand, computing the integration of polynomial templates is easy because we only have to substitute n-th moment $\mathbb{E}[x^n]$ for x^n, but our current constraint solver has difficulty in solving polynomial constraints.

There are a few existing tools that aim at the verification of higher-order probabilistic programs. It is difficult to make a fair comparison because those tools have more or less different problem settings from ours. However, we would like to note that none of them supports all the benchmarks listed in Table 2. [Beutner and Ong 2021] proposed a verification tool for almost sure termination (AST). Note that the AST verification is a special case of the lower-bound verification of the weakest pre-expectation, and the positive-AST (AST within finite expected runtime) verification is the upper-bound verification of the expected cost. The input format of their tool is rather restrictive compared to ours: input programs must be of the form **fix** $f.\lambda x : \textbf{real}.M$. Their tool doesn't accept nested recursion or recursive functions that have more than one argument. As a result, their tool accepts only four of the benchmarks in Table 2 (`coin_flip`, `lics16_fact`, `random_walk`, `random_walk_unif`) and was able to prove AST of only one of them (`coin_flip` in 0.156 sec). [Avanzini et al. 2023] proposed a verification tool for the upper bounds of the weakest pre-expectation and the expected cost of *imperative* probabilistic programs with recursion. Their tool accepts five of the benchmarks in Table 2 (`coin_flip`, `lics16_coins`, `lics16_fact`, `lics16_rec3`, `random_walk`) and was able to solve four of them (`coin_flip` in 0.025 sec, `lics16_coins` in 0.038 sec, `lics16_fact` in 0.035 sec, `random_walk` in 0.036 sec).

7 RELATED WORK

7.1 Weakest Pre-Expectations and Expected Costs

The *weakest pre-expectation transformer* [McIver and Morgan 2005; Olmedo et al. 2016] is a generalisation of the weakest precondition transformer [Dijkstra 1975]. This notion is used to verify

properties such as termination probabilities and probabilistic invariants [Bao et al. 2022; Batz et al. 2023] for imperative probabilistic programming languages. The *expected runtime transformer* [Kaminski et al. 2018] is a similar notion proposed for verification of expected costs. Concerning expected cost analyses, the notion of *ranking supermartingales* [Chakarov and Sankaranarayanan 2013] has also been studied and applied to automatic verification of almost sure termination. It is known that ranking supermartingales give upper bounds of the expected cost and that this can be understood order-theoretically [Takisaka et al. 2018]. *Ranking supermartingales for higher moments* are proposed [Kura et al. 2019] as an extension of ranking supermartingales that gives upper bounds of the higher moments of runtime (i.e. the expected value of *powers* of runtime). By exploiting the relationship between ranking supermartingales and the expected runtime transformer, a cost moment transformer is obtained as a generalisation of the expected runtime transformer [Aguirre et al. 2022]. The *conditional weakest pre-expectation transformer* [Olmedo et al. 2018] is another extension of the weakest pre-expectation transformer for probabilistic programs with conditioning. Studies listed above are targeted at imperative probabilistic programs, and higher-order probabilistic programs are out of scope.

7.2 Verification of Higher-Order Probabilistic Programs

[Avanzini et al. 2021] proposed a CPS transformation for the expected cost analysis for higher-order probabilistic programs. Their CPS transformation translates a probabilistic program into a pure term of simply typed lambda calculus with fixed points whose denotation gives the expected cost. They also provided a program logic EHOL to reason about CPS-transformed programs. In contrast to our approach, they did not introduce an algorithm for automated verification, nor did they incorporate continuous distributions into their language.

There are several program logics proposed for verifying higher-order probabilistic programs [Aguirre et al. 2021, 2017; Sato et al. 2019]. Some of them have been implemented on interactive theorem provers [Hirata et al. 2022], but these approaches largely depend on human intervention.

Probabilistic extensions of higher-order model checking have been studied. A probabilistic extension of higher-order recursion schemes is studied in [Kobayashi et al. 2020]. They provide theoretical results on the decidability and hardness of model checking problems and experimental results. A probabilistic extension of higher-order fixed point logic (PHFL) is proposed in [Mitani et al. 2020]. It is worth mentioning that our λ_{HFL} and their PHFL share most of the syntax of HFL.

As for automated verification, [Beutner and Ong 2021] proposed algorithms to verify lower bounds of termination probability and almost sure termination for higher-order probabilistic programs with continuous distributions. Although their language incorporates soft-conditioning, what they verify differs from conditional weakest pre-expectations considered in our work and [Olmedo et al. 2018] because their semantics essentially ignores soft-conditioning. For example, "score(0); diverge" is equivalent to "diverge" in their semantics where "diverge" is a diverging program, whereas the conditional weakest pre-expectations for these two programs are different. [Wang et al. 2020] studied type-based amortised expected cost analysis for the expected cost of higher-order probabilistic programs. However, their type system does not support continuous distributions.

7.3 Verification of Higher-Order Non-Probabilistic Programs

Dijkstra monads enable specifying and verifying programs via weakest precondition transformers. In [Ahman et al. 2017], a CPS transformation and refinement types are applied for automated verification of higher-order programs. However, our approach and theirs are different. We apply a CPS transformation to a program to be verified to obtain the weakest precondition transformer and then apply refinement types to reason about the weakest precondition transformer. In contrast,

[Ahman et al. 2017] applies a CPS transformation to a monad to define a Dijkstra monad and then uses the Dijkstra monad to verify programs. Their work does consider refinement types and Dijkstra monads at the same time, but these are used as independent mechanisms. In fact, [Maillard et al. 2019] studies Dijkstra monads in a setting without refinement types. Another notable difference is that the method proposed in [Ahman et al. 2017] cannot be applied to probabilistic programs. This is because their soundness is proved using deterministic program semantics.

[Katsura et al. 2020] proposed a refinement type system for a higher-order fixed-point logic ν**HFL**. Their ν**HFL** can be understood as a special case of our λ_{HFL}. If we interpret **Prop** as $\{\text{true}, \text{false}\}$ and choose a set **BO** of basic operators appropriately, we get the quantifier-free fragment of ν**HFL**. Moreover, we get the full ν**HFL** by adding universal quantifiers to our λ_{HFL}, which is a straightforward extension. We note that their refinement type system is developed for non-probabilistic programs and cannot be applied to the problems that we considered in this paper.

8 CONCLUSIONS AND FUTURE WORK

We proposed a dependent refinement type system for a generalised higher-order fixed point logic. Combined with CPS-based encodings of properties of higher-order probabilistic programs, our approach provided an algorithm to verify several properties of probabilistic programs with continuous distributions and (hard) conditioning. Our approach can seamlessly integrate with verification techniques developed for non-probabilistic programs. We implemented our approach and demonstrated its ability.

In future work, we would like to improve our implementation by combining advanced techniques for refinement types and CHC solving. For example, in Section 3, we gave specifications of probabilistic programs using concrete upper bounds provided by hand. To avoid specifying concrete upper bounds, it might be possible to use techniques like CHC optimisation [Gu et al. 2023] and refinement type optimization [Hashimoto and Unno 2015]. We are also interested in automated inference of ghost parameters [Unno et al. 2013] because we had to specify ghost parameters manually to solve Example 3.3 in the current implementation. On the theoretical side, our dependent refinement type system currently has the following limitations: we cannot reason about lower bounds of least fixed points (dually upper bounds of greatest fixed points). Verifying lower bounds for probabilistic programs is studied in [Beutner and Ong 2021; Feng et al. 2023; Hark et al. 2020; McIver and Morgan 2005]. We would like to study a category-theoretic abstraction of these studies and combine it with our type system. Making full use of the generality of our approach is another direction of future work. For example, the combination of nondeterminism and probability is common in the verification of probabilistic programs. Therefore, we would like to investigate an appropriate λ_{HFL}-model for this situation. Expected cost analyses for quantum programs [Avanzini et al. 2022; Liu et al. 2022] could be another interesting application of our approach.

ACKNOWLEDGMENTS

REFERENCES

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. *Proceedings of the ACM on Programming Languages* 5, ICFP (Aug. 2021), 1–30. https://doi.org/10.1145/3473598

Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2017. A Relational Logic for Higher-Order Programs. *Proceedings of the ACM on Programming Languages* 1, ICFP (Aug. 2017), 1–29. https://doi.org/10.1145/3110265

Alejandro Aguirre, Shin-ya Katsumata, and Satoshi Kura. 2022. Weakest Preconditions in Fibrations. *Mathematical Structures in Computer Science* 32, 4 (Oct. 2022), 472–510. https://doi.org/10.1017/S0960129522000330

Danel Ahman, Cătălin Hriţcu, Kenji Maillard, Guido Martínez, Gordon Plotkin, Jonathan Protzenko, Aseem Rastogi, and Nikhil Swamy. 2017. Dijkstra Monads for Free. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages - POPL 2017*. ACM Press, Paris, France, 515–529. https://doi.org/10.1145/3009837.3009878

Martin Avanzini, Gilles Barthe, and Ugo Dal Lago. 2021. On Continuation-Passing Transformations and Expected Cost Analysis. *Proceedings of the ACM on Programming Languages* 5, ICFP (Aug. 2021), 1–30. https://doi.org/10.1145/3473592

Martin Avanzini, Georg Moser, Romain Pechoux, Simon Perdrix, and Vladimir Zamdzhiev. 2022. Quantum Expectation Transformers for Cost Analysis. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, Haifa Israel, 1–13. https://doi.org/10.1145/3531130.3533332

Martin Avanzini, Georg Moser, and Michael Schaper. 2023. Automated Expected Value Analysis of Recursive Programs. *Proceedings of the ACM on Programming Languages* 7, PLDI (June 2023), 1050–1072. https://doi.org/10.1145/3591263

Jialu Bao, Nitesh Trivedi, Drashti Pathak, Justin Hsu, and Subhajit Roy. 2022. Data-Driven Invariant Learning for Probabilistic Programs. In *Computer Aided Verification*. Vol. 13371. Springer International Publishing, Cham, 33–54. https://doi.org/10.1007/978-3-031-13185-1_3

Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science, Vol. 13994)*. Springer Nature Switzerland, Cham, 410–429. https://doi.org/10.1007/978-3-031-30820-8_25

Raven Beutner and Luke Ong. 2021. On Probabilistic Termination of Functional Programs with Continuous Distributions. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, Virtual Canada, 1312–1326. https://doi.org/10.1145/3453483.3454111

Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan, and Andrey Rybalchenko. 2015. Horn Clause Solvers for Program Verification. In *Fields of Logic and Computation II*. Vol. 9300. Springer International Publishing, Cham, 24–51. https://doi.org/10.1007/978-3-319-23534-9_2

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification (Lecture Notes in Computer Science, Vol. 8044)*. Springer Berlin Heidelberg, Saint Petersburg, Russia, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. 2020. Polynomial Invariant Generation for Non-Deterministic Recursive Programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, London UK, 672–687. https://doi.org/10.1145/3385412.3385969

Luis Damas and Robin Milner. 1982. Principal Type-Schemes for Functional Programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '82*. ACM Press, Albuquerque, Mexico, 207–212. https://doi.org/10.1145/582153.582176

Swaraj Dash, Younesse Kaddar, Hugo Paquet, and Sam Staton. 2023. Affine Monads and Lazy Structures for Bayesian Programming. *Proceedings of the ACM on Programming Languages* 7, POPL (Jan. 2023), 1338–1368. https://doi.org/10.1145/3571239

Edsger W. Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM* 18, 8 (Aug. 1975), 453–457. https://doi.org/10.1145/360933.360975

Shenghua Feng, Mingshuai Chen, Han Su, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Naijun Zhan. 2023. Lower Bounds for Possibly Divergent Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (April 2023), 696–726. https://doi.org/10.1145/3586051

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Kallista A. Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A Language for Generative Models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*. AUAI Press, 220–229.

Yu Gu, Takeshi Tsukada, and Hiroshi Unno. 2023. Optimal CHC Solving via Termination Proofs. *Proceedings of the ACM on Programming Languages* 7, POPL (Jan. 2023), 604–631. https://doi.org/10.1145/3571214

Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. 2020. Aiming Low Is Harder: Induction for Lower Bounds in Probabilistic Program Verification. *Proceedings of the ACM on Programming Languages* 4, POPL (Jan. 2020), 1–28. https://doi.org/10.1145/3371105

Kodai Hashimoto and Hiroshi Unno. 2015. Refinement Type Inference via Horn Constraint Optimization. In *Static Analysis*. Vol. 9291. Springer Berlin Heidelberg, Berlin, Heidelberg, 199–216. https://doi.org/10.1007/978-3-662-48288-9_12

Claudio Hermida. 1993. *Fibrations, Logical Predicates and Indeterminates*. Ph. D. Dissertation. University of Edinburgh, UK.

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A Convenient Category for Higher-Order Probability Theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, Reykjavik, Iceland, 1–12. https://doi.org/10.1109/LICS.2017.8005137

Michikazu Hirata, Yasuhiko Minamide, and Tetsuya Sato. 2022. Program Logic for Higher-Order Probabilistic Programs in Isabelle/HOL. In *Functional and Logic Programming*. Vol. 13215. Springer International Publishing, Cham, 57–74. https://doi.org/10.1007/978-3-030-99461-7_4

Bart Jacobs. 2001. *Categorical Logic and Type Theory* (paperback ed ed.). Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier, Amsterdam.

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5 (Aug. 2018), 1–68. https://doi.org/10.1145/3208102

Hiroyuki Katsura, Naoki Iwayama, Naoki Kobayashi, and Takeshi Tsukada. 2020. A New Refinement Type System for Automated νHFL$_\mathbb{Z}$ Validity Checking. In *Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 12470)*. Springer International Publishing, Cham, 86–104. https://doi.org/10.1007/978-3-030-64437-6_5

Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas Reps. 2017. Compositional Recurrence Analysis Revisited. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, Barcelona Spain, 248–262. https://doi.org/10.1145/3062341.3062373

Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. 2020. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. *Logical Methods in Computer Science ; Volume 16* (Oct. 2020), Issue 4 ; 18605974. https://doi.org/10.23638/LMCS-16(4:2)2020

Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe. 2018. Higher-Order Program Verification via HFL Model Checking. In *Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 10801)*. Springer International Publishing, 711–738. https://doi.org/10.1007/978-3-319-89884-1_25

Satoshi Kura. 2021. A General Semantic Construction of Dependent Refinement Type Systems, Categorically. In *Foundations of Software Science and Computation Structures (Lecture Notes in Computer Science, Vol. 12650)*. Springer International Publishing, 406–426. https://doi.org/10.1007/978-3-030-71995-1_21

Satoshi Kura. 2023. Higher-Order Weakest Precondition Transformers via a CPS Transformation. arXiv:2301.09997 [cs]

Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science, Vol. 11428)*. Springer, Prague, Czech Republic, 135–153. https://doi.org/10.1007/978-3-030-17465-1_8

Junyi Liu, Li Zhou, Gilles Barthe, and Mingsheng Ying. 2022. Quantum Weakest Preconditions for Reasoning about Expected Runtimes of Quantum Programs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, Haifa Israel, 1–13. https://doi.org/10.1145/3531130.3533327

Kenji Maillard, Danel Ahman, Robert Atkey, Guido Martínez, Cătălin Hriţcu, Exequiel Rivas, and Éric Tanter. 2019. Dijkstra Monads for All. *Proceedings of the ACM on Programming Languages* 3, ICFP (July 2019), 1–29. https://doi.org/10.1145/3341708

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, New York.

Yo Mitani, Naoki Kobayashi, and Takeshi Tsukada. 2020. A Probabilistic Higher-Order Fixpoint Logic. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 167)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 19:1–19:22. https://doi.org/10.4230/LIPIcs.FSCD.2020.19

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *Functional and Logic Programming*. Vol. 9613. Springer International Publishing, Cham, 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle Mciver. 2018. Conditioning in Probabilistic Programming. *ACM Transactions on Programming Languages and Systems* 40, 1 (March 2018), 1–50. https://doi.org/10.1145/3156018

Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '16*. ACM Press, New York, NY, USA, 672–681. https://doi.org/10.1145/2933575.2935317

Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. 2008. Liquid Types. In *Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '08*. ACM Press, Tucson, AZ, USA, 159. https://doi.org/10.1145/1375581.1375602

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. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–30. https://doi.org/10.1145/3290351

Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, Percy Liang, and Aditya V. Nori. 2013. A Data Driven Approach for Algebraic Loop Invariants. In *Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 7792)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 574–592. https://doi.org/10.1007/978-3-642-37036-6_31

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. 2006. Combinatorial Sketching for Finite Programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS-XII*. ACM Press, San Jose, California, USA, 404. https://doi.org/10.1145/1168857.1168907

Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. 2018. Ranking and Repulsing Supermartingales for Reachability in Probabilistic Programs. In *Automated Technology for Verification and Analysis (Lecture Notes in Computer Science, Vol. 11138)*. Springer International Publishing, Cham, 476–493. https://doi.org/10.1007/978-3-030-01090-4_28

Hiroshi Unno and Naoki Kobayashi. 2009. Dependent Type Inference with Interpolants. In *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*. ACM, Coimbra Portugal, 277–288. https://doi.org/10.1145/1599410.1599445

Hiroshi Unno, Tachio Terauchi, and Naoki Kobayashi. 2013. Automating Relatively Complete Verification of Higher-Order Functional Programs. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '13*. ACM Press, Rome, Italy, 75. https://doi.org/10.1145/2429069.2429081

Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A Domain Theory for Statistical Probabilistic Programming. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–29. https://doi.org/10.1145/3290349

Di Wang, David M. Kahn, and Jan Hoffmann. 2020. Raising Expectations: Automating Expected Cost Analysis with Types. *Proceedings of the ACM on Programming Languages* 4, ICFP (Aug. 2020), 1–31. https://doi.org/10.1145/3408992

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*. 1024–1032.

A   DETAILS OF HFL

Our λ_{HFL} is designed based on [Kura 2023].

Let **Base** be a set of base types and **BO** be a set of typed signatures of basic operations.

A.1   Typing Rules

S-Var
$$\frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}$$

S-BasicOp
$$\frac{\Gamma \vdash M : \mathrm{ar}(\mathbf{op})}{\Gamma \vdash \mathbf{op}\, M : \mathrm{car}(\mathbf{op})}$$

S-Unit
$$\frac{}{\Gamma \vdash () : 1}$$

S-Pair
$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash (M, N) : \sigma \times \tau}$$

S-Fst
$$\frac{\Gamma \vdash M : \sigma_1 \times \sigma_2}{\Gamma \vdash \pi_1\, M : \sigma_1}$$

S-Snd
$$\frac{\Gamma \vdash M : \sigma_1 \times \sigma_2}{\Gamma \vdash \pi_2\, M : \sigma_2}$$

S-Case0
$$\frac{\Gamma \vdash M : 0}{\Gamma \vdash \delta(M) : \tau}$$

S-Inj
$$\frac{\Gamma \vdash M : \sigma_i}{\Gamma \vdash \iota_i\, M : \sigma_1 + \sigma_2}$$

S-Case2
$$\frac{\Gamma \vdash M : \sigma_1 + \sigma_2 \qquad \Gamma, x_1 : \sigma_1 \vdash M_1 : \tau \qquad \Gamma, x_2 : \sigma_2 \vdash M_2 : \tau}{\Gamma \vdash \delta(M, x_1 : \sigma_1.M_1, x_2 : \sigma_2.M_2) : \tau}$$

S-Abs
$$\frac{\Gamma, x : \sigma \vdash M : \mathbf{Prop}}{\Gamma \vdash \lambda x : \sigma.M : \sigma \to \mathbf{Prop}}$$

S-App
$$\frac{\Gamma \vdash M : \sigma \to \mathbf{Prop} \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\, N : \mathbf{Prop}}$$

S-Fix
$$\frac{\Gamma, f : \sigma \to \mathbf{Prop}, \vdash M : \sigma \to \mathbf{Prop}}{\Gamma \vdash \mathbf{fix}\, f.M : \sigma \to \mathbf{Prop}}$$

A.2   Semantics

Let $\mathbb{C}$ be an $\omega\mathbf{CPO}$-enriched cartesian closed category. The underlying (**Set**-enriched) category of $\mathbb{C}$ is denoted by $\mathbb{C}_0$.

Definition A.1. An object $A \in \mathbb{C}$ has a *bottom element* if there exists $\perp^A : 1 \to A$ such that for any $f : 1 \to A$, we have $\perp^A \leq f$ with respect to the partial order on $\mathbb{C}(1, A)$.

Definition A.2. We say $\mathbb{C}$ *admits a lifting strong monad* if

- the algebraic theory defined by one constant (nullary operation) $\perp$ and one inequation $\perp \leq x$ has free algebras, that is, the following universal property holds: for any $X \in \mathbb{C}$, there exists $X_\perp \in \mathbb{C}$ and $\eta_X : X \to X_\perp$ such that if $f : X \to A$ is such that $A \in \mathbb{C}$ has a bottom element, then there exists a unique $\overline{f} : X_\perp \to A$ such that $\overline{f} \circ \eta_X = f$, and
- the induced lifting monad $(-)_\perp$ is strong.

Note that an Eilenberg–Moore algebra of the lifting monad $(-)_\perp$ is just an object $A \in \mathbb{C}$ with a bottom element. Note also that a lifting monad defines a uniform $(-)_\perp$-fixed-point operator $(-)^\dagger : \mathbb{C}_0(X_\perp, X_\perp) \to \mathbb{C}_0(1, X_\perp)$ for any $X \in \mathbb{C}$.

$$f^\dagger : 1 \to X_\perp \quad := \quad \sup_n f^n \circ \perp^{X_\perp} \qquad \text{for any } f : X_\perp \to X_\perp$$

It is known that a uniform T-fixed-point operator uniquely extends to a parameterised uniform T-fixed-point operator $(-)^\dagger : \mathbb{C}_0(X \times A, A) \to \mathbb{C}_0(X, A)$ for any Eilenberg–Moore T-algebra $\zeta : TA \to A$.

Definition A.3 (λ_{HFL}-model). A λ_{HFL}-*model* is a tuple $\mathcal{A} = (\mathbb{C}, A, a, \Omega)$ where

- $\mathbb{C}$ is an $\omega\mathbf{CPO}$-enriched bicartesian closed category that admits a lifting monad.
- $A : \mathbf{Base} \to \mathbb{C}_0$ is an interpretation of each base type.
- For each $\mathbf{op} : \mathrm{ar}(\mathbf{op}) \to \mathrm{car}(\mathbf{op})$, $a(\mathbf{op}) : \mathcal{A}[\![\mathrm{ar}(\mathbf{op})]\!] \to \mathcal{A}[\![\mathrm{car}(\mathbf{op})]\!]$ is an interpretation of $\mathbf{op}$. Here, $\mathcal{A}[\![-]\!]$ is the interpretation of types defined in Definition A.4.

- $\Omega \in \mathbb{C}$ has a bottom element $\bot^{\Omega} : 1 \to \Omega$.

Note that the definition above is simplified compared to [Kura 2023]: we don't require a pseudo-lifting strong monad T on $\mathbb{C}$ and an EM T-algebra on Ω. This is because we unified effect-free constants and modal operators as basic operators. If we have a strong monad from $(-)_{\bot}$ to T, then an EM T-algebra on Ω induces an EM $(-)_{\bot}$-algebra on Ω, and both the uniform T-fixed-point operator and the uniform $(-)_{\bot}$-fixed-point operator give the same interpretation of fixed points.

Definition A.4 (interpretation of types). For each type σ, its interpretation $\mathcal{A}[\![\sigma]\!]$ is defined as follows.

$$\mathcal{A}[\![b]\!] \coloneqq Ab \qquad \mathcal{A}[\![\mathbf{Prop}]\!] \coloneqq \Omega \qquad \mathcal{A}[\![0]\!] \coloneqq 0 \qquad \mathcal{A}[\![\sigma + \tau]\!] \coloneqq \mathcal{A}[\![\sigma]\!] + \mathcal{A}[\![\tau]\!]$$
$$\mathcal{A}[\![1]\!] \coloneqq 1 \qquad \mathcal{A}[\![\sigma \times \tau]\!] \coloneqq \mathcal{A}[\![\sigma]\!] \times \mathcal{A}[\![\tau]\!] \qquad \mathcal{A}[\![\sigma \to \mathbf{Prop}]\!] \coloneqq \mathcal{A}[\![\sigma]\!] \Rightarrow \mathcal{A}[\![\mathbf{Prop}]\!]$$

Definition A.5 (interpretation of contexts). The interpretation $\mathcal{A}[\![\Gamma]\!]$ of a context is defined as follows.

$$\mathcal{A}[\![\cdot]\!] \coloneqq 1 \qquad \mathcal{A}[\![\Gamma, x : \sigma]\!] \coloneqq \mathcal{A}[\![\Gamma]\!] \times \mathcal{A}[\![\sigma]\!]$$

Definition A.6 (interpretation of terms). For any well-typed term $\Gamma \vdash M : \sigma$, its interpretation $\mathcal{A}[\![M]\!] = \mathcal{A}[\![\Gamma \vdash M : \sigma]\!] : \mathcal{A}[\![\Gamma]\!] \to \mathcal{A}[\![\sigma]\!]$ is defined as follows.

$$\mathcal{A}[\![\Gamma, x : \sigma \vdash y : \tau]\!] \coloneqq \begin{cases} \mathcal{A}[\![\Gamma \vdash y : \tau]\!] \circ \pi_1 & x \neq y \\ \pi_2 & x = y \end{cases}$$

$$\mathcal{A}[\![\mathbf{op}\ M]\!] \coloneqq a(\mathbf{op}) \circ \mathcal{A}[\![M]\!] \qquad \mathcal{A}[\![()]\!] \coloneqq\ ! \qquad \mathcal{A}[\![(M_1, M_2)]\!] \coloneqq \langle \mathcal{A}[\![M_1]\!], \mathcal{A}[\![M_2]\!] \rangle$$

$$\mathcal{A}[\![\pi_i\ M]\!] \coloneqq \pi_i \circ \mathcal{A}[\![M]\!] \qquad \mathcal{A}[\![\delta(M)]\!] \coloneqq\ ? \circ \mathcal{A}[\![M]\!] \qquad \mathcal{A}[\![\iota_i\ M]\!] \coloneqq \iota_i \circ \mathcal{A}[\![M]\!]$$

$$\mathcal{A}[\![\delta(M, x_1.M_1, x_2.M_2)]\!] \coloneqq [\mathcal{A}[\![M_1]\!], \mathcal{A}[\![M_2]\!]] \circ [\mathrm{id} \times \iota_1, \mathrm{id} \times \iota_2]^{-1} \circ \langle \mathrm{id}, \mathcal{A}[\![M]\!] \rangle$$

$$\mathcal{A}[\![\lambda x.M]\!] \coloneqq \Lambda(\mathcal{A}[\![M]\!]) \qquad \mathcal{A}[\![M\ N]\!] \coloneqq \mathbf{ev} \circ \langle \mathcal{A}[\![M]\!], \mathcal{A}[\![N]\!] \rangle \qquad \mathcal{A}[\![\mathbf{fix}\ f.M]\!] \coloneqq (\mathcal{A}[\![M]\!])^{\dagger}$$

B   DETAILS OF REFINEMENT TYPE SYSTEM

B.1   Formulas

B.1.1   Well-formed formulas. We define *well-formed formulas* $\Gamma \vdash \phi$ as follows.

$$\frac{\mathbf{ap} : \sigma \qquad \Gamma \vdash M : \sigma}{\Gamma \vdash \mathbf{ap}(M)} \qquad \frac{}{\Gamma \vdash \top} \qquad \frac{}{\Gamma \vdash \bot} \qquad \frac{\Gamma \vdash \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi \implies \phi} \qquad \frac{\Gamma \vdash \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi \wedge \phi}$$

$$\frac{\Gamma \vdash \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi \vee \phi}$$

B.1.2   Semantics. We assume that readers are familiar with fibrations. See [] for the introduction to fibrations.

Notations: Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration.

- $\mathbb{E}_I$ is the fibre category over $I \in \mathbb{B}$.
- The reindexing functor along $u : I \to J$ is denoted by $u^* : \mathbb{E}_J \to \mathbb{E}_I$.
- A cartesian lifting over $u : I \to J$ is denoted by $\overline{u}(Y) : u^*Y \to Y$ where $Y \in \mathbb{E}_J$.

We define a fibration $p : \mathbb{P} \to \mathbb{C}$ by the change-of-base construction.

$$\begin{array}{ccc} \mathbb{P} & \longrightarrow & \mathbf{Sub}(\mathbf{Set}) \\ {\scriptstyle p}\downarrow & \lrcorner & \downarrow{\scriptstyle \mathbf{sub}_{\mathbb{C}}} \\ \mathbb{C} & \xrightarrow{\ \mathbb{C}(1,-)\ } & \mathbf{Set} \end{array}$$

An object in $\mathbb{P}$ is a pair (I, P) such that $I \in \mathbb{C}$ and $P \subseteq \mathbb{C}(1, I)$ and a morphism $f : (I, P) \to (J, Q)$ is a morphism $f : I \to J$ such that for any $x \in P$, $f \circ x \in Q$. We sometimes omit I and write $P = (I, P)$ when I is clear from the context.

LEMMA B.1. $p : \mathbb{P} \to \mathbb{C}$ is a bifibration with a fibred bi-cc structure and simple products/coproducts.

PROOF. The subobject fibration $\mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ is a bifibration with a fibred bi-cc structure and simple products/coproducts, and U preserves finite products (see [Hermida 1993, 4.3.1] and [Jacobs 2001, Lemma 1.9.14]).   □

Notation:
- Fibred initial/terminal object functor are $\bot, \top : \mathbb{C} \to \mathbb{P}$.
- A fibred product, a fibred coproduct, and a fibred exponential for $X, Y \in \mathbb{P}_I$ are denoted by $X \wedge Y, X \vee Y, X \implies Y$.
- For simple products and coproducts, we write $\forall, \exists : \mathbb{P}_{X \times Y} \to \mathbb{P}_X$.

Definition B.2 (interpretation of formulas). Let $\mathcal{P}$ be an interpretation of atomic predicates such that $\mathcal{P}(\mathbf{ap}) \in \mathbb{P}_{\mathcal{A}[\![\mathrm{ar}(\mathbf{ap})]\!]}$ (i.e. $\mathcal{P}(\mathbf{ap}) \subseteq \mathbb{C}(1, \mathcal{A}[\![\mathrm{ar}(\mathbf{ap})]\!])$) for any atomic predicate $\mathbf{ap} : \mathrm{ar}(\mathbf{ap}) \in \mathbf{AP}$. For any well-formed formula $\Gamma \vdash \phi$, we define $\mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \phi]\!]$ as follows.

$$\mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \mathbf{ap}(M)]\!] := (\mathcal{A}[\![M]\!])^* \mathcal{P}(\mathbf{ap}) = \{\gamma : 1 \to \mathcal{A}[\![\Gamma]\!] \mid \mathcal{A}[\![M]\!] \circ \gamma \in \mathcal{P}(\mathbf{ap})\}$$
$$\mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \top]\!] := \top \mathcal{A}[\![\Gamma]\!] \qquad \mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \bot]\!] := \bot \mathcal{A}[\![\Gamma]\!]$$
$$\mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \psi \wedge \phi]\!] := \mathcal{A}[\![\Gamma \vdash \psi]\!] \wedge \mathcal{A}[\![\Gamma \vdash \phi]\!] \qquad \mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \psi \vee \phi]\!] := \mathcal{A}[\![\Gamma \vdash \psi]\!] \vee \mathcal{A}[\![\Gamma \vdash \phi]\!]$$
$$\mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \psi \implies \phi]\!] := \mathcal{A}[\![\Gamma \vdash \psi]\!] \implies \mathcal{A}[\![\Gamma \vdash \phi]\!]$$

B.2 Well-Formed Contexts/Types

We define well-formed contexts $\vdash \dot{\Gamma}$ and well-formed types $\dot{\Gamma} \vdash \dot{\sigma}$ inductively so that any formula in $\dot{\Gamma}$ and $\dot{\sigma}$ is well-formed.

$$\frac{}{\vdash \cdot} \qquad\qquad \frac{\dot{\Gamma} \vdash \dot{\sigma}}{\vdash \dot{\Gamma}, x : \dot{\sigma}}$$

$$\frac{\vdash \dot{\Gamma} \qquad |\dot{\Gamma}|, v : \sigma \vdash \phi \qquad \sigma \in \mathbf{Base} \cup \{\mathbf{Prop}, 1\}}{\dot{\Gamma} \vdash \{v : \sigma \mid \phi\}} \qquad \frac{\dot{\Gamma}, x : \dot{\sigma} \vdash \{v : \mathbf{Prop} \mid \phi\}}{\dot{\Gamma} \vdash (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}}$$

$$\frac{\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}}{\dot{\Gamma} \vdash (x : \dot{\sigma}) \times \dot{\tau}} \qquad\qquad \frac{}{\dot{\Gamma} \vdash 0} \qquad\qquad \frac{\dot{\Gamma} \vdash \dot{\sigma} \qquad \dot{\Gamma} \vdash \dot{\tau}}{\dot{\Gamma} \vdash \dot{\sigma} + \dot{\tau}}$$

B.3 Semantics of Well-Formed Contexts/Types

Let $s_{\mathbb{C}} : s(\mathbb{C}) \to \mathbb{C}$ be the simple fibration on $\mathbb{C}$ (see [] for the definition). Since $\mathbb{C}$ is cartesian closed, $s_{\mathbb{C}} : s(\mathbb{C}) \to \mathbb{C}$ is a CCompC [Jacobs 2001, Theorem 10.5.5]. Since $\mathbb{C}$ is a bi-ccc, $s_{\mathbb{C}} : s(\mathbb{C}) \to \mathbb{C}$ has strong fibred coproducts.

More concretely, $s_{\mathbb{C}} : s(\mathbb{C}) \to \mathbb{C}$ has the following structures.

- An object in $s(\mathbb{C})$ is a pair (I, X) where $I, X \in \mathbb{C}$.
- A morphism in $s(\mathbb{C})$ is a pair $(u, f) : (I, X) \to (J, Y)$ where $u : I \to J$ and $f : I \times X \to Y$.
- Cartesian liftings are given as follows.

$$u^*(J, X) = (I, X) \xrightarrow{(u,\pi_2)} (J, X)$$

$$u^*(\mathrm{id},f)=(\mathrm{id},f\circ(u\times\mathrm{id}))\Big\downarrow \qquad\qquad \Big\downarrow(\mathrm{id},f)$$

$$s(\mathbb{C}) \qquad u^*(J, Y) = (I, Y) \xrightarrow{(u,\pi_2)} (J, Y)$$

$$\Big\downarrow s_{\mathbb{C}}$$

$$\mathbb{C} \qquad\qquad I \xrightarrow{\quad u \quad} J$$

- Fibred terminal object functor $1 : \mathbb{C} \to s(\mathbb{C})$.

$$1I = (I, 1)$$

- Comprehension functor $\{-\} : s(\mathbb{C}) \to \mathbb{C}$.

$$\{(I, X)\} = I \times X \qquad \mathbb{C}(I, \{(J, Y)\}) \cong s_{\mathbb{C}}(1I, (J, Y))$$

- Product $\prod_{(I,X)} : (s_{\mathbb{C}})_{I\times X} \to (s_{\mathbb{C}})_I$.

$$\prod_{(I,X)}(I \times X, Y) = (I, X \Rightarrow Y) \qquad \prod_{(I,X)}(\mathrm{id}_{I\times X}, f) = (\mathrm{id}, \Lambda(f \circ \langle \pi_1 \times \mathrm{id}, \mathbf{ev} \circ (\pi_2 \times \mathrm{id})\rangle)))$$

$$(s_{\mathbb{C}})_{I\times X}(\pi_1^*(I, Z), (I \times X, Y)) \cong (s_{\mathbb{C}})_I((I, Z), \prod_{(I,X)}(I \times X, Y))$$

$$\eta = \Lambda(\pi_2 \circ \pi_1) : (I, Y) \to (I, X \Rightarrow Y)$$

$$\epsilon = (\mathrm{id}, \mathbf{ev} \circ \sigma \circ (\pi_2 \times \mathrm{id})) : (I \times X, X \Rightarrow Y) \dot{\to} (I \times X, Y)$$

- Coproduct

$$\coprod_{(I,X)}(I \times X, Y) = (I, X \times Y) \qquad \coprod_{(I,X)}(\mathrm{id}_{I\times X}, f) = (\mathrm{id}_I,)$$

$$(s_{\mathbb{C}})_{I\times X}((I \times X, Y), \pi_1^*(I, Z)) \cong (s_{\mathbb{C}})_I(\coprod_{(I,X)}(I \times X, Y), (I, Z))$$

$$\eta = (\mathrm{id}, \pi_2 \times \mathrm{id}) : (I \times X, Y) \to (I \times X, X \times Y) \qquad \epsilon = (\mathrm{id}, \pi_2 \circ \pi_2) : (I, X \times Y) \to (I, Y)$$

$$\kappa = \alpha : (I \times X) \times Y \to I \times (X \times Y) \qquad \mathbf{fst} = (\mathrm{id}, \pi_1 \circ \pi_2) : (I, X \times Y) \to (I, X)$$

- Strong fibred initial objects $0 : \mathbb{C} \to s(\mathbb{C})$.

$$0I = (I, 0)$$

$$?_{(I,X)} = (\mathrm{id}, ? \circ \pi_2) : (I, 0) \to (I, X)$$

- Strong fibred binary coproducts

$$(I, X) + (I, Y) = (I, X + Y)$$

$$\iota_1 = (\mathrm{id}, \iota_1 \circ \pi_2) : (I, X) \to (I, X + Y) \qquad \iota_2 = (\mathrm{id}, \iota_2 \circ \pi_2) : (I, X) \to (I, X + Y)$$

The functor

$$\langle(\mathrm{id} \times \iota_1)^*, (\mathrm{id} \times \iota_2)^*\rangle : s(\mathbb{C})_{I\times(X+Y)} \to s(\mathbb{C})_{I\times X} \times s(\mathbb{C})_{I\times Y}$$

is fully faithful.

By applying the construction in [Kura 2021], we obtain a model of a dependent refinement type system.

LEMMA B.3. *We have a SCCompC $\{s_{\mathbb{C}} \mid p\} : \{s(\mathbb{C}) \mid \mathbb{P}\} \to \mathbb{P}$ with strong fibred coproducts. We also have the following morphism of SCCompCs.*

$$\begin{array}{ccc}
\{s(\mathbb{C}) \mid \mathbb{P}\} & \xrightarrow{\ u\ } & s(\mathbb{C}) \\
{\scriptstyle \{s_\mathbb{C}\mid p\}}\downarrow & & \downarrow{\scriptstyle s_\mathbb{C}} \\
\mathbb{P} & \xrightarrow{\ p\ } & \mathbb{C}
\end{array}$$

PROOF. The side condition for the existence of strong (binary) coproducts is satisfied as follows.

- $s(\mathbb{C}) \to \mathbb{C}$ is a CCompC with strong fibred coproducts.
- For any $u : I \to I'$ in $\mathbb{C}$ and $(I', X), (I', Y) \in s(\mathbb{C})_{I'}$, the following two squares are pullbacks.

$$\begin{array}{ccccc}
I \times X & \xrightarrow{\mathrm{id}\times \iota_1} & I \times (X + Y) & \xleftarrow{\mathrm{id}\times \iota_2} & I \times Y \\
{\scriptstyle u\times\mathrm{id}}\downarrow & & {\scriptstyle u\times\mathrm{id}}\downarrow & & \downarrow{\scriptstyle u\times\mathrm{id}} \\
I' \times X & \xrightarrow{\mathrm{id}\times \iota_1} & I' \times (X + Y) & \xleftarrow{\mathrm{id}\times \iota_2} & I' \times Y
\end{array}$$

- $p : \mathbb{P} \to \mathbb{C}$ is a fibred bi-ccc and thus satisfies Frobenius by [Hermida 1993, Remark 4.5.7].
- $p : \mathbb{P} \to \mathbb{C}$ is a cofibration.

□

More concretely, $\{s_\mathbb{C} \mid p\} : \{s(\mathbb{C}) \mid \mathbb{P}\} \to \mathbb{P}$ has the following structures.

- An object $((I, X), P, Q) \in \{s(\mathbb{C}) \mid \mathbb{P}\}$ consists of $(I, X) \in s(\mathbb{C})$, $P \subseteq \mathbb{C}(1, I)$, and $Q \subseteq \mathbb{C}(1, I \times X)$ such that if $\langle i, x \rangle \in Q$, then $i \in P$.
- A morphism in $\{s(\mathbb{C}) \mid \mathbb{P}\}$ (denoted by $(u, f) : ((I, X), P, Q) \;\dot\to\; ((J, Y), R, S)$) is a morphism $(u, f) : (I, X) \to (J, Y)$ in $s(\mathbb{C})$ such that $u : P \;\dot\to\; R$ and $\langle u \circ \pi_1, f \rangle : Q \;\dot\to\; S$ in $\mathbb{P}$, that is, for any $i \in P$, $u \circ i \in R$ and for any $\langle i, x \rangle \in Q$, $\langle u \circ i, f \circ \langle i, x \rangle \rangle \in R$.
- Reindexing functor:

$$\begin{array}{ccc}
((I, X), P, \pi^*_{(I,X)}P \wedge (u \times \mathrm{id})^*R) & \xrightarrow{\langle u, \pi_2 \rangle} & ((J, X), Q, R) \\
{\scriptstyle (\mathrm{id}, f\circ(u\times\mathrm{id}))}\downarrow & & \downarrow{\scriptstyle (\mathrm{id}, f)} \\
((I, Y), P, \pi^*_{(I,Y)}P \wedge (u \times \mathrm{id})^*S) & \xrightarrow{\langle u, \pi_2 \rangle} & ((J, Y), Q, S)
\end{array}$$

$$(I, P) \xrightarrow{\hspace{4cm} u \hspace{4cm}} (J, Q)$$

- Terminal object functor

$$\dot 1(I, P) = ((I, 1), P, \pi^*_1 P)$$

where $\pi^*_1 P = \{\langle i, \mathrm{id} \rangle : 1 \to I \times 1 \mid i \in P\}$

- Comprehension functor

$$\{((I, X), P, Q)\} = (I \times X, Q)$$

- Product

$$\dot{\prod}_{((I,X),P,Q)} (I \times X, Y, Q, R) = ((I, X \Rightarrow Y), P, (\text{deferred to } (14)))$$

where the last component is

$$\{\langle i, f \rangle : 1 \to I \times (X \Rightarrow Y) \mid i \in P \wedge$$
$$\forall x : 1 \to X, \langle i, x \rangle \in Q \implies \langle \langle i, x \rangle, \mathbf{ev} \circ \langle f, x \rangle \rangle \in R\} \tag{14}$$

- Coproduct

$$\dot{\coprod}_{((I,X),P,Q)} (I \times X, Y, Q, R) = ((I, X \times Y), P, (\alpha^{-1})^*R)$$

where $(\alpha^{-1})^*R = \{\langle i, \langle x, y \rangle \rangle : 1 \to I \times (X \times Y) \mid \langle \langle i, x \rangle, y \rangle \in R\}$

- Fibred binary coproduct

$$((I, X), P, Q) \dot{+} ((I, Y), P, R)$$
$$= ((I, X + Y), P, \{(\mathrm{id}_I \times \iota_1) \circ f \mid f \in Q\} \cup \{(\mathrm{id}_I \times \iota_2) \circ g \mid g \in R\})$$

- Fibred initial object

$$\dot{0}(I, P) = ((I, 0), P, \emptyset)$$

Note that morphisms in $\{s(\mathbb{C}) \mid \mathbb{P}\}$ and $\mathbb{P}$ are denoted by $(u, f) : ((I, X), P, Q) \dot{\to} ((J, Y), R, S)$ and $u : (I, P) \dot{\to} (J, R)$, which indicate that they are unique morphisms over $(u, f) : (I, X) \dot{\to} (J, Y)$ in $s(\mathbb{C})$ and $u : I \dot{\to} J$ in $\mathbb{C}$, respectively.

Definition B.4 (interpretation of well-formed contexts/types). A well-formed context $\vdash \dot{\Gamma}$ is interpreted as $\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \in \mathbb{P}_{\mathcal{A}[\![|\dot{\Gamma}|]\!]}$.

$$[\![\cdot]\!] := \top 1 \in \mathbb{P}_1 \qquad [\![\dot{\Gamma}, x : \dot{\sigma}]\!] := \{[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]\} \in \mathbb{P}_{\mathcal{A}[\![|\dot{\Gamma}|]\!] \times \mathcal{A}[\![|\dot{\sigma}|]\!]}$$

A well-formed type $\dot{\Gamma} \vdash \dot{\sigma}$ is interpreted as $\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] \in \{s(\mathbb{C}) \mid \mathbb{P}\}_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]}$ such that $u\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] = (\mathcal{A}[\![|\dot{\Gamma}|]\!], \mathcal{A}[\![|\dot{\sigma}|]\!])$.

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \{v : \sigma \mid \phi\}]\!] := ((\mathcal{A}[\![|\dot{\Gamma}|]\!], \mathcal{A}[\![\sigma]\!]), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!], \pi_1^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, v : \sigma \vdash \phi]\!])$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}]\!] := \dot{\prod}_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \{v : \mathbf{Prop} \mid \phi\}]\!]$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \times \dot{\tau}]\!] := \dot{\bigsqcup}_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash 0]\!] := \dot{0}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \qquad \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma} + \dot{\tau}]\!] := \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] \dot{+} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]$$

B.4 Subtyping relation

We define $\dot{\Gamma} \mid v : \sigma.\phi \vDash \psi$ by

$$\pi_1^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, v : \sigma \vdash \phi]\!] \leq \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, v : \sigma \vdash \psi]\!].$$

Intuitively, this means that if formulas in $\dot{\Gamma}$ and ϕ are true, then ψ is also true.

We define a subtyping relation $\dot{\Gamma} \vdash \dot{\sigma} <: \dot{\tau}$.

SUB-REFL
$$\frac{\dot{\Gamma} \vdash \dot{\sigma}}{\dot{\Gamma} \vdash \dot{\sigma} <: \dot{\sigma}}$$

SUB-TRANS
$$\frac{\dot{\Gamma} \vdash \dot{\sigma}_1 <: \dot{\sigma}_2 \qquad \dot{\Gamma} \vdash \dot{\sigma}_2 <: \dot{\sigma}_3}{\dot{\Gamma} \vdash \dot{\sigma}_1 <: \dot{\sigma}_3}$$

SUB-REFINE
$$\frac{\dot{\Gamma} \vdash v : \sigma.\phi \vDash \psi \qquad \sigma \in \mathbf{Base} \cup \{\mathbf{Prop}, 1\}}{\dot{\Gamma} \vdash \{v : \sigma \mid \phi\} <: \{v : \sigma \mid \psi\}}$$

SUB-DFUN
$$\frac{\dot{\Gamma} \vdash \dot{\sigma}_2 <: \dot{\sigma}_1 \qquad \dot{\Gamma}, x : \dot{\sigma}_1 \vdash \tau_1 \qquad \dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_1 <: \dot{\tau}_2}{\dot{\Gamma} \vdash (x : \dot{\sigma}_1) \to \dot{\tau}_1 <: (x : \dot{\sigma}_2) \to \dot{\tau}_2}$$

SUB-DPAIR
$$\frac{\dot{\Gamma} \vdash \dot{\sigma}_1 <: \dot{\sigma}_2 \qquad \dot{\Gamma}, x : \dot{\sigma}_2 \vdash \tau_2 \qquad \dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_1 <: \dot{\tau}_2}{\dot{\Gamma} \vdash (x : \dot{\sigma}_1) \times \dot{\tau}_1 <: (x : \dot{\sigma}_2) \times \dot{\tau}_2}$$

SUB-SUM
$$\frac{\dot{\Gamma} \vdash \dot{\sigma}_1 <: \dot{\sigma}_2 \qquad \dot{\Gamma} \vdash \dot{\tau}_1 <: \dot{\tau}_2}{\dot{\Gamma} \vdash \dot{\sigma}_1 + \dot{\tau}_1 <: \dot{\sigma}_2 + \dot{\tau}_2}$$

B.5 Typing rules

A well-typed term $\dot\Gamma \vdash M : \dot\sigma$ is defined as follows.

R-Sub
$$\frac{\dot\Gamma \vdash M : \dot\sigma \qquad \dot\Gamma \vdash \dot\sigma <: \dot\tau}{\dot\Gamma \vdash M : \dot\tau}$$

R-App
$$\frac{\dot\Gamma \vdash M : (x : \dot\sigma) \to \{v : \textbf{Prop} \mid \phi\} \qquad \dot\Gamma \vdash N : \dot\sigma}{\dot\Gamma \vdash M\,N : \{v : \textbf{Prop} \mid \phi[N/x]\}}$$

R-Abs
$$\frac{\dot\Gamma, x : \dot\sigma \vdash M : \{v : \textbf{Prop} \mid \phi\}}{\dot\Gamma \vdash \lambda x : \dot\sigma.M : (x : \dot\sigma) \to \{v : \textbf{Prop} \mid \phi\}}$$

R-Unit
$$\frac{\vdash \dot\Gamma}{\dot\Gamma \vdash () : \{v : 1 \mid \top\}}$$

R-Pair
$$\frac{\dot\Gamma \vdash M : \dot\sigma \qquad \dot\Gamma \vdash N : \dot\tau[M/x]}{\dot\Gamma \vdash (M, N) : (x : \dot\sigma) \times \dot\tau}$$

R-Fst
$$\frac{\dot\Gamma \vdash M : (x : \dot\sigma) \times \dot\tau}{\dot\Gamma \vdash \pi_1\,M : \dot\sigma}$$

R-Snd
$$\frac{\dot\Gamma \vdash M : (x : \dot\sigma) \times \dot\tau}{\dot\Gamma \vdash \pi_2\,M : \dot\tau[\pi_1\,M/x]}$$

R-Case0
$$\frac{\dot\Gamma \vdash M : 0 \qquad \dot\Gamma \vdash \dot\tau}{\dot\Gamma \vdash \delta(M) : \dot\tau}$$

R-Inj
$$\frac{\dot\Gamma \vdash M : \dot\sigma_i \qquad \dot\Gamma \vdash \dot\sigma_{3-i}}{\dot\Gamma \vdash \iota_i\,M : \dot\sigma_1 + \dot\sigma_2}$$

R-Case2
$$\frac{\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2 \vdash \dot\tau \qquad \dot\Gamma \vdash M : \dot\sigma_1 + \dot\sigma_2 \qquad \dot\Gamma, x_1 : \dot\sigma_1 \vdash N_1 : \dot\tau[\iota_1\,x_1/z] \qquad \dot\Gamma, x_2 : \dot\sigma_2 \vdash N_2 : \dot\tau[\iota_2\,x_2/z]}{\dot\Gamma \vdash \delta(M, x_1.N_1, x_2.N_2) : \dot\tau[M/z]}$$

R-VarRefine
$$\frac{\vdash \dot\Gamma \qquad (x : \{v : b \mid \phi\}) \in \dot\Gamma}{\dot\Gamma \vdash x : \{v : b \mid v = x\}}$$

R-Var
$$\frac{\vdash \dot\Gamma \qquad (x : \dot\sigma) \in \dot\Gamma}{\dot\Gamma \vdash x : \dot\sigma}$$

R-Fix
$$\frac{\dot\Gamma, f : (x : \dot\sigma) \to \{v : \textbf{Prop} \mid \phi\} \vdash M : (x : \dot\sigma) \to \{v : \textbf{Prop} \mid \phi\} \qquad |\dot\Gamma|, x : |\dot\sigma|, v : \textbf{Prop} \vdash \phi \qquad \phi \text{ is admissible at } v}{\dot\Gamma \vdash \textbf{fix}\,f.M : (x : \dot\sigma) \to \{v : \textbf{Prop} \mid \phi\}}$$

R-BasicOp
$$\frac{\dot\Gamma \vdash \dot\tau \qquad \mathrm{ar}(\textbf{op}) = |\dot\sigma| \qquad \mathrm{car}(\textbf{op}) = |\dot\tau| \qquad \dot\Gamma \vdash M : \dot\sigma \qquad (\mathrm{id} \times a(\textbf{op})) : \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \dot\sigma]\!] \dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \dot\tau]\!]}{\dot\Gamma \vdash \textbf{op}(M) : \dot\tau}$$

R-BasicConst

$$\frac{\mathbf{op} : 1 \to \tau \qquad \tau \in \mathbf{Base} \cup \{\mathbf{Prop}\}}{\dot{\Gamma} \vdash \mathbf{op} : \{v : \tau \mid v =_\tau \mathbf{op}\}}$$

R-BasicSimple

$$\frac{\begin{array}{cc} & \mathbf{op} : \sigma_1 \times \cdots \times \sigma_n \to \tau \\ \sigma_1, \ldots, \sigma_n, \tau \in \mathbf{Base} \cup \{\mathbf{Prop}\} & \dot{\Gamma} \vdash M : \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \phi[\mathbf{op}(v_1, \ldots, v_n)/v]\} \end{array}}{\dot{\Gamma} \vdash \mathbf{op}(M) : \{v : \tau \mid \phi\}}$$

R-BasicBool

$$\frac{\begin{array}{c} \mathbf{op} : \sigma_1 \times \cdots \times \sigma_n \to 1 + 1 \qquad \sigma_1, \ldots, \sigma_n \in \mathbf{Base} \cup \{\mathbf{Prop}\} \\ v_1 : \sigma_1, \ldots, v_n : \sigma_n \vdash \psi_t \qquad \mathcal{A}^{\mathcal{P}}[\![(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \psi_t]\!] \subseteq \pi_2^* a(\mathbf{op})^* \{\iota_1\} \\ v_1 : \sigma_1, \ldots, v_n : \sigma_n \vdash \psi_f \qquad \mathcal{A}^{\mathcal{P}}[\![(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \psi_f]\!] \subseteq \pi_2^* a(\mathbf{op})^* \{\iota_2\} \\ \dot{\Gamma} \vdash M : \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \psi_t \wedge \phi_t[()/v] \vee \psi_f \wedge \phi_f[()/v]\} \end{array}}{\dot{\Gamma} \vdash \mathbf{op}(M) : \{v : 1 \mid \phi_t\} + \{v : 1 \mid \phi_f\}}$$

C   DETAILS OF PROOFS

C.1   Preparation

Definition C.1. Let Γ_1, Γ_2 be contexts and σ be a type. We define $\mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2} : \mathcal{A}[\![\Gamma_1, x : \sigma, \Gamma_2]\!] \to \mathcal{A}[\![\Gamma_1, \Gamma_2]\!]$ as follows.

$$\mathrm{proj}_{\Gamma_1;\sigma;\cdot} := \pi_1 \qquad \mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2,y:\tau} := \mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2} \times \mathrm{id}$$

Lemma C.2 (weakening for terms). *For any well-typed term $\Gamma_1, \Gamma_2 \vdash M : \tau$ and type σ,*

$$\mathcal{A}[\![\Gamma_1, \Gamma_2 \vdash M : \tau]\!] \circ \mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2} = \mathcal{A}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash M : \tau]\!]$$

Lemma C.3 (weakening for formulas). *For any well-formed $\Gamma_1, \Gamma_2 \vdash \phi$ and type σ,*

$$\mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2}^* \mathcal{A}^{\mathcal{P}}[\![\Gamma_1, \Gamma_2 \vdash \phi]\!] = \mathcal{A}^{\mathcal{P}}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash \phi]\!]$$

Proof. By induction.

- For basic predicates including $\leq_{\mathbf{Prop}}$ and $=_b$,

$$\begin{aligned} \mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2}^* \mathcal{A}^{\mathcal{P}}[\![\Gamma_1, \Gamma_2 \vdash a(M)]\!] &= \mathrm{proj}_{\Gamma_1;\sigma;\Gamma_2}^* (\mathcal{A}[\![\Gamma_1, \Gamma_2 \vdash M : \tau]\!])^* \mathcal{P}(a) \\ &= (\mathcal{A}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash M : \tau]\!])^* \mathcal{P}(a) \\ &= \mathcal{A}^{\mathcal{P}}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash a(M)]\!] \end{aligned}$$

  by Lemma C.2.
- For $\bot, \top, \vee, \wedge, \implies$, we use the fact that fibred bi-cc structures are preserved by reindexing.

$\square$

Lemma C.4 (weakening for types). *Let $\dot{\Gamma}_1 \vdash \dot{\sigma}$ be a well-formed type.*

- *For any well-formed context $\vdash \dot{\Gamma}_1, \dot{\Gamma}_2$, we have*

$$\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|} : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2]\!].$$

- *For any well-formed type $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash \dot{\tau}$, we have*

$$\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|}^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash \dot{\tau}]\!] = \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}]\!].$$

Proof. By simultaneous induction on $\vdash \dot{\Gamma}_1, \dot{\Gamma}_2$ and $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash \dot{\tau}$.

- If $\dot{\Gamma}_2 = \cdot$, we have

$$\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;\cdot} : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1]\!]$$

because $\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;\cdot} = \pi_1 = \pi_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1 \vdash \dot{\sigma}]\!]}$.

- If $\dot{\Gamma}_2 = \dot{\Gamma}_2', y : \dot{\sigma}'$, we have

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2', y : \dot{\sigma}']\!]$$
$$\|$$
$$\{\mathrm{proj}^*_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2' \vdash \dot{\sigma}']\!]\}$$
$$\downarrow \{\overline{\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2'|}}(\dots)\}$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2', y : \dot{\sigma}']\!]$$

$$\{\overline{\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|}}(\dots)\} = \mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|} \times \mathrm{id}$$
$$= \mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|, y:|\dot{\sigma}'|}$$

- $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash \{v : \tau \mid \phi\}$: By definition of the interpretation,

$$\mathrm{proj}^*_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash \{v : \tau \mid \phi\}]\!]$$
$$= \mathrm{proj}^*_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|}((\mathcal{A}[\![|\dot{\Gamma}_1, \dot{\Gamma}_2|]\!], \mathcal{A}[\![\tau]\!]), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2]\!], \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1, \dot{\Gamma}_2|, v : \tau \vdash \phi]\!])$$
$$= ((\mathcal{A}[\![|\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2|]\!], \mathcal{A}[\![\tau]\!]), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!],$$
$$\pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \wedge (\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|} \times \mathrm{id})^* \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1, \dot{\Gamma}_2|, v : \tau \vdash \phi]\!])$$

and by Lemma C.3,

$$(\mathrm{proj}_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|} \times \mathrm{id})^* \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1, \dot{\Gamma}_2|, v : \tau \vdash \phi]\!]$$
$$= \mathrm{proj}^*_{|\dot{\Gamma}_1|;|\dot{\sigma}|;|\dot{\Gamma}_2|, v:\tau} \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1, \dot{\Gamma}_2|, v : \tau \vdash \phi]\!]$$
$$= \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2|, v : \tau \vdash \phi]\!]$$

- $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash (x : \dot{\tau}_1) \to \{v : \tau \mid \phi\}$ and $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash (x : \dot{\tau}_1) \times \dot{\tau}_2$: By the BC condition.
- $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash 0$ and $\dot{\Gamma}_1, \dot{\Gamma}_2 \vdash \dot{\tau}_1 + \dot{\tau}_2$: Because reindexing functors preserve fibred coproducts.

□

Definition C.5. Let Γ_1, Γ_2 be contexts and $\Gamma_1 \vdash M : \sigma$ be a well-typed term. We define $\mathrm{subst}_{M;\Gamma_2} : \mathcal{A}[\![\Gamma_1, \Gamma_2]\!] \to \mathcal{A}[\![\Gamma_1, x : \sigma, \Gamma_2]\!]$ as follows.

$$\mathrm{subst}_{M;\cdot} := \langle \mathrm{id}, \mathcal{A}[\![M]\!] \rangle \qquad \mathrm{subst}_{M;\Gamma_2, y:\tau} := \mathrm{subst}_{M;\Gamma_2} \times \mathrm{id}$$

LEMMA C.6. *For any well-typed terms* $\Gamma_1, x : \sigma, \Gamma_2 \vdash N$ *and* $\Gamma_1 \vdash M : \sigma$, *we have*

$$\mathcal{A}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash N]\!] \circ \mathrm{subst}_{M;\Gamma_2} = \mathcal{A}[\![\Gamma_1, \Gamma_2 \vdash N[M/x]]\!]$$

PROOF. By induction on N.   □

LEMMA C.7 (SUBSTITUTION IN FORMULAS). *For any well-formed* $\Gamma_1, x : \sigma, \Gamma_2 \vdash \phi$ *and* $\Gamma_1 \vdash M : \sigma$,

$$\mathcal{A}^{\mathcal{P}}[\![\Gamma_1, \Gamma_2 \vdash \phi[M/x]]\!] = \mathrm{subst}^*_{M;\Gamma_2} \mathcal{A}^{\mathcal{P}}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash \phi]\!]$$

PROOF. Similarly to Lemma C.3

- For basic predicates including $\leq_{\mathbf{Prop}}$ and $=_b$,

$$\mathrm{subst}^*_{M;\Gamma_2}\mathcal{A}^{\mathcal{P}}[\![\Gamma_1, x : \sigma, \Gamma_2 \vdash a(N)]\!]$$

$$= \mathrm{subst}^*_{M;\Gamma_2}(\mathcal{A}[\![\Gamma_1.x : \sigma, \Gamma_2 \vdash N : \tau]\!])^*\mathcal{P}(a)$$

$$= (\mathcal{A}[\![\Gamma_1, \Gamma_2 \vdash N[M/x] : \tau]\!])^*\mathcal{P}(a)$$

$$= \mathcal{A}^{\mathcal{P}}[\![\Gamma_1, \Gamma_2 \vdash a(N)[M/x]]\!]$$

  by Lemma C.2.
- For $\bot, \top, \vee, \wedge, \implies$, we use the fact that fibred bi-cc structures are preserved by reindexing.

$$\square$$

LEMMA C.8 (SUBSTITUTION IN TYPES). *For any well-formed type $\dot{\Gamma}_1 \vdash \dot{\sigma}$ and well-typed term* $|\dot{\Gamma}_1| \vdash M : |\dot{\sigma}|$, *if* $\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1]\!] \to \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}]\!]$ *in* $\mathbb{P}$, *then*

- *For any well-formed context* $\vdash \dot{\Gamma}_1, \dot{\Gamma}_2$,

$$\mathrm{subst}_{M;|\dot{\Gamma}_2|} : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \qquad in \ \mathbb{P}$$

- *For any well-formed type* $\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}$,

$$\mathrm{subst}^*_{M;|\dot{\Gamma}_2|}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}]\!] = \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash \dot{\tau}[M/x]]\!]$$

PROOF. By simultaneous induction on $\vdash \dot{\Gamma}_1, \dot{\Gamma}_2$ and $\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}$.

- If $\dot{\Gamma}_2 = \cdot$, then we have $\mathrm{subst}_{M;\cdot} : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}]\!]$ by assumption.
- If $\dot{\Gamma}_2 = \dot{\Gamma}'_2, y : \dot{\tau}$, then

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}'_2[M/x], y : \dot{\tau}[M/x]]\!]$$

$$\|$$

$$\{\mathrm{subst}^*_{M;|\dot{\Gamma}'_2|}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}'_2 \vdash \dot{\tau}]\!]\}$$

$$\downarrow \{\overline{\mathrm{subst}_{M;|\dot{\Gamma}'_2|}}(\dots)\}$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}'_2, y : \dot{\tau}]\!]$$

$$\{\overline{\mathrm{subst}_{M;|\dot{\Gamma}'_2|}}(\dots)\} = \mathrm{subst}_{M;|\dot{\Gamma}'_2|} \times \mathrm{id} = \mathrm{subst}_{M;|\dot{\Gamma}'_2,y:\dot{\tau}|}$$

- $\{v : \tau \mid \phi\}$: By the definition of reindexing in $\{s_{\mathbb{C}} \mid p\} : \{s(\mathbb{C}) \mid \mathbb{P}\} \to \mathbb{P}$.

$$\mathrm{subst}^*_{M;|\dot{\Gamma}_2|}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \{v : \tau \mid \phi\}]\!]$$

$$= \mathrm{subst}^*_{M;|\dot{\Gamma}_2|}((\mathcal{A}[\![|\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2|]\!], \mathcal{A}[\![\tau]\!]), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!],$$

$$\pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1|, x : |\dot{\sigma}|, |\dot{\Gamma}_2|, v : \tau \vdash \phi]\!])$$

$$= ((\mathcal{A}[\![|\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]|]\!], \mathcal{A}[\![\tau]\!]), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!],$$

$$\pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!] \wedge (\mathrm{subst}_{M;|\dot{\Gamma}_2|} \times \mathrm{id})^*(\pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1|, x : |\dot{\sigma}|, |\dot{\Gamma}_2|, v : \tau \vdash \phi]\!]))$$

$$\pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!] \wedge (\mathrm{subst}_{M;|\dot{\Gamma}_2|} \times \mathrm{id})^*(\pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1|, x : |\dot{\sigma}|, |\dot{\Gamma}_2|, v : \tau \vdash \phi]\!])$$

$$= \pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!] \wedge \pi_1^*\mathrm{subst}^*_{M;|\dot{\Gamma}_2|}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!] \wedge \mathrm{subst}^*_{M;|\dot{\Gamma}_2|,v:\tau}\mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1|, x : |\dot{\sigma}|, |\dot{\Gamma}_2|, v : \tau \vdash \phi]\!]$$

$$= \pi_1^*\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}_1|, |\dot{\Gamma}_2|, v : \tau \vdash \phi[M/x]]\!]$$

  Here, we used Lemma C.7 and $|\dot{\Gamma}_2[M/x]| = |\dot{\Gamma}_2|$.

- $(y : \dot{\tau}) \rightarrow \{v : \mathbf{Prop} \mid \phi\}$: Apply the BC condition.

$$
\begin{array}{ccc}
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x], y : \dot{\tau}[M/x]]\!] & \xrightarrow{\mathrm{subst}_{M;|\dot{\Gamma}_2|,y:|\dot{\tau}|}} & \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2, y : \dot{\tau}]\!] \\
\downarrow \pi & \lrcorner & \downarrow \pi \\
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!] & \xrightarrow{\mathrm{subst}_{M;|\dot{\Gamma}_2|}} & \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!]
\end{array}
$$

$$
\mathrm{subst}^*_{M;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash (y : \dot{\tau}) \rightarrow \{v : \mathbf{Prop} \mid \phi\}]\!]
$$

$$
= \mathrm{subst}^*_{M;|\dot{\Gamma}_2|} \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x:\dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2, y : \dot{\tau} \vdash \{v : \mathbf{Prop} \mid \phi\}]\!]
$$

$$
= \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash \dot{\tau}[M/x]]\!]} \mathrm{subst}^*_{M;|\dot{\Gamma}_2|,y:|\dot{\tau}|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2, y : \dot{\tau} \vdash \{v : \mathbf{Prop} \mid \phi\}]\!]
$$

$$
= \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash \dot{\tau}[M/x]]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x], y : \dot{\tau}[M/x] \vdash \{v : \mathbf{Prop} \mid \phi[M/x]\}]\!]
$$

$$
= \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash (y : \dot{\tau}[M/x]) \rightarrow \{v : \mathbf{Prop} \mid \phi[M/x]\}]\!]
$$

- $(y : \dot{\tau}_1) \times \dot{\tau}_2$: Apply the BC condition (almost the same as above).
- 0: Fibred initial objects are preserved by reindexing functors.

$$
\mathrm{subst}^*_{M;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash 0]\!] = \mathrm{subst}^*_{M;|\dot{\Gamma}_2|} 0 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2]\!]
$$

$$
= 0 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x]]\!]
$$

$$
= \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash 0]\!]
$$

- $\dot{\tau}_1 + \dot{\tau}_2$: Fibred binary coproducts are preserved by reindexing functors.

$$
\mathrm{subst}^*_{M;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}_1 + \dot{\tau}_2]\!]
$$

$$
= \mathrm{subst}^*_{M;|\dot{\Gamma}_2|} (\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}_1]\!] + \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}_2]\!])
$$

$$
= \mathrm{subst}^*_{M;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}_1]\!] + \mathrm{subst}^*_{M;|\dot{\Gamma}_2|} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, x : \dot{\sigma}, \dot{\Gamma}_2 \vdash \dot{\tau}_2]\!]
$$

$$
= \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash \dot{\tau}_1[M/x]]\!] + \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash \dot{\tau}_2[M/x]]\!]
$$

$$
= \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}_1, \dot{\Gamma}_2[M/x] \vdash (\dot{\tau}_1 + \dot{\tau}_2)[M/x]]\!]
$$

$\square$

C.2 Proofs for Subtyping Rules

PROPOSITION C.9 (SOUNDNESS OF SUBTYPING). *For any $\dot{\Gamma} \vdash \dot{\sigma} <: \dot{\tau}$, we have*

$$
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] \leq \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]
$$

where we define $((I, X), P, Q) \leq ((I, X), P, Q')$ if $Q \subseteq Q'$, or equivalently, $\mathrm{id}_{(I,X)} : ((I, X), P, Q) \rightarrow ((I, X), P, Q')$.

PROOF. By induction on derivation of $\dot{\Gamma} \vdash \dot{\sigma} <: \dot{\tau}$.

- Sub-Refl, Sub-Trans: By reflexivity and transitivity of $\subseteq$.
- Sub-Refine: Obvious from the definition of the interpretation.
- Sub-Sum: Easy.

- Sub-DFun: By IH, we have

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}_2]\!] \leq \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}_1]\!]$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_1]\!] \leq \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_2]\!]$$

We apply the BC condition.

$$
\begin{array}{ccc}
\{s(\mathbb{C}) \mid \mathbb{P}\}_{\{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_1]\!]\}} & \xrightarrow{\ \Pi\ } & \{s(\mathbb{C}) \mid \mathbb{P}\}_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]} \\
{\scriptstyle\{\mathrm{id}\}^*}\downarrow & & \parallel \\
\{s(\mathbb{C}) \mid \mathbb{P}\}_{\{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_2]\!]\}} & \xrightarrow{\ \Pi\ } & \{s(\mathbb{C}) \mid \mathbb{P}\}_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]}
\end{array}
$$

$$
\begin{aligned}
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}_1) \to \dot{\tau}_1]\!] &= \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_1]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_1]\!] \\
&= \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_2]\!]} \{\mathrm{id}\}^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_1]\!] \\
&= \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_2]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_1]\!] \\
&\leq \prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_2]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_2]\!] \\
&= \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}_2) \to \dot{\tau}_2]\!]
\end{aligned}
$$

- Sub-DPair: By IH, we have

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}_1]\!] \leq \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}_2]\!]$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_1]\!] \leq \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_2]\!]$$

We apply the BC condition.

$$
\begin{aligned}
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}_1) \times \dot{\tau}_1]\!] &= \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_1]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_1]\!] \\
&\leq \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_1]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_1 \vdash \dot{\tau}_2]\!] \\
&= \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_1]\!]} \{\mathrm{id}\}^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_2]\!] \\
&= \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_2]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}_2 \vdash \dot{\tau}_2]\!] \\
&= \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}_2) \times \dot{\tau}_2]\!]
\end{aligned}
$$

$\square$

C.3 Proofs for Typing Rules

THEOREM C.10 (SOUNDNESS). *For any well-typed term $\dot{\Gamma} \vdash M : \dot{\sigma}$,*

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$

Note that this is equivalent to the following.

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!] \rangle : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]\}$$

PROOF. By induction on derivation of $\dot{\Gamma} \vdash M : \dot{\sigma}$. The basic idea of the proof is to interpret terms in $\{s_\mathbb{C} \mid p\} : \{s(\mathbb{C}) \mid \mathbb{P}\} \to \mathbb{P}$ as terms of a dependent type system. Proofs for individual cases are given below.

- R-Sub: Lemma C.11
- R-App: Lemma C.12
- R-Abs: Lemma C.13
- R-Unit: Lemma C.14
- R-Pair: Lemma C.15
- R-Fst: Lemma C.16
- R-Snd: Lemma C.17
- R-Case0: Lemma C.18
- R-Inj: Lemma C.19
- R-Case2: Lemma C.20
- R-Var: Lemma C.21
- R-VarRefine: Lemma C.22
- R-Fix: Lemma C.29
- R-BasicOp: Lemma C.30

$\square$

LEMMA C.11. *R-Sub is sound.*

PROOF. By IH and Proposition C.9, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] \le \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!].$$

Therefore, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!].$$

$\square$

LEMMA C.12. *R-App is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \to \{v : \mathbf{Prop} \mid \phi\}]\!]$$
$$(\mathrm{id}, \mathcal{A}[\![N]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$

We consider the following vertical morphism in $\{s(\mathbb{C}) \mid \mathbb{P}\}$.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\downarrow{\scriptstyle(\mathrm{id},\mathcal{A}[\![M]\!]\circ\pi_1)}$$

$$\prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}\vdash\{v:\mathbf{Prop}\mid\phi\}]\!]$$

$$\|$$

$$\langle\mathrm{id},\mathcal{A}[\![N]\!]\rangle^{*}\pi_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}^{*}\prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}\vdash\{v:\mathbf{Prop}\mid\phi\}]\!]$$

$$\downarrow{\scriptstyle\langle\mathrm{id},\mathcal{A}[\![N]\!]\rangle^{*}\epsilon^{\pi^{*}\dashv\Pi}}$$

$$\langle\mathrm{id},\mathcal{A}[\![N]\!]\rangle^{*}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}\vdash\{v:\mathbf{Prop}\mid\phi\}]\!]$$

$$\|{\scriptstyle\text{by Lemma C.8}}$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\{v:\mathbf{Prop}\mid\phi[N/x]\}]\!]$$

The composite is equal to $(\mathrm{id},\mathcal{A}[\![M\ N]\!]\circ\pi_1):1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]\to\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\{v:\mathbf{Prop}\mid\phi[N/x]\}]\!]$.

$$\langle\mathrm{id},\mathcal{A}[\![N]\!]\rangle^{*}(\mathrm{id},\mathbf{ev}\circ\sigma\circ(\pi_2\times\mathrm{id}))\circ(\mathrm{id},\mathcal{A}[\![M]\!]\circ\pi_1)$$

$$=(\mathrm{id},\mathbf{ev}\circ\sigma\circ(\pi_2\times\mathrm{id})\circ(\langle\mathrm{id},\mathcal{A}[\![N]\!]\rangle\times\mathrm{id})\circ\langle\pi_1,\mathcal{A}[\![M]\!]\circ\pi_1\rangle)$$

$$=(\mathrm{id},\mathbf{ev}\circ\langle\mathcal{A}[\![M]\!],\mathcal{A}[\![N]\!]\rangle\circ\pi_1)$$

$$=(\mathrm{id},\mathcal{A}[\![M\ N]\!]\circ\pi_1)$$

$\square$

LEMMA C.13. *R-ABS is sound.*

PROOF. By IH, we have

$$(\mathrm{id},\mathcal{A}[\![M]\!]\circ\pi_1):1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}]\!]\dot{\to}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}\vdash\{v:\mathbf{Prop}\mid\phi\}]\!]$$

We consider the following vertical morphism in $\{s(\mathbb{C})\mid\mathbb{P}\}$.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\downarrow{\scriptstyle\eta^{\pi^{*}\dashv\Pi}}$$

$$\prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}\pi_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}^{*}1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\|$$

$$\prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}]\!]$$

$$\downarrow{\scriptstyle\prod(\mathrm{id},\mathcal{A}[\![M]\!]\circ\pi_1)}$$

$$\prod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]}\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma},x:\dot{\sigma}\vdash\{v:\mathbf{Prop}\mid\phi\}]\!]$$

$$\|$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash(x:\dot{\sigma})\to\{v:\mathbf{Prop}\mid\phi\}]\!]$$

The composite is equal to $(\mathrm{id}, \mathcal{A}[\![\lambda x.M]\!] \circ \pi_1)$.

$$\prod (\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) \circ \eta^{\pi^* \dashv \prod}$$
$$= (\mathrm{id}, \Lambda(\mathcal{A}[\![M]\!] \circ \pi_1 \circ \langle \pi_1 \times \mathrm{id}, \mathbf{ev} \circ (\pi_2 \times \mathrm{id})\rangle)) \circ \eta^{\pi^* \dashv \prod}$$
$$= (\mathrm{id}, \Lambda(\mathcal{A}[\![M]\!]) \circ \pi_1) \circ (\mathrm{id}, \dots)$$
$$= (\mathrm{id}, \Lambda(\mathcal{A}[\![M]\!]) \circ \pi_1)$$
$$= (\mathrm{id}, \mathcal{A}[\![\lambda x.M]\!] \circ \pi_1)$$

$\square$

LEMMA C.14. *R-UNIT is sound.*

PROOF. Note that we have

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \{v : 1 \mid \top\}]\!] = 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

Thus,

$$(\mathrm{id}, ! \circ \pi_1) = ! : 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \to 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$\square$

LEMMA C.15. *R-PAIR is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$
$$(\mathrm{id}, \mathcal{A}[\![N]\!] \circ \pi_1) : 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}[M/x]]\!]$$

We consider the following vertical morphism in $\{s(\mathbb{C}) \mid \mathbb{P}\}$.

$$1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$
$$\Big\downarrow {\scriptstyle (\mathrm{id}, \mathcal{A}[\![N]\!] \circ \pi_1)}$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}[M/x]]\!]$$
$$\Big\| {\scriptstyle \text{by Lemma C.8}}$$
$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]$$
$$\Big\downarrow {\scriptstyle \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* \eta^{\coprod \dashv \pi^*}}$$
$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* \pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]} \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]$$
$$\Big\|$$
$$\coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]$$
$$\Big\|$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \times \dot{\tau}]\!]$$

The composite is equal to $(\mathrm{id}, \mathcal{A}[\![(M, N)]\!] \circ \pi_1)$.

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!] \rangle^* \eta^{\coprod \dashv \pi^*} \circ (\mathrm{id}, \mathcal{A}[\![N]\!] \circ \pi_1)$$
$$= (\mathrm{id}, (\pi_2 \times \mathrm{id}) \circ (\langle \mathrm{id}, \mathcal{A}[\![M]\!] \rangle \times \mathrm{id}) \circ \langle \pi_1, \mathcal{A}[\![N]\!] \circ \pi_1 \rangle)$$
$$= (\mathrm{id}, \langle \mathcal{A}[\![M]\!], \mathcal{A}[\![N]\!] \rangle \circ \pi_1)$$
$$= (\mathrm{id}, \mathcal{A}[\![(M, N)]\!] \circ \pi_1)$$

$\square$

LEMMA C.16. *R-Fst is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \xrightarrow{\cdot} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \times \dot{\tau}]\!]$$

We consider the following composite.

$$
\begin{array}{c}
1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \\
\Big\downarrow {\scriptstyle (\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1)} \\
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \times \dot{\tau}]\!] \\
\Big\| \\
\coprod \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!] \\
\Big\downarrow {\scriptstyle \mathbf{fst}} \\
\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]
\end{array}
$$

This is equal to $(\mathrm{id}, \mathcal{A}[\![\pi_1 \, M]\!] \circ \pi_1)$.

$$\mathbf{fst} \circ (\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) = (\mathrm{id}, \pi_1 \circ \pi_2 \circ \langle \pi_1, \mathcal{A}[\![M]\!] \circ \pi_1 \rangle)$$
$$= (\mathrm{id}, \pi_1 \circ \mathcal{A}[\![M]\!] \circ \pi_1)$$
$$= (\mathrm{id}, \mathcal{A}[\![\pi_1 \, M]\!] \circ \pi_1)$$

$\square$

LEMMA C.17. *R-Snd is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1 \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \xrightarrow{\cdot} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \times \dot{\tau}]\!]$$

We consider the following composite.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\|$$

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}, y : \dot{\tau}]\!]$$

$$\Big\downarrow \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* \eta^{\sqcup \dashv \pi^*}$$

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* \pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]} \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]} 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}, y : \dot{\tau}]\!]$$

$$\Big\downarrow \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* \pi^* \mathbf{fst}$$

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* \pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]$$

$$\|$$

$$\langle \mathrm{id}, \mathcal{A}[\![\pi_1 \, M]\!]\rangle^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]$$

$$\Big\| \text{by Lemma C.8}$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}[\pi_1 \, M/x]]\!]$$

Here, we used the following equation.

$$\pi_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\tau}]\!]} \circ \kappa^{-1} \circ \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle = \pi_1 \circ \alpha^{-1} \circ \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle$$
$$= \langle \mathrm{id}, \pi_1 \circ \mathcal{A}[\![M]\!]\rangle$$
$$= \langle \mathrm{id}, \mathcal{A}[\![\pi_1 \, M]\!]\rangle$$

The composite above is equal to $(\mathrm{id}, \mathcal{A}[\![\pi_2 \, M]\!] \circ \pi_1)$.

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* \pi^* \mathbf{fst} \circ \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^*(\kappa^{-1})^* \eta^{\sqcup \dashv \pi^*}$$
$$= (\mathrm{id}, \pi_1 \circ \pi_2 \circ (\langle \mathrm{id}, \pi_1 \circ \mathcal{A}[\![M]\!]\rangle \times \mathrm{id})) \circ (\mathrm{id}, (\pi_2 \times \mathrm{id}) \circ ((\alpha^{-1} \circ \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle) \times \mathrm{id}))$$
$$= (\mathrm{id}, \pi_1 \circ \pi_2) \circ (\mathrm{id}, (\pi_2 \circ \mathcal{A}[\![M]\!]) \times \mathrm{id})$$
$$= (\mathrm{id}, \pi_1 \circ \pi_2 \circ \langle \pi_1, (\pi_2 \circ \mathcal{A}[\![M]\!]) \times \mathrm{id}\rangle)$$
$$= (\mathrm{id}, \pi_2 \circ \mathcal{A}[\![M]\!] \circ \pi_1)$$
$$= (\mathrm{id}, \mathcal{A}[\![\pi_2 \, M]\!] \circ \pi_1)$$

$$\square$$

LEMMA C.18. *R-Case0 is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} 0\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

Consider the following.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\Big\downarrow (\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1)$$

$$0\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\Big\downarrow ?$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]$$

This is equal to $(\mathrm{id}, \mathcal{A}[\![\delta(M)]\!] \circ \pi_1)$.

$$? \circ (\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) = (\mathrm{id}, ? \circ \pi_2 \circ \langle \pi_1, \mathcal{A}[\![M]\!] \circ \pi_1 \rangle)$$
$$= (\mathrm{id}, ? \circ \mathcal{A}[\![M]\!] \circ \pi_1)$$
$$= (\mathrm{id}, \mathcal{A}[\![\delta(M)]\!] \circ \pi_1)$$

$\square$

LEMMA C.19. *R-INJ is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma \vdash \dot\sigma_i]\!]$$

Consider the following.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!]$$
$$\downarrow_{(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1)}$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma \vdash \dot\sigma_i]\!]$$
$$\downarrow_{\iota_i}$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma \vdash \dot\sigma_1 + \dot\sigma_2]\!]$$

This is equal to $(\mathrm{id}, \mathcal{A}[\![\iota_i\, M]\!] \circ \pi_1)$.

$$\iota_i \circ (\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) = (\mathrm{id}, \iota_i \circ \pi_2 \circ \langle \pi_1, \mathcal{A}[\![M]\!] \circ \pi_1 \rangle)$$
$$= (\mathrm{id}, \iota_i \circ \mathcal{A}[\![M]\!] \circ \pi_1)$$
$$= (\mathrm{id}, \mathcal{A}[\![\iota_i\, M]\!] \circ \pi_1)$$

$\square$

LEMMA C.20. *R-CASE2 is sound.*

PROOF. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma \vdash \dot\sigma_1 + \dot\sigma_2]\!]$$
$$(\mathrm{id}, \mathcal{A}[\![N_1]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_1 : \dot\sigma_1]\!] \dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_1 : \dot\sigma_1 \vdash \dot\tau[\iota_1\, x_1/z]]\!]$$
$$(\mathrm{id}, \mathcal{A}[\![N_2]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_2 : \dot\sigma_2]\!] \dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_2 : \dot\sigma_2 \vdash \dot\tau[\iota_2\, x_2/z]]\!]$$

By Lemma C.4,C.8, we have the following for $i = 1, 2$.

$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_i : \dot\sigma_i \vdash \dot\tau[\iota_i\, x_i/z]]\!]$$
$$= \langle \mathrm{id}, \mathcal{A}[\![|\dot\Gamma|, x_i : |\dot\sigma_i| \vdash \iota_i\, x_i : |\dot\sigma_1| + |\dot\sigma_2|]\!] \rangle^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_i : \dot\sigma_i, z : \dot\sigma_1 + \dot\sigma_2 \vdash \dot\tau]\!]$$
$$= \langle \mathrm{id}, \iota_i \circ \pi_2 \rangle^* \mathrm{proj}_{|\dot\Gamma|;|\dot\sigma_i|;z:|\dot\sigma_i|+|\dot\sigma_2|} \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2 \vdash \dot\tau]\!]$$
$$= (\mathrm{id} \times \iota_i)^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2 \vdash \dot\tau]\!]$$

$$1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, x_1 : \dot\sigma_1]\!] = \{\iota_1\}^* 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2]\!] = (\mathrm{id} \times \iota_1)^* 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2]\!]$$

Since we have strong fibred coproducts,

$$(\mathrm{id}, [\mathcal{A}[\![N_1]\!] \circ \pi_1, \mathcal{A}[\![N_2]\!] \circ \pi_1] \circ [(\mathrm{id} \times \iota_1) \times \mathrm{id}, (\mathrm{id} \times \iota_2) \times \mathrm{id}]^{-1})$$
$$: 1\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2]\!] \dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, z : \dot\sigma_1 + \dot\sigma_2 \vdash \dot\tau]\!]$$

Now consider the following.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\|$$

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* \pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}_1+\dot{\sigma}_2]\!]} 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$

$$\|$$

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, z : \dot{\sigma}_1 + \dot{\sigma}_2]\!]$$

$$\Big\downarrow \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* (\mathrm{id}, \dots)$$

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, z : \dot{\sigma}_1 + \dot{\sigma}_2 \vdash \dot{\tau}]\!]$$

$$\Big\| \text{by Lemma C.8}$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}[M/z]]\!]$$

This is equal to $(\mathrm{id}, \mathcal{A}[\![\delta(M, x_1.N_1, x_2.N_2)]\!] \circ \pi_1)$.

$$\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle^* (\mathrm{id}, [\mathcal{A}[\![N_1]\!] \circ \pi_1, \mathcal{A}[\![N_2]\!] \circ \pi_1] \circ [(\mathrm{id} \times \iota_1) \times \mathrm{id}, (\mathrm{id} \times \iota_2) \times \mathrm{id}]^{-1})$$

$$= (\mathrm{id}, [\mathcal{A}[\![N_1]\!] \circ \pi_1, \mathcal{A}[\![N_2]\!] \circ \pi_1] \circ [(\mathrm{id} \times \iota_1) \times \mathrm{id}, (\mathrm{id} \times \iota_2) \times \mathrm{id}]^{-1} \circ (\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle \times \mathrm{id}))$$

$$= (\mathrm{id}, [\mathcal{A}[\![N_1]\!], \mathcal{A}[\![N_2]\!]] \circ [\mathrm{id} \times \iota_1, \mathrm{id} \times \iota_2]^{-1} \circ \pi_1 \circ (\langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle \times \mathrm{id}))$$

$$= (\mathrm{id}, [\mathcal{A}[\![N_1]\!], \mathcal{A}[\![N_2]\!]] \circ [\mathrm{id} \times \iota_1, \mathrm{id} \times \iota_2]^{-1} \circ \langle \mathrm{id}, \mathcal{A}[\![M]\!]\rangle \circ \pi_1)$$

$$= (\mathrm{id}, \mathcal{A}[\![\delta(M, x_1.N_1, x_2.N_2)]\!] \circ \pi_1)$$

$$\square$$

LEMMA C.21. *R-VAR is sound.*

PROOF. By induction on the context.

- If $\dot{\Gamma}, x : \dot{\sigma} \vdash x : \dot{\sigma}$, then we prove

$$(\mathrm{id}, \pi_2 \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\sigma}]\!].$$

Consider the following.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}]\!]$$

$$\Big\downarrow \eta^{\mathrm{II}\dashv\pi^*}$$

$$\pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]} \coprod_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]} 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma}]\!]$$

$$\Big\downarrow \pi^*\mathbf{fst}$$

$$\pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}\vdash\dot{\sigma}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$

$$\Big\| \text{by Lemma C.4}$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \dot{\sigma}]\!]$$

This is equal to $(\mathrm{id}, \pi_2 \circ \pi_1)$.

$$\pi^*\mathbf{fst} \circ \eta^{\mathrm{II}\dashv\pi^*} = (\mathrm{id}, \pi_1 \circ \pi_2 \circ (\pi_1 \times \mathrm{id})) \circ (\mathrm{id}, \pi_2 \times \mathrm{id})$$

$$= (\mathrm{id}, \pi_1 \circ \pi_2 \circ (\pi_1 \times \mathrm{id}) \circ \langle \pi_1, \pi_2 \times \mathrm{id}\rangle)$$

$$= (\mathrm{id}, \pi_2 \circ \pi_1)$$

- If $\dot{\Gamma}, y : \dot{\tau} \vdash x : \dot{\sigma}$ and $(x : \dot{\sigma}) \in \dot{\Gamma}$, then we prove

$$(\mathrm{id}, \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, y : |\dot{\tau}| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, y : \dot{\tau}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, y : \dot{\tau} \vdash \dot{\sigma}]\!]$$

where IH is given as follows.

$$(\mathrm{id}, [\![|\dot{\Gamma}| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$

Consider the following.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, y : \dot{\tau}]\!]$$
$$\|$$
$$\pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]} 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!]$$
$$\Big\downarrow \pi^*(\mathrm{id}, [\![|\dot{\Gamma}| \vdash x:|\dot{\sigma}|]\!] \circ \pi_1)$$
$$\pi^*_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]$$
$$\Big\| \text{by Lemma C.4}$$
$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, y : \dot{\tau} \vdash \dot{\sigma}]\!]$$

This is equal to $(\mathrm{id}, \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, y : |\dot{\tau}| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1)$.

$$\pi^*(\mathrm{id}, [\![|\dot{\Gamma}| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1)$$
$$= (\mathrm{id}, [\![|\dot{\Gamma}| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1 \circ (\pi_1 \times \mathrm{id}))$$
$$= (\mathrm{id}, [\![|\dot{\Gamma}| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1 \circ \pi_1)$$
$$= (\mathrm{id}, [\![|\dot{\Gamma}, y : |\dot{\tau}|| \vdash x : |\dot{\sigma}|]\!] \circ \pi_1)$$

$\square$

LEMMA C.22. *R-VARREFINE is sound.*

PROOF. • If $\dot{\Gamma}, x : \{v : b \mid \phi\} \vdash x : \{v : b \mid v = x\}$, then we prove

$$(\mathrm{id}, \pi_2 \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!] \dot{\to} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\} \vdash \{v : b \mid v = x\}]\!].$$

Since

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\} \vdash \{v : b \mid v = x\}]\!]$$
$$= ((\ldots, \ldots), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!],$$
$$\pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, x : b, v : b \vdash v = x]\!])$$
$$= ((\ldots, \ldots), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!],$$
$$\pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!] \wedge \langle \pi_2, \pi_2 \circ \pi_1 \rangle^* \mathcal{P}(=))$$

it suffices to prove

$$\pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!] \le \langle \pi_1, \pi_2 \circ \pi_1 \rangle^* (\pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!] \wedge \langle \pi_2, \pi_2 \circ \pi_1 \rangle^* \mathcal{P}(=))$$

This follows from the following equations.

$$\langle \pi_1, \pi_2 \circ \pi_1 \rangle^* \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!] = \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \{v : b \mid \phi\}]\!]$$

$$\langle \pi_1, \pi_2 \circ \pi_1 \rangle^* \langle \pi_2, \pi_2 \circ \pi_1 \rangle^* \mathcal{P}(=) = \pi_1^* \pi_2^* \langle \mathrm{id}, \mathrm{id} \rangle^* \mathcal{P}(=)$$
$$= \pi_1^* \pi_2^* \top (Ab)$$
$$= \top \mathcal{A}[\![|\dot{\Gamma}|, x : b, v : b]\!]$$

- If $\dot{\Gamma}, y : \dot{\tau} \vdash x : \{v : b \mid v = x\}$, then the proof is the same as R-Var.

$\square$

Definition C.23. Let $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ be a context and $\gamma \in 1 \rightarrow \mathcal{A}[\![\Gamma]\!]$. For each $f : 1 \rightarrow \mathcal{A}[\![\sigma_i]\!]$, we define $\gamma[x_i \mapsto f] : 1 \rightarrow \mathcal{A}[\![\Gamma]\!]$ by

$$\gamma[x_i \mapsto f] := \langle \ldots \langle \mathrm{id}, \gamma_1 \rangle, \ldots, f, \ldots, \gamma_n \rangle$$

where $\gamma = \langle \ldots \langle \mathrm{id}, \gamma_1 \rangle, \ldots, \gamma_i, \ldots, \gamma_n \rangle$.

Definition C.24. Let $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ be a context. We say $Q \in \mathbb{P}_{\mathcal{A}[\![\Gamma]\!]}$ is *admissible at variable x_i* if

- $\mathbb{C}(1, \mathcal{A}[\![\sigma_i]\!])$ has a bottom element $\perp_{\mathcal{A}[\![\sigma_i]\!]}$
- for any $\gamma : 1 \rightarrow \mathcal{A}[\![\Gamma]\!]$, we have $\gamma[x_i \mapsto \perp_{\mathcal{A}[\![\sigma_i]\!]}] \in Q$
- for any $\gamma : 1 \rightarrow \mathcal{A}[\![\Gamma]\!]$ and any ω-chain $f_0 \leq f_1 \leq \ldots$ in $\mathbb{C}(1, \mathcal{A}[\![\sigma_i]\!])$, if $\gamma[x_i \mapsto f_k] \in Q$ holds for each k, then $\gamma[x_i \mapsto \sup_k f_k] \in Q$ (i.e. Q is chain-closed at x_i)

Definition C.25. $((I, X), P, Q) \in \{s(\mathbb{C}) \mid \mathbb{P}\}$ is *admissible* if

- X is an EM T-algebra (which implies that $\mathbb{C}(1, X)$ has a bottom element $\perp_X : 1 \rightarrow X$)
- for each $i \in P$, $\langle i, \perp_X \rangle \in Q$
- for any $i \in P$ and ω-chain $f_0 \leq f_1 \leq \ldots$ such that $\langle i, f_n \rangle \in Q$ for any n, we have $\langle i, \sup_n f_n \rangle \in Q$

LEMMA C.26. *If $\vdash \dot{\Gamma}$ and $\mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, v : \mathbf{Prop} \vdash \phi]\!]$ is admissible at v, then $\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \{v : \mathbf{Prop} \mid \phi\}]\!]$ is admissible.*

PROOF. By definition,

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \{v : \mathbf{Prop} \mid \phi\}]\!] = ((\mathcal{A}[\![|\dot{\Gamma}|]\!], \Omega), \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!], \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, v : \mathbf{Prop} \vdash \phi]\!])$$

is admissible. $\square$

LEMMA C.27. *Let $((I, X), P, Q) \in \{s(\mathbb{C}) \mid \mathbb{P}\}$. If $((I \times X, Y), Q, R) \in \{s(\mathbb{C}) \mid \mathbb{P}\}$ is admissible, then $\dot{\prod}_{((I,X),P,Q)}((I \times X, Y), Q, R)$ is admissible.*

PROOF. Note that we have $\perp_{X \Rightarrow Y} = \Lambda(\perp_Y \circ \pi_1)$.

$$\perp_{X \Rightarrow Y} = \Lambda(\nu_Y \circ T\mathbf{ev} \circ \theta'^T) \circ T?_{X \Rightarrow Y} \circ \perp_{T0}$$
$$= \Lambda(\nu_Y \circ T\mathbf{ev} \circ T(?_{X \Rightarrow Y} \times \mathrm{id}) \circ \theta'^T \circ (\perp_{T0} \times \mathrm{id}))$$
$$= \Lambda(\nu_Y \circ T\mathbf{ev} \circ T(\Lambda(?_Y \circ \pi_1) \times \mathrm{id}) \circ \theta'^T \circ (\perp_{T0} \times \mathrm{id}))$$
$$= \Lambda(\nu_Y \circ T?_Y \circ T\pi_1 \circ \theta'^T \circ (\perp_{T0} \times \mathrm{id}))$$
$$= \Lambda(\nu_Y \circ T?_Y \circ \pi_1 \circ (\perp_{T0} \times \mathrm{id}))$$
$$= \Lambda(\nu_Y \circ T?_Y \circ \perp_{T0} \circ \pi_1)$$
$$= \Lambda(\perp_Y \circ \pi_1)$$

So, (14) satisfies the second condition of Def. C.25 because for each $x : 1 \rightarrow X$, if $\langle i, x \rangle \in Q$, then

$$\langle \langle i, x \rangle, \mathbf{ev} \circ \langle \perp_{X \Rightarrow Y}, x \rangle \rangle = \langle \langle i, x \rangle, \perp_Y \circ \pi_1 \circ \langle \mathrm{id}, x \rangle \rangle = \langle \langle i, x \rangle, \perp_Y \rangle \in R.$$

It is routine to check that (14) satisfies the third condition of Def. C.25. $\square$

LEMMA C.28. *Let A be an EM T-algebra for a pseudo-lifting strong monad T with structure map $v : TA \rightarrow A$. If $((I, A), P, Q) \in \{s(\mathbb{C}) \mid \mathbb{P}\}$ is admissible and $(\mathrm{id}, f) : ((I, A), P, Q) \dashrightarrow ((I, A), P, Q)$, then $(\mathrm{id}, f^\dagger \circ \pi_1) : 1(I, P) \dashrightarrow ((I, A), P, Q)$*

PROOF. It suffices to show $\langle \mathrm{id}, f^\dagger \rangle : P \dashrightarrow Q$ (that is, $\langle i, f^\dagger \circ i \rangle \in Q$ for any $i \in P$) under the assumption that we have $\langle \pi_1, f \rangle : Q \dashrightarrow Q$.

Recall that $f^\dagger$ is defined by

$$f^\dagger := \Lambda^{-1}(v_X) \circ \langle \sup_n \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)} \circ \;!, \mathrm{id} \rangle$$

where

$$\lceil f \rceil := \eta^T \circ \Lambda(f \circ \langle \pi_2, \Lambda^{-1}(v_X) \rangle) : T(I \Rightarrow A) \rightarrow T(I \Rightarrow A)$$

$$v_X := \Lambda(v \circ T\mathbf{ev} \circ \theta'^T) : T(I \Rightarrow A) \rightarrow I \Rightarrow A$$

We show

$$\Lambda^{-1}(v_X) \circ \langle \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)} \circ \;!, \mathrm{id} \rangle \circ i = (f \circ \langle i \circ \;!, \mathrm{id} \rangle)^n \circ \bot_A$$

by induction on n.

- Base case: Note that we have $\bot_{T(I \Rightarrow A)} = T?_{I \Rightarrow A} \circ \bot_0$ and $?_{I \Rightarrow A} = \Lambda(?_A \circ \pi_1)$.

$$\begin{aligned}
\Lambda^{-1}(v_X) \circ \langle \bot_{T(I \Rightarrow A)} \circ \;!, \mathrm{id} \rangle \circ i &= v \circ T\mathbf{ev} \circ \theta'^T \circ \langle T?_{I \Rightarrow A} \circ \bot_0 \circ \;!, \mathrm{id} \rangle \circ i \\
&= v \circ T\mathbf{ev} \circ \theta'^T \circ \langle T\Lambda(?_A \circ \pi_1) \circ \bot_0, i \rangle \\
&= v \circ T(?_A \circ \pi_1) \circ \theta'^T \circ \langle \bot_0, i \rangle \\
&= v \circ T?_A \circ T\pi_2 \circ \theta^T \circ \langle i, \bot_0 \rangle \\
&= v \circ T?_A \circ \pi_2 \circ \langle i, \bot_0 \rangle \\
&= v \circ T?_A \circ \bot_0 \\
&= \bot_A
\end{aligned}$$

- Step case:

$$\begin{aligned}
&\Lambda^{-1}(v_X) \circ \langle \lceil f \rceil^{n+1} \circ \bot_{T(I \Rightarrow A)} \circ \;!, \mathrm{id} \rangle \circ i \\
&= v \circ T\mathbf{ev} \circ \theta'^T \circ (\lceil f \rceil \times \mathrm{id}) \circ \langle \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)}, i \rangle \\
&= v \circ T\mathbf{ev} \circ \eta^T \circ (\Lambda(f \circ \langle \pi_2, \Lambda^{-1}(v_X) \rangle) \times \mathrm{id}) \circ \langle \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)}, i \rangle \\
&= f \circ \langle \pi_2, \Lambda^{-1}(v_X) \rangle \circ \langle \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)}, i \rangle \\
&= f \circ \langle i, \Lambda^{-1}(v_X) \circ \langle \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)}, i \rangle \rangle \\
&= f \circ \langle i \circ \;!, \mathrm{id} \rangle \circ \Lambda^{-1}(v_X) \circ \langle \lceil f \rceil^n \circ \bot_{T(I \Rightarrow A)}, i \rangle
\end{aligned}$$

Now we prove $\langle i, f^\dagger \circ i \rangle = \langle i, \sup_n (f \circ \langle i \circ \;!, \mathrm{id} \rangle)^n \circ \bot_A \rangle \in Q$ using the admissibility of $((I, A), P, Q)$. It suffices to prove $\langle i, (f \circ \langle i \circ \;!, \mathrm{id} \rangle)^n \circ \bot_A \rangle \in Q$ for each n.

- We have $\langle i, \bot_A \rangle \in Q$ by the definition of the admissibility of $((I, A), P, Q)$.
- Let $g = f \circ \langle i \circ \;!, \mathrm{id} \rangle$. If $\langle i, g^n \circ \bot_A \rangle \in Q$, then

$$\begin{aligned}
\langle i, g^{n+1} \circ \bot_A \rangle &= \langle i, (f \circ \langle i \circ \;!, \mathrm{id} \rangle) \circ g^n \circ \bot_A \rangle \\
&= \langle i, f \circ \langle i, g^n \circ \bot_A \rangle \rangle \\
&= \langle \pi_1, f \rangle \circ \langle i, g^n \circ \bot_A \rangle
\end{aligned}$$

and since we have $\langle \pi_1, f \rangle : Q \dashrightarrow Q$, it follows that $\langle i, g^{n+1} \circ \bot_A \rangle \in Q$ holds.

<div style="text-align:right">□</div>

LEMMA C.29. *R-Fix is sound.*

PROOF. By Lemma C.26, $\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, x : \dot{\sigma} \vdash \{v : \mathbf{Prop} \mid \phi\}]\!]$ is admissible. By Lemma C.27, $\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \rightarrow \{v : \mathbf{Prop} \mid \phi\}]\!]$ is admissible. Let $X := \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \rightarrow \{v : \mathbf{Prop} \mid \phi\}]\!]$. By IH, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1) : 1\{X\} \dot{\rightarrow} \pi_X^* X$$

Since there is an isomorphism $\mathbb{E}_{\{X\}}(1\{X\}, \pi^* Y) \cong \mathbb{E}_I(X, Y)$ for any SCCompC $p : \mathbb{E} \rightarrow \mathbb{B}$ and $X, Y \in \mathbb{E}_I$, we have

$$(\mathrm{id}, \mathcal{A}[\![M]\!]) : X \dot{\rightarrow} X$$

By Lemma C.28, we have

$$(\mathrm{id}, \mathcal{A}[\![\mathbf{fix}\, f.M]\!] \circ \pi_1) = (\mathrm{id}, (\mathcal{A}[\![M]\!])^\dagger \circ \pi_1) : 1[\![\dot{\Gamma}]\!] \dot{\rightarrow} [\![\dot{\Gamma} \vdash (x : \dot{\sigma}) \rightarrow \{v : \mathbf{Prop} \mid \phi\}]\!]$$

$\square$

LEMMA C.30. *R-BasicOp is sound.*

PROOF. We have

$$(\mathrm{id}, \mathcal{A}[\![\mathbf{op}(M)]\!] \circ \pi_1) : 1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \dot{\rightarrow} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]$$

as the following composite.

$$1\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \xrightarrow{(\mathrm{id}, \mathcal{A}[\![M]\!] \circ \pi_1)} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] \xrightarrow{(\mathrm{id}, a(\mathbf{op}) \circ \pi_2)} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]$$

Here, the second morphism exists because we have a fully faithful functor $\{s(\mathbb{C}) \mid \mathbb{P}\} \rightarrow \mathbb{P}^{\rightarrow}$.

$$\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!] \xrightarrow{(\mathrm{id}, a(\mathbf{op}) \circ \pi_2)} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!] \qquad \mapsto \qquad \begin{array}{ccc} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, v : \dot{\sigma}]\!] & \xrightarrow{\mathrm{id} \times a(\mathbf{op})} & \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, v : \dot{\tau}]\!] \\ \downarrow{\scriptstyle \pi_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\sigma}]\!]}} & & \downarrow{\scriptstyle \pi_{\mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma} \vdash \dot{\tau}]\!]}} \\ \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] & \xrightarrow{\mathrm{id}} & \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}]\!] \end{array}$$

$\square$

C.4 Proofs for Specialised Rules for Basic Operators

COROLLARY C.31. *R-BasicConst is sound.*

PROOF. By Lemma C.30, it suffices to prove

$$(\mathrm{id} \times a(\mathbf{op})) : \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, v : 1]\!] \dot{\rightarrow} \mathcal{A}^{\mathcal{P}}[\![\dot{\Gamma}, v : \{v : \tau \mid v =_\tau \mathbf{op}()\}]\!]$$

This follows from the equation below.

$$\begin{aligned} & (\mathrm{id} \times a(\mathbf{op}))^* \mathcal{A}^{\mathcal{P}}[\![|\dot{\Gamma}|, v : \tau \vdash v =_\tau \mathbf{op}()]\!] \\ &= (\mathrm{id} \times a(\mathbf{op}))^* \langle \pi_2, a(\mathbf{op}) \circ\,! \rangle^* \mathcal{P}(=) \\ &= \langle a(\mathbf{op}) \circ\,!, a(\mathbf{op}) \circ\,! \rangle^* \mathcal{P}(=) \\ &= \top \mathcal{A}[\![|\dot{\Gamma}|, v : 1]\!] \end{aligned}$$

$\square$

LEMMA C.32. *Let* $\Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash \phi$ *be a formula.*

$$\alpha^* \mathcal{A}^{\mathcal{P}}[\![\Gamma, (x_1, x_2) : \sigma_1 \times \sigma_2 \vdash \phi]\!] = \mathcal{A}[\![\Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash \phi]\!]$$

Proof. By substitution and weakening.

$$\alpha^* \mathcal{A}^{\mathcal{P}} [\![\Gamma, (x_1, x_2) : \sigma_1 \times \sigma_2 \vdash \phi]\!]$$
$$= \alpha^* \mathcal{A}^{\mathcal{P}} [\![\Gamma, x : \sigma_1 \times \sigma_2 \vdash \phi[\pi_1 \, x/x_1, \pi_2 \, x/x_2]]\!]$$
$$= \alpha^* \langle \mathrm{id}, \pi_1 \circ \pi_2 \rangle^* \langle \mathrm{id}, \pi_2 \circ \pi_2 \circ \pi_1 \rangle^* \mathcal{A}^{\mathcal{P}} [\![\Gamma, (x_1, x_2) : \sigma_1 \times \sigma_2, x_1 : \sigma_1, x_2 : \sigma_2 \vdash \phi]\!]$$
$$= \alpha^* \langle \mathrm{id}, \pi_1 \circ \pi_2 \rangle^* \langle \mathrm{id}, \pi_2 \circ \pi_2 \circ \pi_1 \rangle^* ((\pi_1 \times \mathrm{id}) \times \mathrm{id})^* \mathcal{A}^{\mathcal{P}} [\![\Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash \phi]\!]$$
$$= \mathcal{A}^{\mathcal{P}} [\![\Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash \phi]\!]$$

$\square$

Lemma C.33. *Let* $\sigma_1, \ldots, \sigma_n \in B \cup \{1, \mathbf{Prop}\}$.

$$\mathcal{A}^{\mathcal{P}} [\![\dot\Gamma \vdash \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \phi\}]\!]$$
$$= ((\mathcal{A}[\![|\dot\Gamma|]\!], \quad \mathcal{A}[\![\sigma_1]\!] \times (\cdots \times \mathcal{A}[\![\sigma_n]\!])), \quad \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma]\!],$$
$$\pi^* \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, (v_1, (\ldots, v_n)) : \sigma_1 \times (\cdots \times \sigma_n) \vdash \phi]\!])$$

Proof. By induction on n. The base case is trivial. If $n > 1$,

$$\mathcal{A}^{\mathcal{P}} [\![\dot\Gamma \vdash \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \phi\}]\!]$$
$$= \coprod_{\mathcal{A}^{\mathcal{P}} [\![\dot\Gamma \vdash \{v_1 : \sigma_1 \mid \top\}]\!]} \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v_1 : \sigma_1 \vdash \{(v_2, \ldots, v_n) : \sigma_2 \times \cdots \times \sigma_n \mid \phi\}]\!]$$
$$= \coprod_{\mathcal{A}^{\mathcal{P}} [\![\dot\Gamma \vdash \{v_1 : \sigma_1 \mid \top\}]\!]} ((\mathcal{A}[\![|\dot\Gamma|, v_1 : \sigma_1]\!], \mathcal{A}[\![\sigma_2]\!] \times (\cdots \times \mathcal{A}[\![\sigma_n]\!])), \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v_1 : \sigma_1]\!],$$
$$\pi^* \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v_1 : \sigma_1]\!] \wedge \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, v_1 : \sigma_1, (v_2, (\ldots, v_n)) : \sigma_2 \times (\cdots \times \sigma_n) \vdash \phi]\!])$$
$$= ((\mathcal{A}[\![|\dot\Gamma|]\!], \mathcal{A}[\![b_1]\!] \times (\cdots \times \mathcal{A}[\![b_n]\!])), \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma]\!],$$
$$(\alpha^{-1})^* (\pi^* \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v_1 : b_1]\!] \wedge \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, v_1 : b_1, (v_2, (\ldots, v_n)) : b_2 \times (\cdots \times b_n) \vdash \phi]\!]))$$

We have

$$(\alpha^{-1})^* \pi^* \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v_1 : b_1]\!] = (\alpha^{-1})^* \pi^* \pi^* \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma]\!] = \pi^* \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma]\!]$$

We also have

$$(\alpha^{-1})^* \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, v_1 : b_1, (v_2, (\ldots, v_n)) : b_2 \times (\cdots \times b_n) \vdash \phi]\!]$$
$$= (\alpha^{-1})^* \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, v_1 : b_1, v' : b_2 \times (\cdots \times b_n) \vdash \phi[\ldots]]\!]$$
$$= (\alpha^{-1})^* \alpha^* \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, v : b_1 \times (\cdots \times b_n) \vdash \phi[\ldots][\pi_1 \, v/v_1, \pi_2 \, v/v']]\!]$$
$$= \mathcal{A}^{\mathcal{P}} [\![|\dot\Gamma|, (v_1, (\ldots, v_n)) : b_1 \times (\cdots \times b_n) \vdash \phi]\!]$$

by Lemma C.32. $\square$

Corollary C.34. *R-BasicSimp is sound.*

Proof. By Lemma C.30, it suffices to prove

$$(\mathrm{id} \times a(\mathbf{op})) : \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v : \{(v_1, \ldots, v_n) : \sigma_1 \times (\cdots \times \sigma_n) \mid \phi[\mathbf{op}(v_1, \ldots, v_n)/v]\}]\!] \dot\to \mathcal{A}^{\mathcal{P}} [\![\dot\Gamma, v : \{v : \tau \mid \phi\}]\!]$$

By Lemma C.33,

$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{(v_1, \ldots, v_n) : \mathbf{Prop}^n \mid \phi[\mathbf{op}(v_1, \ldots, v_n)/v]\}]\!]$$

$$= \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, (v_1, (\ldots, v_n)) : \sigma_1 \times (\cdots \times \sigma_n) \vdash \phi[\mathbf{op}(v_1, \ldots, v_n)/v]]\!]$$

$$= \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, u : \sigma_1 \times (\cdots \times \sigma_n) \vdash \phi[\mathbf{op}(u)/v]]\!]$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{v : \mathbf{Prop} \mid \phi\}]\!]$$

$$= \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, v : \tau \vdash \phi]\!]$$

By Lemma C.3,C.7,

$$\mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, u : \sigma_1 \times (\cdots \times \sigma_n) \vdash \phi[\mathbf{op}(u)/v]]\!]$$

$$= \langle \mathrm{id}, a(\mathbf{op}) \circ \pi_2 \rangle^* (\pi_1 \times \mathrm{id})^* \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, v : \tau \vdash \phi]\!]$$

$$= (\mathrm{id} \times a(\mathbf{op}))^* \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, v : \tau \vdash \phi]\!]$$

$\square$

COROLLARY C.35. *R-BasicBool is sound*

PROOF. By Lemma C.30, it suffices to prove

$$(\mathrm{id} \times a(\mathbf{op})) : \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \psi_t \wedge \phi_t[()/v] \vee \psi_f \wedge \phi_f[()/v]\}]\!]$$

$$\dot\to \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{v : 1 \mid \phi_t\} + \{v : 1 \mid \phi_f\}]\!]$$

By definition,

$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{v : 1 \mid \phi_t\}]\!]$$

$$= \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, v : 1 \vdash \phi_t]\!]$$

$$= \pi^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \pi^* \langle \mathrm{id}, ! \rangle^* \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, v : 1 \vdash \phi_t]\!]$$

$$= \pi^* (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_t[()/v]]\!])$$

$$\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{v : 1 \mid \phi_t\} + \{v : 1 \mid \phi_f\}]\!]$$

$$= \{\langle \gamma, \iota_1 \rangle \mid \gamma \in \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \cap \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_t[()/v]]\!]\} \cup \{\langle \gamma, \iota_2 \rangle \mid \gamma \in \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \cap \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_f[()/v]]\!]\}$$

$$= (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_t[()/v]]\!]) \dot\times \{\iota_1\} \vee (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_f[()/v]]\!]) \dot\times \{\iota_2\}$$

$$(\mathrm{id} \times a(\mathbf{op}))^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{v : 1 \mid \phi_t\} + \{v : 1 \mid \phi_f\}]\!]$$

$$= (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_t[()/v]]\!]) \dot\times (a(\mathbf{op}))^* \{\iota_1\}$$

$$\quad \vee (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_f[()/v]]\!]) \dot\times (a(\mathbf{op}))^* \{\iota_2\}$$

$$= \pi_1^* (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_t[()/v]]\!]) \wedge (! \times \mathrm{id})^* \pi_2^* (a(\mathbf{op}))^* \{\iota_1\}$$

$$\quad \vee \pi_1^* (\mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_f[()/v]]\!]) \wedge (! \times \mathrm{id})^* \pi_2^* (a(\mathbf{op}))^* \{\iota_2\}$$

$$\geq \pi_1^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge (\pi_1^* \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_t[()/v]]\!] \wedge (! \times \mathrm{id})^* \mathcal{A}^{\mathcal{P}}[\![(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \psi_t]\!]$$

$$\quad \vee \pi_1^* \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma| \vdash \phi_f[()/v]]\!] \wedge (! \times \mathrm{id})^* \mathcal{A}^{\mathcal{P}}[\![(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \psi_f]\!])$$

$$= \pi_1^* \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma]\!] \wedge \mathcal{A}^{\mathcal{P}}[\![|\dot\Gamma|, (v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \vdash \phi_t[()/v] \wedge \psi_t \vee \phi_f[()/v] \wedge \psi_f]\!]$$

$$= \mathcal{A}^{\mathcal{P}}[\![\dot\Gamma, v : \{(v_1, \ldots, v_n) : \sigma_1 \times \cdots \times \sigma_n \mid \psi_t \wedge \phi_t[()/v] \vee \psi_f \wedge \phi_f[()/v]\}]\!]$$

$\square$

Lemma C.36. *R-Unif is sound in the* λ_{HFL}*-model for expected costs and weakest pre-expectations.*

Proof. By Lemma C.30, it suffices to prove

$$(\mathrm{id} \times a(\mathbf{unif}))$$
$$: \mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}, v : (x : \{x : \mathbf{real} \mid 0 \le x \wedge x \le 1\}) \to \{v : \mathbf{Prop} \mid v \le N\, x\}]\!]$$
$$\dot{\to} \mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}, v : \{v : \mathbf{Prop} \mid v \le \mathbf{unif}(N)\}]\!]$$

By definition,

$$\mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}, v : (x : \{x : \mathbf{real} \mid 0 \le x \wedge x \le 1\}) \to \{v : \mathbf{Prop} \mid v \le N\, x\}]\!]$$

$$= \{ \prod_{\mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma} \vdash \{x:\mathbf{real} \mid 0 \le x \wedge x \le 1\}]\!]} \mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}, x : \{x : \mathbf{real} \mid 0 \le x \wedge x \le 1\} \vdash \{v : \mathbf{Prop} \mid v \le N\, x\}]\!] \}$$

$$= \{ \langle \gamma, f \rangle : 1 \to \mathcal{A}^{\xi}[\![|\dot{\Gamma}|]\!] \times (\mathbb{R} \Rightarrow \Omega) \mid \gamma \in \mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}]\!] \wedge$$

$$\forall x : 1 \to \mathbb{R}, 0 \le x \le 1 \implies \mathbf{ev} \circ \langle f, x \rangle \le \mathbf{ev} \circ \langle \mathcal{A}^{\xi}[\![N]\!] \circ \gamma, x \rangle \}$$

Recall that $a(\mathbf{unif}) : \mathbb{R} \Rightarrow \Omega \to \Omega$ is defined by $a(\mathbf{unif})(f) = \int_0^1 f(x)\, \mathrm{d}x$. By monotonicity of $a(\mathbf{unif})$, if

$$\langle \gamma, f \rangle \in \mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}, v : (x : \{x : \mathbf{real} \mid 0 \le x \wedge x \le 1\}) \to \{v : \mathbf{Prop} \mid v \le N\, x\}]\!]$$

then

$$(\mathrm{id} \times a(\mathbf{unif})) \circ \langle \gamma, f \rangle \in \{ \langle \gamma, y \rangle \mid \gamma \in \mathcal{A}^{\xi,\mathcal{P}}[\![\dot{\Gamma}]\!] \wedge y \le a(\mathbf{unif}) \circ \mathcal{A}^{\xi}[\![N]\!] \circ \gamma \}$$

$\square$

C.5 Proofs for Rules for Admissibility

Lemma C.37. *If* $[\![a]\!]$ *is chain-closed, then* $[\![a(M)]\!]$ *is chain-closed at any variable.*

Proof. Obvious. $\square$

Lemma C.38. $[\![\le_{\mathbf{Prop}}]\!]$ *is chain-closed.*

Proof. Easy. $\square$

Lemma C.39. *If* $Q_1, Q_2 \in \mathbb{P}_{[\![\Gamma]\!]}$ *are admissible at* x_i*, then so are* $Q_1 \cap Q_2$ *and* $Q_1 \cup Q_2$.

Proof. The only non-trivial part of the proof is the chain-closed-ness of $Q_1 \cup Q_2$. Assume $\gamma[x_i \mapsto f_k] \in Q_1 \cup Q_2$ holds for each k where $f_0 \le f_1 \le \ldots$ is an ω-chain in $\mathbb{C}(1, [\![\sigma_i]\!])$ and $\gamma : 1 \to [\![\Gamma]\!]$. Then either $\{k \mid \gamma[x_i \mapsto f_k] \in Q_1\}$ or $\{k \mid \gamma[x_i \mapsto f_k] \in Q_2\}$ must be an infinite set. That is, we have an infinite sequence of indices $k_0 \le k_1 \le \ldots$ such that either $\gamma[x_i \mapsto \sup_l f_{k_l}] \in Q_1$ or $\gamma[x_i \mapsto \sup_l f_{k_l}] \in Q_2$ holds. Since we have $\gamma[x_i \mapsto \sup_l f_{k_l}] = \gamma[x_i \mapsto \sup_k f_k]$, we have $\gamma[x_i \mapsto \sup_k f_k] \in Q_1 \cup Q_2$. $\square$

Lemma C.40. *If* $Q \in \mathbb{P}_{[\![\Gamma]\!]}$ *is admissible at* x_i *and* $R \in \mathbb{P}_{[\![\Gamma \setminus x_i]\!]}$*, then*

$$\{ \gamma : 1 \to [\![\Gamma]\!] \mid \gamma \in \mathrm{proj}^*_{\Gamma; x_i} R \implies \gamma \in Q \} \tag{15}$$

is admissible at x_i*. Here* $\mathrm{proj}_{\Gamma; v} : [\![\Gamma]\!] \to [\![\Gamma \setminus v]\!]$ *is the canonical projection.*

$$\mathrm{proj}_{\Gamma, v:\sigma; v} = \pi_1 \qquad \mathrm{proj}_{\Gamma, x:\sigma; v} = \mathrm{proj}_{\Gamma; v} \times \mathrm{id}$$

Proof. Note that for any $\gamma : 1 \to [\![\Gamma]\!]$ and $f : 1 \to [\![\sigma_i]\!]$, if $\gamma \in \mathrm{proj}^* R$, then $\gamma[x_i \mapsto f] \in \mathrm{proj}^* R$. For and $\gamma : 1 \to [\![\Gamma]\!]$ and any ω-chain $f_0 \le f_1 \le \ldots$ in $\mathbb{C}(1, [\![\sigma_i]\!])$, if $\gamma[x_i \mapsto f_j]$ is in (15) for any j, then there are two cases.

- If $\gamma[x_i \mapsto f_j] \notin \mathrm{proj}^* R$ for some j, then $\gamma[x_i \mapsto \sup_j f_j] \notin \mathrm{proj}^* R$.
- If $\gamma[x_i \mapsto f_j] \in \mathrm{proj}^* R$ for any j, then $\gamma[x_i \mapsto f_j] \in Q$ for any j. Therefore, $\gamma[x_i \mapsto \sup_j f_j] \in Q$.

In any case, $\gamma[x_i \mapsto \sup_j f_j]$ is in (15). $\qquad\square$

LEMMA C.41. *If* $\Gamma \vdash \mathrm{adm}(v, \phi)$, *then* $\mathcal{A}^{\mathcal{P}}[\![\Gamma \vdash \phi]\!]$ *is admissible at* v.

PROOF. By induction on derivation of $\Gamma \vdash \mathrm{adm}(v, \phi)$.

- ADM-LEQ: $[\![\mathbf{Prop}]\!]$ has a bottom element because it is an EM T-algebra. For each $\gamma : 1 \to [\![\Gamma]\!]$, we have $\gamma[v \mapsto \bot_{[\![\mathbf{Prop}]\!]}] \in [\![v \leq_{\mathbf{Prop}} M]\!]$ because $\bot_{[\![\mathbf{Prop}]\!]} \leq [\![M]\!] \circ \gamma[v \mapsto \bot_{[\![\mathbf{Prop}]\!]}]$. $[\![v \leq_{\mathbf{Prop}} M]\!]$ is chain-closed at v by Lemma C.37,C.38.
- ADM-AND, ADM-OR: by Lemma C.39.
- ADM-IMP: by Lemma C.40

$\qquad\square$

D BENCHMARK PROGRAMS

D.1 Weakest Pre-Expectation

<div align="center">lics16_rec3</div>

```
let[@adm] rec rec3 k =
  let p = 0.5 in
  p *. k ()
    +. (1.0 -. p) *. rec3 (fun () -> rec3 (fun () -> rec3 (fun () -> k ())))

[@@@assert "typeof(rec3) <: (unit -> { r : prop | r = 1.0 })
  -> { ret : prop | 0.0 <= ret && ret <= (2.236068 - 1.0) / 2.0 }"]
```

<div align="center">lics16_rec3_ghost</div>

```
let[@adm] rec rec3 (*ghost*)x k =
  let p = 0.5 in
  p *. k () +. (1.0 -. p) *. rec3 ((2.0 /. 3.0) *. (2.0 /. 3.0) *. x) (fun () ->
    rec3 ((2.0 /. 3.0) *. x) (fun () ->
      rec3 x (fun () -> k ())))

[@@@assert "typeof(rec3) <: { x : prop | x = 1.0 }
  -> (unit -> { r : prop | r = 1.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 2.0 / 3.0 }"]
```

<div align="center">lics16_coins</div>

```
let p1 = 1.0 /. 2.0
let p2 = 1.0 /. 3.0
let coins k =
  p1 *. (p2 *. k 0 0 +. (1.0 -. p2) *. k 0 1)
    +. (1.0 -. p1) *. (p2 *. k 1 0 +. (1.0 -. p2) *. k 1 1)

[@@@assert "typeof(coins) <: ((x:int) -> (y:int)
    -> { r : prop | x = y && r = 1.0 || x <> y && r = 0.0 })
  -> { ret : prop | ret = 0.5 }"]
```

D.2 Expected Cost Analysis

random_walk

```
let p = 2.0 /. 3.0
let[@adm] rec f x k =
  if x = 0
    then k ()
    else p *. (1.0 +. f (x - 1) k) +. (1.0 -. p) *. (1.0 +. f (x + 1) k)

[@@@assert "typeof(f) <: { x:int | x = 1 } -> (unit -> { r:prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 3.0 }"]
```

random_walk_unif

```
external unif : (float -> float) -> float = "unknown"
let[@adm] rec rw x k =
  if x >= 0.0 then
    unif (fun y ->
        let l = -2.0 in
        let r = 1.0 in
        1.0 +. rw (x +. (r -. l) *. y +. l) k)
  else k ()

[@@@assert "typeof(rw) <: { x:float | x = 1.0 } -> (unit -> { r:prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 6.0 }"]
```

coin_flip

```
let p = 0.5
let[@adm] rec f x k = p *. k () +. (1.0 -. p) *. (1.0 +. f () k)

[@@@assert "typeof(f) <: unit -> (unit -> { r : prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 1.0 }"]
```

coin_flip_unif

```
external unif : (float -> float) -> float = "unknown"
let[@adm] rec f x k = unif (fun p -> p *. k () +. (1.0 -. p) *. (1.0 +. f () k))

[@@@assert "typeof(f) <: unit -> (unit -> { r : prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 1.0 }"]
```

icfp21_walk

```
let[@adm] rec walk p f n k =
  p n (fun b -> if b then k n else 1.0 +. f n (fun m -> walk p f m k))

let geo = walk (fun _ k -> 0.5 *. k true +. 0.5 *. k false) (fun n k -> k (n + 1))

[@@@assert "typeof(geo) <: int -> (int -> { r : prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 1.0 }"]

let p = 2.0 /. 3.0
let randomWalk = walk (fun n k -> k (n <= 0))
  (fun n k -> p *. k (n - 1) +. (1.0 -. p) *. k (n + 1))

[@@@assert "typeof(randomWalk) <: (n:int) -> (int -> { r : prop | r = 0.0 })
```

```
  -> { ret : prop | 0.0 <= ret && ret <= 3.0 * float_of_int (abs n) }"]
```

icfp21_coupons

```
let rec length = function [] -> 0 | _ :: xs -> 1 + length xs
let rec mem c = function [] -> false | x :: xs -> c = x || mem c xs
let rec is_subset cs os =
  match cs with [] -> true | c :: cs' -> mem c os && is_subset cs' os
let rec sum f = function [] -> 0.0 | x :: xs -> f x +. sum f xs
let[@adm] rec collect cs os k =
  if is_subset cs os then k os
  else
    let len_inv = 1.0 /. float_of_int (length cs) in
    1.0 +. sum (fun c -> len_inv *. collect cs (c :: os) k) cs
let coupons cs k = collect cs [] k

[@@@assert "typeof(coupons) <: { x : int list | x = 1 :: 2 :: [] }
  -> (int list -> { r : prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret && ret <= 2.0 * (1.0 + 1.0 / 2.0) }"]
```

lics16_fact

```
let[@adm] rec fact x k =
  let p = 5.0 /. 6.0 in
  if x <= 0 then k 1
  else
    1.0 +. p *. fact (x - 1) (fun y -> k (y * x))
      +. (1.0 -. p) *. fact (x - 2) (fun y -> k (y * x))

[@@@assert "typeof(fact) <: (x:{ x:int | x >= 0 })
  -> (int -> { r : prop | r = 0.0 })
  -> { ret : prop | 0.0 <= ret
    && ret <= 1.0 + float_of_int x / (2.0 - 5.0 / 6.0) }"]
```

D.3 Cost Moment Analysis

coin_flip_ord2

```
let p = 0.5
let[@adm] rec f x k =
  let r11, r12 = k () in
  let r21, r22 = f () k in
  p *. r11 +. (1.0 -. p) *. (r21 +. 1.0),
  p *. r12 +. (1.0 -. p) *. (r22 +. 2.0 *. r21 +. 1.0)

[@@@assert "typeof(f) <: unit
  -> (unit -> { r : prop * prop | r = Tuple(0.0, 0.0) })
  -> { ret : prop * prop | 0.0 <= $proj(0, ret) && $proj(0, ret) <= 1.0
    && 0.0 <= $proj(1, ret) && $proj(1, ret) <= 3.0 }"]
```

coin_flip_ord3

```
let p = 0.5
let[@adm] rec f x k =
  let r11, r12, r13 = k () in
  let r21, r22, r23 = f () k in
  p *. r11 +. (1.0 -. p) *. (r21 +. 1.0),
```

```
  p *. r12 +. (1.0 -. p) *. (r22 +. 2.0 *. r21 +. 1.0),
  p *. r13 +. (1.0 -. p) *. (r23 +. 3.0 *. r22 +. 3.0 *. r21 +. 1.0)

[@@@assert "typeof(f) <: unit
  -> (unit -> { r : prop * prop * prop | r = Tuple(0.0, 0.0, 0.0) })
  -> { ret : prop * prop * prop | 0.0 <= $proj(0, ret) && $proj(0, ret) <= 1.0
    && 0.0 <= $proj(1, ret) && $proj(1, ret) <= 3.0
    && 0.0 <= $proj(2, ret) && $proj(2, ret) <= 13.0 }"]
```

D.4 Conditional Weakest Pre-Expectation

toplas18_ex4.4

```
let[@admc] rec f m k =
  let (r11, r12) = f (m + 1) k in
  let (r21, r22) = k (m + 1) in
  (0.75 *. r11 +. 0.125 *. r21, 0.75 *. r12 +. 0.125 *. r22)

[@@@assert "typeof(f) <: { m : int | m = 0 }
  -> ((m:int) -> { r : prop * real | (m = 1 && $proj(0, r) = 1.0
    || m <> 1 && $proj(0, r) = 0.0) && $proj(1, r) = 1.0 })
  -> { ret : prop * real | 0.0 <= $proj(0, ret)
    && $proj(0, ret) <= 0.25 * $proj(1, ret)
    && 0.0 <= $proj(1, ret) && $proj(1, ret) <= 1.0 }"]
```

two_coin_conditioning

```
let[@admc] rec diverge () (k : unit -> float * float) : float * float
  = diverge () k

[@@@assert "typeof(diverge) <: unit -> (unit -> prop * real)
  -> { ret : prop * real | $proj(0, ret) = 0.0 && $proj(1, ret) = 1.0 }"]

let[@admc] rec f m k =
  let (r11, r12) = f () k in
  let (r21, r22) = k () in
  let (r31, r32) = diverge () k in
  0.25 *. r11 +. 0.25 *. r21 +. 0.25 *. r31,
  0.25 *. r12 +. 0.25 *. r22 +. 0.25 *. r32

[@@@assert "typeof(f) <: unit
  -> (unit -> { r : prop * real | $proj(0, r) = 1.0 && $proj(1, r) = 1.0 })
  -> { ret : prop * real | 0.0 <= $proj(0, ret)
    && $proj(0, ret) <= 0.5 * $proj(1, ret)
    && 0.0 <= $proj(1, ret) && $proj(1, ret) <= 1.0 }"]
```