3-2007

Syntactic Logical Relations for Polymorphic and Recursive Types

Karl Crary
Carnegie Mellon University, crary@cs.cmu.edu

Robert Harper
Carnegie Mellon University, rwh@cs.cmu.edu

Published In
.

Syntactic Logical Relations for Polymorphic and Recursive Types

Karl Crary[1] Robert Harper[2]

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The method of logical relations assigns a relational interpretation to types that expresses operational invariants satisfied by all terms of a type. The method is widely used in the study of typed languages, for example to establish contextual equivalences of terms. The chief difficulty in using logical relations is to establish the existence of a suitable relational interpretation. We extend work of Pitts and Birkedal and Harper on constructing relational interpretations of types to polymorphism and recursive types, and apply it to establish parametricity and representation independence properties in a purely operational setting. We argue that, once the existence of a relational interpretation has been established, it is straightforward to use it to establish properties of interest.

Keywords: Operational semantics, type structure, logics of programs, lambda calculus and related systems, data abstraction, polymorphism.

1 Introduction

It is a pleasure to contribute this paper in honor of Gordon D. Plotkin on the occasion of his sixtieth birthday. Plotkin's research on the mathematical foundations of programming languages is singularly influential, providing the foundations for much subsequent work and establishing new approaches to neglected or ill-understood problems. Several themes have direct bearing on the work described herein.

One important theme is the use of operational semantics for defining and analyzing programming languages. Plotkin's *structural operational semantics* [25,24] is particularly influential. Execution is modelled as a transition system whose states are programs. The transition relation is inductively defined by a collection of inference rules that mirror the structure of the program. The transition rules specify

[1] Email: crary@cs.cmu.edu
[2] Email: rwh@cs.cmu.edu

both the individual steps of execution and the order in which these steps are executed. Properties of the execution behavior of programs, such as type safety, may be proved by induction over the transition rules.

Another theme in Plotkin's work is the analysis of *observational equivalence* between program fragments. Applying Leibniz's Principle of Identity of Indiscernibles, two program fragments are observationally equivalent iff they engender the same observable outcomes however they may be used in a complete program. Being the coarsest consistent congruence, one may use coinduction to prove that two fragments are observationally equivalent. But in practice such a direct proof is rarely effective; a more tractable characterization is needed. Plotkin, in his seminal study of PCF [22], considered denotational methods that turned out to provide sufficient, but not necessary, conditions for observational equivalence. This requires proving *computational adequacy* of the denotational semantics relative to the operational semantics of PCF, which Plotkin proved using the method of *logical relations* [32].

Pitts showed that logical relations may also be used to obtain a useful characterization of observational equivalence [18]. Associated with each type is an equivalence relation, called *logical equivalence*, that is defined by assigning an action on relations to each type constructor. The relational action is defined so that logical equivalence is a consistent congruence, so that it is a sufficient (and, in many cases, necessary) condition for observational equivalence. For simple type systems logical equivalence is defined by induction on the structure of types, leading to an elegant proof method for establishing observational equivalences. But in richer type systems more complex methods are required.

One complicating factor is *impredicative polymorphism*, as introduced by Girard and Reynolds [7,8,28]. The concept of *relational parametricity* [29] generalizes logical equivalence to polymorphic types. As in simpler cases, parametricity may be used to characterize observational equivalence for polymorphic languages. Through Mitchell and Plotkin's account of abstract types as existential types [16] (which may be encoded using only polymorphic types), relational parametricity may be used to prove representation independence for abstract types [15]. This may be seen as a generalization of Hoare's method [10] of proving correctness of abstract type implementations.

Recursive types present further complications for the method of logical relations. Here again we encounter central themes in Plotkin's research, namely solving recursive type equations, and developing the theory of parametricity in the presence of recursive types. Building on Scott's lattice models for the untyped λ-calculus, Plotkin developed the denotational semantics of recursive types [23]. In particular, his work with Smyth [31] initiated the category-theoretic study of recursive types, leading to Freyd's universal characterization of a recursively defined type as a *minimal invariant*. Abadi and Plotkin studied relational parametricity in the presence of recursive types in a denotational setting, proposing a partial equivalence relation interpretation [1] and developing a logic for parametricity in this setting [26].

All of these themes of Plotkin's research figure into the present work. We consider a call-by-value operational interpretation of the Girard-Reynolds polymorphic

typed λ-calculus [7,28] extended with general recursive types. Our main result is a complete characterization of observational equivalence—in which we observe termination at unit type—in terms of logical equivalence. We then use logical equivalence to state and prove parametricity properties of the language. In particular, we reproduce an example from Sumii and Pierce [33] to provide a comparison with their bisimulation technique.

The main technical challenge is to define logical equivalence for such a rich type system. We wish to provide a definition that is sound and complete with respect to observational equivalence, and that permits us to derive parametricity properties of polymorphic types. To account for recursive types we work over a complete lattice of admissible relations over terms that are pointed, respect observational equivalence, and support reasoning by fixed point induction (see Section 2 for precise definitions). We assign to each type constructor an admissible action on this space of relations that ensures that logical equivalence is a congruence, and hence contained in observational equivalence. Since admissible relations respect observational equivalence, the desired complete characterization follows.

The crucial problem is to define an admissible relational action to each type constructor of the language. The relational action for the function type constructor relates functions that map related arguments to related results. To assign a relational action to polymorphic types we use a variant of Girard's method, with admissible relations as candidates. The action relates two polymorphic expressions if corresponding instances are related for each choice of admissible relation between those instance types. Following Pitts [19], the relational action associated with a general recursive type relies on an operational version of Freyd's *minimal invariant* characterization of the solution of a type equation. This property, called *syntactic minimal invariance*, was proved by Birkedal and Harper [4] for a simply typed language with a single recursive type using *ciu equivalence* [14]. Here we give a new, streamlined proof based on *applicative equivalence* [12]. Logical equivalence is then constructed by exploiting syntactic minimal invariance to handle recursive types, and Girard's method for polymorphic types.

Logical equivalence is particularly convenient for proving parametricity and representation independence properties of the language. Arguing that logical relations-based methods for deriving these properties are overly complex, Sumii and Pierce [33] have developed a new method based on a novel form of bisimulation. We argue, on the contrary, that the main complication is in justifying the definition of logical equivalence, which can be done once and for all. We show in Section 6 that deriving representation independence properties using logical equivalence is straightforward. For example, we show how to obtain the results of Sumii and Pierce by exploiting relational parametricity. The proof reduces to choosing a suitable admissible relation, and proving that this relation is preserved by the operations of the abstract type.

$$
\begin{array}{lll}
\textit{Types} & \tau & ::= \alpha \mid 1 \mid \tau_1 \to \tau_2 \mid \forall\alpha.\tau \mid \mu\alpha.\tau \\
\textit{Terms} & e & ::= x \mid * \mid \lambda x{:}\tau.e \mid e_1 e_2 \mid \Lambda\alpha.e \mid e[\tau] \mid \mathtt{in}_{\mu\alpha.\tau} e \mid \mathtt{out}\, e \\
\textit{Values} & v & ::= x \mid * \mid \lambda x{:}\tau.e \mid \Lambda\alpha.e \mid \mathtt{in}_{\mu\alpha.\tau} v \mid \\
\textit{Typings} & \Gamma & ::= \epsilon \mid \Gamma, \alpha \mid \Gamma, x{:}\tau
\end{array}
$$

Fig. 1. Syntax

2 Preliminaries

2.1 The Language

The syntax of the language is given in Figure 1. The static semantics is given in Figure 2 and the operational semantics is given in Figure 3. The operational semantics consists of a small-step, call-by-value transition relation between closed terms, written $e \mapsto e'$.

We define the set of types and the type-indexed family of sets of expressions as follows:

$$\textit{Type} \stackrel{\text{def}}{=} \{\tau \mid \vdash \tau \text{ type}\}$$
$$\textit{Exp}_\tau \stackrel{\text{def}}{=} \{e \mid \vdash e : \tau\}$$

We write capture-avoiding substitution of E for X in E' as $E'[E/X]$. As usual, we identify expressions that differ only in the names of bound variables.

The language enjoys the usual type safety properties, as expressed by the following lemmas, which we use throughout without explicit reference.

Lemma 2.1 (Type preservation) *If $\vdash e : \tau$ and $e \mapsto e'$ then $\vdash e' : \tau$.*

Lemma 2.2 (Progress) *If $\vdash e : \tau$ and e is not a value, then (for some e') $e \mapsto e'$.*

Lemma 2.3 (Unique types) *If $\Gamma \vdash e : \tau$ and $\Gamma \vdash e : \tau'$ then $\tau = \tau'$.*

We will employ the abbreviations in Figure 4. Justified by unicity of types, we will sometimes omit the type subscripts from these abbreviations when they are clear from context. These elementary properties of the dynamic semantics will be of use in the sequel:

$$
\begin{array}{rcll}
\bot_\tau & \mapsto^2 & \bot_\tau & \\
\mathit{fix}_{\tau_1\to\tau_2}\, F & \mapsto^3 & \lambda y{:}\tau_1.\, F\,(\mathit{fix}\, F)\, y & \text{(for values } F\text{)} \\
\mathit{fix}^{i+1}_{\tau_1\to\tau_2}\, F & \mapsto & \lambda y{:}\tau_1.\, F(\mathit{fix}^i\, F) y & \text{(for values } F\text{)}
\end{array}
$$

$$\frac{}{\vdash \epsilon \ \text{context}}$$

$$\frac{\vdash \Gamma \ \text{context} \quad \Gamma \vdash \tau \ \text{type} \quad x \notin \text{Dom}(\Gamma)}{\vdash \Gamma, x{:}\tau \ \text{context}} \qquad \frac{\vdash \Gamma \ \text{context} \quad \alpha \notin \text{Dom}(\Gamma)}{\vdash \Gamma, \alpha \ \text{context}}$$

$$\frac{\vdash \Gamma \ \text{context} \quad \text{FV}(\tau) \subseteq \text{Dom}(\Gamma)}{\Gamma \vdash \tau \ \text{type}}$$

$$\frac{\vdash \Gamma \ \text{context} \quad \Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\vdash \Gamma \ \text{context}}{\Gamma \vdash * : 1}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.e : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

$$\frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau} \qquad \frac{\Gamma \vdash e : \forall\alpha.\tau' \quad \Gamma \vdash \tau \ \text{type}}{\Gamma \vdash e[\tau] : \tau'[\tau/\alpha]}$$

$$\frac{\Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \mathtt{in}_{\mu\alpha.\tau} e : \mu\alpha.\tau} \qquad \frac{\Gamma \vdash e : \mu\alpha.\tau}{\Gamma \vdash \mathtt{out}\, e : \tau[\mu\alpha.\tau/\alpha]}$$

Fig. 2. Static Semantics

$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2} \qquad \frac{e \mapsto e'}{v\, e \mapsto v\, e'} \qquad \frac{}{(\lambda x{:}\tau.e)v \mapsto e[v/x]}$$

$$\frac{e \mapsto e'}{e[\tau] \mapsto e'[\tau]} \qquad \frac{}{(\Lambda\alpha.e)[\tau] \mapsto e[\tau/\alpha]}$$

$$\frac{e \mapsto e'}{\mathtt{in}_{\mu\alpha.\tau} e \mapsto \mathtt{in}_{\mu\alpha.\tau} e'} \qquad \frac{e \mapsto e'}{\mathtt{out}\, e \mapsto \mathtt{out}\, e'} \qquad \frac{}{\mathtt{out}(\mathtt{in}_{\mu\alpha.\tau} v) \mapsto v}$$

Fig. 3. Structured Operational Semantics

2.2 Applicative Equivalence

The relational interpretation of types will be defined over equivalence classes of terms taken modulo *applicative equivalence* [11], a convenient form of operational

$$
\begin{aligned}
id_\tau &\stackrel{\text{def}}{=} \lambda x{:}\tau.x \\
\bot_\tau &\stackrel{\text{def}}{=} (\lambda x{:}T.\,(\mathtt{out}\,x)\,x)\;(\mathtt{in}_T(\lambda x{:}T.\,(\mathtt{out}\,x)\,x)) \\
&\quad (\text{where } T = \mu\alpha.\,\alpha \to \tau) \\
\mathit{fix}_{\tau_1\to\tau_2} &\stackrel{\text{def}}{=} (\mathtt{out}(\mathtt{in}_{T'}v))(\mathtt{in}_{T'}v) \\
&\quad (\text{where } v \;= \lambda x{:}T'.\,\lambda f{:}T\to T.\,\lambda y{:}\tau_1.\,f\,((\mathtt{out}\,x)\,x\,f)\,y \\
&\quad\;\; \text{and} \quad T \;= \tau_1 \to \tau_2 \\
&\quad\;\; \text{and} \quad T' = \mu\alpha.\,\alpha \to (T\to T) \to T) \\
\mathit{fix}^0_{\tau_1\to\tau_2} &\stackrel{\text{def}}{=} \lambda f{:}((\tau_1\to\tau_2)\to\tau_1\to\tau_2).\,\lambda y{:}\tau_1.\,\bot_{\tau_2} \\
\mathit{fix}^{i+1}_{\tau_1\to\tau_2} &\stackrel{\text{def}}{=} \lambda f{:}((\tau_1\to\tau_2)\to\tau_1\to\tau_2).\,\lambda y{:}\tau_1.\,f\,(\mathit{fix}^i_{\tau_1\to\tau_2}\,f)\,y \\
\mathit{fix}^\omega_{\tau_1\to\tau_2} &\stackrel{\text{def}}{=} \mathit{fix}_{\tau_1\to\tau_2}
\end{aligned}
$$

Fig. 4. Abbreviations

equivalence. (In Section 5 we show that applicative equivalence coincides with contextual equivalence, the coarsest consistent equivalence relation on expressions.)

Applicative equivalence of open terms is defined by considering its closed substitution instances. For this and other purposes we need the notion of a substitution for the variables in a typing context.

Definition 2.4 *We define a* substitution *to be a mapping from type variables to types and from term variables to terms. A substitution* σ satisfies *a context* Γ *(written* $\vdash \sigma : \Gamma$*) if:*

- $\vdash \Gamma$ context, *and*
- $\mathrm{Dom}(\sigma) = \mathrm{Dom}(\Gamma)$, *and*
- *for all* $\alpha \in \Gamma$, $\vdash \sigma(\alpha)$ type, *and*
- *for all* $x : \tau \in \Gamma$, $\vdash \sigma(x) : \sigma(\tau)$.

Applicative approximation is coinductively defined to be the largest pre-order satisfying conditions determined by the elimination rules for each type. We distinguish between approximation of *computations*, which may not terminate, from *values*, which are fully evaluated.

Definition 2.5

(i) *Applicative approximation is defined to be the greatest relation* $\vdash e_1 \preceq e_2 : \tau$ *over closed terms such that:*
- $\vdash e_1 \preceq e_2 : \tau$ *only if* $\vdash e_1, e_2 : \tau$, *and* $e_1 \mapsto^* v_1$ *implies that* $e_2 \mapsto^* v_2$ *for some* v_2 *such that* $\vdash v_1 \preceq_{val} v_2 : \tau$, *where*
- $\vdash v_1 \preceq_{val} v_2 : \tau$ *if and only if* $\vdash v_1, v_2 : \tau$, *and*

- $\tau = 1$, *or*
- $\tau = \tau_1 \to \tau_2$ *and (for all* v *such that* $\vdash v : \tau_1$*)* $\vdash v_1 v \preceq v_2 v : \tau_2$, *or*
- $\tau = \forall\alpha.\tau'$ *and (for all* τ'' *such that* $\vdash \tau''$ type*)* $\vdash v_1[\tau''] \preceq v_2[\tau''] : \tau'[\tau''/\alpha]$, *or*
- $\tau = \mu\alpha.\tau'$ *and* $\vdash \mathtt{out}\, v_1 \preceq \mathtt{out}\, v_2 : \tau'[\tau/\alpha]$.

(ii) *We extend applicative approximation to open terms as follows:* $\Gamma \vdash e_1 \preceq e_2 : \tau$ *if* $\Gamma \vdash e_1, e_2 : \tau$, *and (for all* σ *such that* $\vdash \sigma : \Gamma$*)* $\vdash \sigma(e_1) \preceq \sigma(e_2) : \sigma(\tau)$.

(iii) *Two terms* e_1 *and* e_2 *are applicatively equivalent at type* τ *in context* Γ *(written* $\Gamma \vdash e_1 \approx e_2 : \tau$*) if* $\Gamma \vdash e_1 \preceq e_2 : \tau$ *and* $\Gamma \vdash e_2 \preceq e_1 : \tau$.

(iv) *If* $\vdash e_1, e_2 : \tau$ *then we write* $e_1 \preceq e_2$ *to mean* $\vdash e_1 \preceq e_2 : \tau$. *If* $\Gamma \vdash e_1, e_2 : \tau$ *then we write* $\Gamma \vdash e_1 \preceq e_2$ *to mean* $\Gamma \vdash e_1 \preceq e_2 : \tau$.

(v) *Similarly, if* $\vdash e_1, e_2 : \tau$ *then we write* $e_1 \approx e_2$ *to mean* $\vdash e_1 \approx e_2 : \tau$. *If* $\Gamma \vdash e_1, e_2 : \tau$ *then we write* $\Gamma \vdash e_1 \approx e_2$ *to mean* $\Gamma \vdash e_1 \approx e_2 : \tau$.

Note that the "only if" conditions on applicative approximation become "if and only if" for applicative equivalence defined as the greatest fixed point of these conditions.

The following elementary properties follow readily from the definitions.

Proposition 2.6

- *Applicative approximation and equivalence are reflexive (over appropriately typed terms) and transitive.*
- *Applicative equivalence is symmetric.*
- *If* e_1 *is well-typed and* $e_1 \mapsto e_2$ *then* $e_1 \approx e_2$.

Lemma 2.7 (Substitutivity and Congruence) *Applicative approximation is substitutive and a congruence, in the following sense:*

- *If* $\Gamma, x{:}\tau, \Gamma' \vdash e_1 \preceq e_1'$ *and* $\Gamma \vdash e_2 \preceq e_2' : \tau$ *then* $\Gamma, \Gamma' \vdash e_1[e_2/x] \preceq e_1'[e_2'/x]$.
- *Applicative approximation is closed under the following rules:*

$$\frac{\Gamma, x{:}\tau \vdash e \preceq e'}{\Gamma \vdash \lambda x{:}\tau.e \preceq \lambda x{:}\tau.e'} \qquad \frac{\Gamma \vdash e_1 \preceq e_1' \quad \Gamma \vdash e_2 \preceq e_2'}{\Gamma \vdash e_1 e_2 \preceq e_1' e_2'}$$

$$\frac{\Gamma, \alpha \vdash e \preceq e'}{\Gamma \vdash \Lambda\alpha.e \preceq \Lambda\alpha.e'} \qquad \frac{\Gamma \vdash e \preceq e'}{\Gamma \vdash e[\tau] \preceq e'[\tau]}$$

$$\frac{\Gamma \vdash e \preceq e'}{\Gamma \vdash \mathtt{in}_{\mu\alpha.\tau} e \preceq \mathtt{in}_{\mu\alpha.\tau} e'} \qquad \frac{\Gamma \vdash e \preceq e'}{\Gamma \vdash \mathtt{out}\, e \preceq \mathtt{out}\, e'}$$

Proof. By a straightforward application of Howe's method [11]. □

Corollary 2.8 *For all well-formed types* τ_1 *and* τ_2, *if* $i \leq j$ *then* $fix^i_{\tau_1\to\tau_2} \preceq fix^j_{\tau_1\to\tau_2} \preceq fix_{\tau_1\to\tau_2}$.

Applicative equivalence is established by exhibiting a relation satisfying the requirements of Definition 2.5.

Lemma 2.9 (Coinduction for Applicative Approximation) *Suppose R is a type-indexed relation such that for all $\tau \in Type$, R_τ is a binary relation over Exp_τ. Suppose further that:*

(i) *if $e\ R_\tau\ e'$ and e halts then e' halts, and*

(ii) *if $e\ R_{\tau_1 \to \tau_2}\ e'$ and $v \in Exp_{\tau_1}$ then $e\,v\ R_{\tau_2}\ e'\,v$, and*

(iii) *if $e\ R_{\forall\alpha.\tau}\ e'$ and $\tau' \in Type$ then $e[\tau']\ R_{\tau[\tau'/\alpha]}\ e'[\tau']$, and*

(iv) *if $e\ R_{\mu\alpha.\tau}\ e'$ then $\mathtt{out}\,e\ R_{\tau[\mu\alpha.\tau/\alpha]}\ \mathtt{out}\,e'$.*

Then $e\ R_\tau\ e'$ implies $e \preceq e'$.

Proof. Let R' and R'_{val} be the relations defined as follows:

- $\vdash e_1\ R'\ e_2 : \tau$ if and only if $\vdash e_1, e_2 : \tau$ and there exists e'_1, e'_2 such that $e_1 \approx e'_1\ R\ e'_2 \approx e_2$.
- $\vdash v_1\ R'_{val}\ v_2 : \tau$ if and only if $\vdash v_1, v_2 : \tau$, and
  - $\tau = 1$, or
  - $\tau = \tau_1 \to \tau_2$ and (for all v such that $\vdash v : \tau_1$) $\vdash v_1 v\ R'\ v_2 v : \tau_2$, or
  - $\tau = \forall\alpha.\tau'$ and (for all τ'' such that $\vdash \tau''$ type) $\vdash v_1[\tau'']\ R'\ v_2[\tau''] : \tau'[\tau''/\alpha]$, or
  - $\tau = \mu\alpha.\tau'$ and $\vdash \mathtt{out}\,v_1\ R'\ \mathtt{out}\,v_2 : \tau'[\tau/\alpha]$.

We claim that if $e_1\ R'\ e_2$ and $e_1 \mapsto^* v_1$ then $e_2 \mapsto^* v_2$ for some v_2 such that $v_1\ R'_{val}\ v_2$. It follows by coinduction that $e_1\ R'\ e_2$ implies $e_1 \preceq e_2$, since R' fits the specification of $\preceq$, and $\preceq$ is the greatest relation fitting its specification. The result then follows by the reflexivity of $\approx$.

Suppose $e_1 \approx e'_1\ R\ e'_2 \approx e_2$ and $e_1 \mapsto^* v_1$. Then $e'_1{\downarrow}$, and by condition 1, $e'_2{\downarrow}$, so $e_2{\downarrow}$. Thus, let $e_2 \mapsto^* v_2$. Note that $v_1 \approx e'_1$ and $v_2 \approx e'_2$.

It remains to show that $\vdash v_1\ R'_{val}\ v_2 : \tau$, where τ is the type of v_1 and v_2. We proceed by cases on τ. Suppose $\tau = \tau_1 \to \tau_2$. Let $\vdash v : \tau_1$ be arbitrary. By condition 2, $e'_1 v\ R\ e'_2 v$. Thus, using congruence, $v_1 v \approx e'_1 v\ R\ e'_2 v \approx v_2 v$. Hence $v_1 v\ R'\ v_2 v$, and consequently $v_1\ R'_{val}\ v_2$. The other cases are similar or trivial. □

2.3 Compactness

The operational analogue of fixed point induction relies on *compactness*, which states that in a terminating computation only finitely many "unrollings" of a recursive function are necessary for the result. However, in the presence of higher types, the precise statement must account for residual occurrences of recursive functions in the result. We can elegantly manage such residual occurrences by expressing unrollings up to applicative approximation, rather than equality. This device helps because we may observe that a term with residual occurrences dominates a similar term without them, and thereby neglect those residual occurrences.

This focus on applicative approximation leads us to prove a stronger, least upper bound theorem (Theorem 2.11 below). From that theorem we can obtain compact-

ness as a simple corollary. However, we will find that many later results are more conveniently obtained by using the least upper bound theorem directly.

Notation Let w be a distinguished variable. Then $e^{f[i]}$ is defined to mean $e[\mathit{fix}^i\, f/w]$. Note that by Corollary 2.8 and congruence, if $i \leq j$ then $e^{f[i]} \preceq e^{f[j]} \preceq e^{f[\omega]}$ (provided the three terms are closed and well-typed).

Lemma 2.10 (Simulation) *Suppose f is a value, and suppose $e^{f[\omega]} \mapsto^* v$ (where $e^{f[\omega]}$ is closed and well-typed). Then there exist j, v' such that $v = v'^{f[\omega]}$ and for all $k \geq j$, $e^{f[k]} \succeq v'^{f[k-j]}$.*

Proof. Let $e^{f[\omega]} \mapsto^l v$. The proof is by induction on l, with an inner induction on the structure of e. If e is a value, the result follows trivially by letting $v' = e$. We proceed by cases on the non-value forms of e.

Case 1: Suppose $e = w$. Then $e^{f[\omega]} = \mathit{fix}\, f$. Since $e^{f[\omega]}$ is well-typed, f must have function type, so let $\vdash f : \tau_1 \to \tau_2$. Then $v = \lambda y{:}\tau_1.\, f\,(\mathit{fix}\, f)\, y$. Let $v' = \lambda y{:}\tau_1.\, f\, w\, y$, and let $j = 1$. Then $v = v'^{f[\omega]}$. Suppose $k \geq 1$. Then:

$$\begin{aligned} e^{f[k]} &= \mathit{fix}^k\, f \\ &\mapsto^+ \lambda y{:}\tau_1.\, f(\mathit{fix}^{k-1}\, f)y \text{ (since } k > 0) \\ &= v'^{f[k-1]} \end{aligned}$$

Hence $e^{f[k]} \succeq v'^{f[k-j]}$.

Case 2: Suppose $e = e_1 e_2$. Then:

$$\begin{aligned} e^{f[\omega]} &= e_1^{f[\omega]} e_2^{f[\omega]} \\ &\mapsto^m v_1 e_2^{f[\omega]} \quad \text{(for some } m < l) \\ &\mapsto^n v_1 v_2 \quad \text{(for some } n < l) \\ &\mapsto^+ v \end{aligned}$$

By induction, there exist i_1, v_1', i_2, v_2' such that (for $p = 1, 2$) $v_p = v_p'^{f[\omega]}$ and for all $k \geq i_p$, $e_p^{f[k]} \succeq v_p'^{f[k-i_p]}$. Observe that v_1' must be of the form $\lambda x{:}\tau.e_1'$. Let $e' = e_1'[v_2'/x]$. Then:

$$\begin{aligned} e^{f[\omega]} &\mapsto^* v_1 v_2 \\ &= (\lambda x{:}\tau.\, e_1'^{f[\omega]})\, v_2'^{f[\omega]} \\ &\mapsto e_1'^{f[\omega]}[v_2'^{f[\omega]}/x] \\ &= e'^{f[\omega]} \\ &\mapsto^o v \quad \text{(for some } o < l) \end{aligned}$$

By induction, there exist i, v' such that $v = v'^{f[\omega]}$ and for all $k \geq i$, $e'^{f[k]} \succeq v'^{f[k-i]}$. Let $j = i + i_1 + i_2$ and suppose $k \geq j$. Then:

$$\begin{aligned}
e^{f[k]} &= e_1^{f[k]} e_2^{f[k]} \\
&\succeq v_1'^{f[k-i_1]} v_2'^{f[k-i_2]} && \text{(by congruence)} \\
&\succeq v_1'^{f[k-i_1-i_2]} v_2'^{f[k-i_1-i_2]} && \text{(by congruence)} \\
&= (\lambda x{:}\tau.e_1'^{f[k-i_1-i_2]}) v_2'^{f[k-i_1-i_2]} \\
&\mapsto e_1'^{f[k-i_1-i_2]}[v_2'^{f[k-i_1-i_2]}/x] \\
&= (e_1'[v_2'/x])^{f[k-i_1-i_2]} \\
&= e'^{f[k-i_1-i_2]} \\
&\succeq v'^{f[(k-i_1-i_2)-i]} && \text{(since } k - i_1 - i_2 \geq i\text{)} \\
&= v'^{f[k-j]}
\end{aligned}$$

Hence $e^{f[k]} \succeq v'^{f[k-j]}$.

Case 3: Suppose $e = \mathtt{in}_{\mu\alpha.\tau} e_1$. Then v is of the form $\mathtt{in}_{\mu\alpha.\tau} v_1$, where $e_1^{f[\omega]} \mapsto^* v_1$. By the inner induction, there exist j, v_1' such that $v_1 = v_1'^{f[\omega]}$ and for all $k \geq j$, $e_1^{f[k]} \succeq v_1'^{f[k-j]}$. Let $v' = \mathtt{in}_{\mu\alpha.\tau} v_1'$. Then $v = v'^{f[\omega]}$, and for all $k \geq j$, $e^{f[k]} = \mathtt{in}_{\mu\alpha.\tau} e_1^{f[k]} \succeq \mathtt{in}_{\mu\alpha.\tau} v_1'^{f[k-j]} = v'^{f[k-j]}$, by congruence.

The other two cases are similar. □

Lemma 2.11 (Least Upper Bound) *Suppose f halts.*[3] *If $\forall j.\, e^{f[j]} \preceq e'$ then $e^{f[\omega]} \preceq e'$.*

Proof. Since f halts, by congruence we may assume, without loss of generality, that f is a value. Let R and R_{val} be the relations defined as follows:

- $\vdash e_1\ R\ e_2 : \tau$ if and only if $\vdash e_1, e_2 : \tau$, and e_1 has the form $e_1'^{f[\omega]}$ where $\forall j.\, e_1'^{f[j]} \preceq e_2$.
- $\vdash v_1\ R_{val}\ v_2 : \tau$ if and only if $\vdash v_1, v_2 : \tau$, and
  - $\tau = 1$, or
  - $\tau = \tau_1 \to \tau_2$ and (for all v such that $\vdash v : \tau_1$) $\vdash v_1 v\ R\ v_2 v : \tau_2$, or
  - $\tau = \forall\alpha.\tau'$ and (for all τ'' such that $\vdash \tau''$ type) $\vdash v_1[\tau'']\ R\ v_2[\tau''] : \tau'[\tau''/\alpha]$, or
  - $\tau = \mu\alpha.\tau'$ and $\vdash \mathtt{out}\, v_1\ R\ \mathtt{out}\, v_2 : \tau'[\tau/\alpha]$.

We claim that if $e_1\ R\ e_2$ and $e_1 \mapsto^* v_1$ then $e_2 \mapsto^* v_2$ for some v_2 such that $v_1\ R_{val}\ v_2$. The result follows from this claim by coinduction, since R therefore fits the specification of $\preceq$, and $\preceq$ is the greatest relation fitting its specification.

Therefore, suppose $e_1\ R\ e_2$ and $e_1 \mapsto^* v_1$. Then $e_1'^{f[\omega]} \mapsto^* v_1$. By Lemma 2.10, there exist j, v_1' such that $v_1 = v_1'^{f[\omega]}$ and $\forall k \geq j.\, v_1'^{f[k-j]} \preceq e_1'^{f[k]}$. By assumption,

[3] The lemma can also easily be seen to hold if f does not halt, since (for all k) $\mathit{fix}\, \bot \approx \mathit{fix}^k\, \bot$.

transitivity, and a change of variables (letting $i = k - j$), $\forall i.\, v_1'^{f[i]} \preceq e_2$. Therefore $e_2 \mapsto^* v_2$ and (since evaluation is deterministic) $\forall i.\, v_1'^{f[i]} \preceq_{val} v_2$. It remains to show that $v_1\ R_{val}\ v_2$. We proceed by cases on the type of v_1:

Case 1: Suppose $\vdash v_1 : 1$. Then $v_1\ R_{val}\ v_2$.

Case 2: Suppose $\vdash v_1 : \tau_1 \to \tau_2$ and suppose $\vdash v : \tau_1$. Then $v_1 v = (v_1' v)^{f[\omega]}$. But for any i, $(v_1' v)^{f[i]} = (v_1'^{f[i]})v \preceq v_2 v$ by congruence. Thus $v_1 v\ R\ v_2 v$ and hence $v_1\ R_{val}\ v_2$.

Case 3: Suppose $\vdash v_1 : \forall\alpha.\tau$ and suppose $\vdash \tau'$ type. Then $v_1[\tau'] = (v_1'[\tau'])^{f[\omega]}$. But for any i, $(v_1'[\tau'])^{f[i]} = (v_1'^{f[i]})[\tau'] \preceq v_2[\tau']$ by congruence. Thus $v_1[\tau']\ R\ v_2[\tau']$ and hence $v_1\ R_{val}\ v_2$.

Case 4: Suppose $\vdash v_1 : \mu\alpha.\tau$. Then $\mathtt{out}\, v_1 = (\mathtt{out}\, v_1')^{f[\omega]}$. But for any i, $(\mathtt{out}\, v_1')^{f[i]} = \mathtt{out}(v_1'^{f[i]}) \preceq \mathtt{out}\, v_2$. Thus $\mathtt{out}\, v_1\ R\ \mathtt{out}\, v_2$ and hence $v_1\ R_{val}\ v_2$. □

Corollary 2.12 (Compactness) *Suppose f halts. If $e^{f[\omega]}$ halts (and is closed and well-typed) then there exists j such that $e^{f[j]}$ halts.*

Proof. Suppose, for contradiction,[4] that $e^{f[\omega]}\!\downarrow$ and for all j, $e^{f[j]}\!\uparrow$. Then for all j, $e^{f[j]} \preceq \bot_\tau$. By Lemma 2.11, $e^{f[\omega]} \preceq \bot_\tau$ and hence $e^{f[\omega]}\!\uparrow$, but this contradicts the assumption. □

2.4 Admissibility and Strictness

We will restrict attention to the class of *admissible* relations, defined by operational analogues of the chain completeness conditions arising in denotational semantics.

The type-tuple-indexed sets of expression class vectors and relations are defined as follows:

$$ECV_{\tau_1,\ldots,\tau_n} \stackrel{\text{def}}{=} (Exp_{\tau_1}/\approx_{\tau_1}) \times \cdots \times (Exp_{\tau_n}/\approx_{\tau_n})$$
$$Rel_{\tau_1,\ldots,\tau_n} \stackrel{\text{def}}{=} \mathcal{P}(ECV_{\tau_1,\ldots,\tau_n})$$

Definition 2.13 *A relation $R \in Rel_{\tau_1,\ldots,\tau_n}$ is* admissible *if it satisfies the following two conditions:*[5]

- *(***Pointedness***)* $(\bot_{\tau_1}, \ldots, \bot_{\tau_n}) \in R$
- *(***Completeness***) Suppose (for $k = 1, \ldots, n$) $\tau_k', \tau_k'' \in$ Type and $w{:}\tau_k' \to \tau_k'' \vdash e_k : \tau_k$ and $\vdash f_k : (\tau_k' \to \tau_k'') \to \tau_k' \to \tau_k''$. If for all i there exists $j \geq i$ such that $(e_1^{f_1[j]}, \ldots, e_n^{f_n[j]}) \in R$, then $(e_1^{f_1[\omega]}, \ldots, e_n^{f_n[\omega]}) \in R$.*

Definition 2.14 *A relation $R \in Rel_{\tau_1,\ldots,\tau_n}$ is* strict *if whenever $(e_1, \ldots, e_n) \in R$ and for some i, $e_i\!\downarrow$, then for all i, $e_i\!\downarrow$.*

---

[4] If one prefers, a constructive proof can also be derived directly from Lemma 2.10.

[5] The pointedness condition is stated in this manner for simplicity, without regard for constructivity. The theorems in this paper may be carried out constructively if it is replaced by the (constructively stronger) proposition $((\exists i.\, e_i\!\downarrow) \Rightarrow (e_1, \ldots, e_n) \in R) \Rightarrow (e_1, \ldots, e_n) \in R$. These conditions are equivalent in a classical setting.

The lattice properties of the class of relations are necessary for the interpretation of recursive types.

Lemma 2.15 *For any $\tau_1, \ldots, \tau_n \in Type$, the set of strict, admissible relations in $Rel_{\tau_1,\ldots,\tau_n}$ forms a complete lattice, with bottom element $\{(e_1, \ldots, e_n) \in ECV_{\tau_1,\ldots,\tau_n} \mid e_1{\uparrow} \wedge \cdots \wedge e_n{\uparrow}\}$, top element $\{(e_1, \ldots, e_n) \in ECV_{\tau_1,\ldots,\tau_n} \mid (\exists i.\, e_i{\downarrow}) \Rightarrow (\forall i.\, e_i{\downarrow})\}$, meets computed by intersections, and joins computed by intersection of all upper bounds.*

Lemma 2.16 (Fixed Point Induction) *Suppose $R \in Rel_{\tau_1\to\tau_1',\ldots,\tau_n\to\tau_n'}$ is admissible, $F_1, \ldots, F_n$ halt, and (for $1 \le i \le n$) $\vdash F_i : (\tau_i \to \tau_i') \to \tau_i \to \tau_i'$. If $(\lambda x{:}\tau_1.\bot_{\tau_1'}, \ldots, \lambda x{:}\tau_n.\bot_{\tau_n'}) \in R$ and, for all $(f_1, \ldots, f_n) \in R$, $(\lambda x{:}\tau_1.\, F_1 f_1 x, \ldots, \lambda x{:}\tau_n.\, F_n f_n x) \in R$, then $(fix\, F_1, \ldots, fix\, F_n) \in R$.*

Proof. Observe that $w{:}\tau_k \to \tau_k' \vdash w : \tau_k \to \tau_k'$. We show by induction that for all i, $(w^{F_1[i]}, \ldots, w^{F_n[i]}) \in R$. By the first assumption, $(w^{F_1[0]}, \ldots, w^{F_n[0]}) = (fix^0\, F_1, \ldots, fix^0\, F_n) \approx (\lambda x{:}\tau_1.\bot_{\tau_1'}, \ldots, \lambda x{:}\tau_n.\bot_{\tau_n'}) \in R$. Suppose, for induction, that $(w^{F_1[i]}, \ldots, w^{F_n[i]}) = (fix^i\, F_1, \ldots, fix^i\, F_n) \in R$. Then $(w^{F_1[i+1]}, \ldots, w^{F_n[i+1]}) = (fix^{i+1}\, F_1, \ldots, fix^{i+1}\, F_n) \approx (\lambda x{:}\tau_1.F_1(fix^i\, F_1)x, \ldots, \lambda x{:}\tau_n.F_n(fix^i\, F_n)x) \in R$ by the second assumption. Hence, for all i there exists $j \ge i$ (namely i itself) such that $(w^{F_1[j]}, \ldots, w^{F_n[j]}) \in R$. By completeness of R, $(fix\, F_1, \ldots, fix\, F_n) = (w^{F_1[\omega]}, \ldots, w^{F_n[\omega]}) \in R$. □

Notation We write $fix\, g(x{:}\tau_1){:}\tau_2.e$ to mean $fix(\lambda g{:}\tau_1 \to \tau_2.\, \lambda x{:}\tau_1.e)$. Also, when f is of the form $fix\, g(x{:}\tau_1){:}\tau_2.e$, we write f^i to mean $fix^i(\lambda g{:}\tau_1 \to \tau_2.\, \lambda x{:}\tau_1.e)$.

Corollary 2.17 (Fixed Point Induction) *Suppose $R \in Rel_{\tau_1\to\tau_1',\tau_2\to\tau_2'}$ is admissible and (for $i = 1, 2$) $g{:}(\tau_i \to \tau_i'), x{:}\tau_i \vdash e_i : \tau_i'$. If $(\lambda x{:}\tau_1.\bot_{\tau_1'}, \lambda x{:}\tau_2.\bot_{\tau_2'}) \in R$ and, for all $(f_1, f_2) \in R$, $(\lambda x{:}\tau_1.e_1[f_1/g], \lambda x{:}\tau_2.e_2[f_2/g]) \in R$, then $(fix\, g(x{:}\tau_1){:}\tau_1'.e_1, fix\, g(x{:}\tau_2){:}\tau_2'.e_2) \in R$.*

3 Syntactic Minimal Invariance

In a domain setting the solution to a mixed-variance recursive domain equation may be universally characterized as a *minimal invariant* $i : F(D, D) \cong D$ of a bifunctor F over a category of domains and its opposite [6]. The minimality of i ensures that every element of D is the limit of its finite projections; this amounts to the requirement that a certain recursively defined function associated with the equation is the identity. Pitts [19] showed that the existence of the minimal invariant is sufficient for the construction of relations over a recursive domain, and uses this to prove adequacy of a denotational semantics using a logical relations argument.

Following Birkedal and Harper [4], we prove an operational analogue of the minimal invariance condition, called *syntactic minimal invariance*. The key observation is that the finite projections alluded to above are definable in the language, as is their limit, which is a recursively defined function. We then show that this limit is applicatively equivalent to the identity. The argument we give here is an extension to and an improvement on the proof of syntactic minimal invariance given by

$$
\begin{aligned}
\pi_{\alpha} &\stackrel{\text{def}}{=} p_{\alpha} \\
\pi_{\tau_1 \to \tau_2} &\stackrel{\text{def}}{=} \lambda f{:}(\tau_1 \to \tau_2).\, \lambda x{:}\tau_1.\, \pi_{\tau_2}(f(\pi_{\tau_1} x)) \\
\pi_{\forall \alpha.\tau} &\stackrel{\text{def}}{=} \lambda f{:}(\forall \alpha.\tau).\, \Lambda \alpha.\, (\pi_{\tau}[id_{\alpha}/p_{\alpha}])(f[\alpha]) \\
\pi_{\mu \alpha.\tau} &\stackrel{\text{def}}{=} \mathit{fix}\, f(x{:}\mu\alpha.\tau){:}\mu\alpha.\tau.\mathtt{in}_{\mu\alpha.\tau}((\pi_{\tau}[\mu\alpha.\tau, f/\alpha, p_{\alpha}])(\mathtt{out}\, x))) \\
\pi_{1} &\stackrel{\text{def}}{=} \lambda x{:}1.*
\end{aligned}
$$

Fig. 5. Syntactic Projection Functions

Birkedal and Harper. The extension consists of considering general recursive, as well as polymorphic, types, rather than a single, fixed recursive type. This requires a bit more machinery, but proceeds along substantially the same lines as before. The technical improvement is that the argument is streamlined by considering a range of possible "decorations" of terms with syntactic projections, which affords a stronger induction hypothesis (see the proof of Lemma 3.13).

To begin with, we define the syntactic projection function $\pi_{\tau} : \tau \to \tau$ for each type τ as shown in Figure 5. Note that for type variables the syntactic projection functions defer to identified term variables of the form p_{α}. An appropriate projection function is later substituted for p_{α}—the identity in the case of polymorphic variables, and the projection itself in the case of recursive type variables.

Lemma 3.1 *If* $\beta_1, \ldots, \beta_n \vdash \tau$ type *then* $\beta_1, \ldots, \beta_n, p_{\beta_1}{:}\beta_1 \to \beta_1, \ldots, p_{\beta_n}{:}\beta_n \to \beta_n \vdash \pi_{\tau} : \tau \to \tau$.

Lemma 3.2 *The terms* $\pi_{\tau}[\tau', \pi_{\tau'}/\alpha, p_{\alpha}]$ *and* $\pi_{\tau[\tau'/\alpha]}$ *are syntactically identical.*

3.1 Projections Approximate the Identity

It is relatively straightforward to show that the projection π_{τ} applicatively approximates the identity function on type τ. The argument proceeds by an outer induction on the structure of τ, with an inner fixed point induction in the case of recursive types.

Lemma 3.3 *Suppose* $\beta_1, \ldots, \beta_n \vdash \tau$ type*, and for all* $1 \leq i \leq n$, $\vdash v_i \preceq id_{\tau_i} : \tau_i$. *Then* $\pi_{\tau}[\vec{\tau}, \vec{v}/\vec{\beta}, p_{\vec{\beta}}] \preceq id_{\tau[\vec{\tau}/\vec{\beta}]} : \tau[\vec{\tau}/\vec{\beta}] \to \tau[\vec{\tau}/\vec{\beta}]$.

Proof. By induction on τ. Let $\sigma = [\vec{\tau}, \vec{v}/\vec{\beta}, p_{\vec{\beta}}]$.

Case 1: Suppose $\tau = \beta_i$. Then $\pi_{\tau}\sigma = p_{\beta_i}\sigma = v_i$. By assumption, $v_i \preceq id_{\tau_i}$.

Case 2: Suppose $\tau = 1$. Then $\pi_{\tau}\sigma = \lambda x{:}1.* \approx id_1$.

Case 3: Suppose $\tau = \tau_1 \to \tau_2$, and let $\tau_1' \to \tau_2' = (\tau_1 \to \tau_2)[\vec{\tau}/\vec{\beta}]$. Then $\pi_{\tau}\sigma = \lambda f{:}(\tau_1' \to \tau_2').\, \lambda x{:}\tau_1'.\, (\pi_{\tau_2}\sigma)(f((\pi_{\tau_1}\sigma)x))$. Suppose $\vdash v : \tau_1' \to \tau_2'$. We wish to show that $\lambda x{:}\tau_1'.\, (\pi_{\tau_2}\sigma)(v((\pi_{\tau_1}\sigma)x)) \preceq_{val} v$. Suppose $\vdash v' : \tau_1'$. Then it suffices to show

that $(\pi_{\tau_2}\sigma)(v((\pi_{\tau_1}\sigma)v')) \preceq v\,v'$. By induction, $\pi_{\tau_1}\sigma \preceq id_{\tau_1'}$, so $(\pi_{\tau_1}\sigma)v' \preceq v'$. By congruence, $v((\pi_{\tau_1}\sigma)v') \preceq v\,v'$. By induction, $\pi_{\tau_2}\sigma \preceq id_{\tau_2'}$, so $(\pi_{\tau_2}\sigma)(v((\pi_{\tau_1}\sigma)v')) \preceq v((\pi_{\tau_1}\sigma)v') \preceq v\,v'$.

Case 4: Suppose $\tau = \forall\alpha.\tau_1$ and let $\tau_1' = \tau_1[\vec{\tau}/\vec{\beta}]$. Then $\pi_\tau\sigma = \lambda f{:}(\forall\alpha.\tau_1').\,\Lambda\alpha.\,(\pi_{\tau_1}\sigma[id_\alpha/p_\alpha])(f[\alpha])$. Suppose $\vdash v : \forall\alpha.\tau_1'$. We wish to show that $\Lambda\alpha.\,(\pi_{\tau_1}\sigma[id_\alpha/p_\alpha])(v[\alpha]) \preceq v$. Suppose $\vdash \tau'$ type. Then it suffices to show that $(\pi_{\tau_1}\sigma[\tau', id_{\tau'}/\alpha, p_\alpha])(v[\tau']) \preceq v[\tau']$. Certainly $id_{\tau'} \preceq id_{\tau'}$, so by induction, $\pi_{\tau_1}\sigma[\tau', id_{\tau'}/\alpha, p_\alpha] \preceq id_{\tau_1'[\tau'/\alpha]}$, and the result follows.

Case 5: Suppose $\tau = \mu\alpha.\tau_1$ and let $\tau_1' = \tau_1[\vec{\tau}/\vec{\beta}]$. Define $\pi = \pi_\tau\sigma$ and note that π is a *fix* value. We show by induction that for all i, $\pi^i \preceq id_{\mu\alpha.\tau_1'}$. The result follows by the Least Upper Bound lemma.

The base case is trivial. For the inductive case, assume that $\pi^i \preceq id_{\mu\alpha.\tau_1'}$. Suppose $\vdash v : \mu\alpha.\tau_1'$. We wish to show that $\mathtt{in}_{\mu\alpha.\tau_1'}((\pi_{\tau_1}\sigma[\mu\alpha.\tau_1', \pi^i/\alpha, p_\alpha])(\mathtt{out}\,v)) \preceq v$. Since $\pi^i \preceq id_{\mu\alpha.\tau_1'}$, by the outer induction, $\pi_{\tau_1}\sigma[\mu\alpha.\tau_1', \pi^i/\alpha, p_\alpha] \preceq id_{\tau_1'[\mu\alpha.\tau_1'/\alpha]}$. Thus $(\pi_{\tau_1}\sigma[\mu\alpha.\tau_1', \pi^i/\alpha, p_\alpha])(\mathtt{out}\,v) \preceq \mathtt{out}\,v$. By congruence, $\mathtt{in}_{\mu\alpha.\tau_1'}((\pi_{\tau_1}\sigma[\mu\alpha.\tau_1', \pi^i/\alpha, p_\alpha])(\mathtt{out}\,v)) \preceq \mathtt{in}_{\mu\alpha.\tau_1'}(\mathtt{out}\,v)$. But, v must be of the form $\mathtt{in}_{\mu\alpha.\tau_1'}v'$ so $\mathtt{in}_{\mu\alpha.\tau_1'}(\mathtt{out}\,v) \mapsto \mathtt{in}_{\mu\alpha.\tau'}v' = v$. Thus $\mathtt{in}_{\mu\alpha.\tau_1'}(\mathtt{out}\,v) \preceq v$. □

3.2 Projections Dominate the Identity

To prove that projections applicatively dominate the identity requires a slightly more complex argument. Intuitively, the evaluation of $\pi_\tau e$ may result in a term containing many further occurrences of projections at arbitrary places in the term. Call each of these a *decoration* of the underlying term by some number of projections, and note that $\pi_\tau e$ is one such decoration. We show that an expression is applicatively dominated by all of its decorations, from which the result follows directly.

The decoration of a term is determined by its type. To account for substitution during type checking, we must consider all possible ways that the type of a term may arise as a substitution instance of another, which we call *factorings*. Moreover, for the sake of the induction we must consider all possible compositions of projections based on factorings. This leads to the following definitions.

Definition 3.4

- *When τ is a type and σ is a substitution on types, we say that (τ, σ) is a* factoring *of τ' (written $\tau' \lhd (\tau, \sigma)$) if $\tau\sigma = \tau'$.*
- *If $\varphi = (\tau, [\vec{\tau}/\vec{\alpha}])$ is a factoring and the free variables of $\tau[\vec{\tau}/\vec{\alpha}]$ are $\vec{\beta}$, then π_φ is defined to be $\pi_\tau[\vec{\tau}, id_{\vec{\tau}}, id_{\vec{\beta}}/\vec{\alpha}, p_{\vec{\alpha}}, p_{\vec{\beta}}] : \tau[\vec{\tau}/\vec{\alpha}] \to \tau[\vec{\tau}/\vec{\alpha}]$.*
- *We write $\tau \lhd \varphi_1, \ldots \varphi_n$ to mean $\tau \lhd \varphi_i$ for every $1 \le i \le n$, and we write $\pi_{\varphi_1,\ldots,\varphi_n}[e]$ to mean $\pi_{\varphi_1}(\cdots(\pi_{\varphi_n}e))$,*

Proposition 3.5 *If φ factors τ then for any type substitution σ there exists a factoring φ' of $\tau\sigma$ such that $\pi_\varphi\sigma = \pi_{\varphi'}$.*

The possible decorations for a term is determined by a syntax-directed collection of rules.

Definition 3.6

- *The decoration relation* $\Gamma \vdash e \lhd e' : \tau$ *is defined as follows:*

$$\frac{}{\Gamma \vdash x \lhd \pi_{\vec{\varphi}}[x] : \tau}\ (\Gamma(x) = \tau \text{ and } \tau \lhd \vec{\varphi}) \qquad \frac{}{\Gamma \vdash * \lhd \pi_{\vec{\varphi}}[*] : 1}\ (1 \lhd \vec{\varphi})$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e \lhd \bar{e} : \tau_2 \quad \Gamma \vdash \tau_1 \text{ type} \quad \Gamma \vdash \tau_2 \text{ type}}{\Gamma \vdash \lambda x{:}\tau_1.e \lhd \pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}] : \tau_1 \to \tau_2}\ (x \notin \mathrm{Dom}(\Gamma) \text{ and } \tau_1 \to \tau_2 \lhd \vec{\varphi})$$

$$\frac{\Gamma \vdash e_1 \lhd \bar{e}_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 \lhd \bar{e}_2 : \tau_1}{\Gamma \vdash e_1 e_2 \lhd \pi_{\vec{\varphi}}[\bar{e}_1 \bar{e}_2] : \tau_2}\ (\tau_2 \lhd \vec{\varphi})$$

$$\frac{\Gamma, \alpha \vdash e \lhd \bar{e} : \tau}{\Gamma \vdash \Lambda\alpha.e \lhd \pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}] : \forall\alpha.\tau}\ (\alpha \notin \mathrm{Dom}(\Gamma) \text{ and } \forall\alpha.\tau \lhd \vec{\varphi})$$

$$\frac{\Gamma \vdash e \lhd \bar{e} : \forall\alpha.\tau \quad \Gamma \vdash \tau' \text{ type}}{\Gamma \vdash e[\tau'] \lhd \pi_{\vec{\varphi}}[\bar{e}[\tau']] : \tau[\tau'/\alpha]}\ (\tau[\tau'/\alpha] \lhd \vec{\varphi})$$

$$\frac{\Gamma \vdash e \lhd \bar{e} : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \mathtt{in}_{\mu\alpha.\tau} e \lhd \pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau} \bar{e}] : \mu\alpha.\tau}\ (\mu\alpha.\tau \lhd \vec{\varphi})$$

$$\frac{\Gamma \vdash e \lhd \bar{e} : \mu\alpha.\tau}{\Gamma \vdash \mathtt{out}\, e \lhd \pi_{\vec{\varphi}}[\mathtt{out}\, \bar{e}] : \tau[\mu\alpha.\tau/\alpha]}\ (\tau[\mu\alpha.\tau/\alpha] \lhd \vec{\varphi})$$

- *Since terms have unique types, we may write* $\Gamma \vdash e \lhd e'$ *(or* $e \lhd e'$ *for closed terms) without ambiguity.*

The decoration relation is compositional in the sense that it commutes with substitution.

Proposition 3.7

- *If* $\Gamma \vdash e \lhd \bar{e} : \tau$ *then* $\Gamma \vdash e : \tau$ *and* $\Gamma \vdash \bar{e} : \tau$.
- *If* $\Gamma, \alpha, \Gamma' \vdash e \lhd \bar{e} : \tau$ *and* $\Gamma \vdash \tau'$ type *then* $\Gamma, (\Gamma'[\tau'/\alpha]) \vdash e[\tau'/\alpha] \lhd \bar{e}[\tau'/\alpha] : \tau[\tau'/\alpha]$.
- *If* $\Gamma, x{:}\tau_1, \Gamma' \vdash e \lhd \bar{e} : \tau_2$ *and* $\Gamma \vdash e' \lhd \bar{e}' : \tau_1$ *then* $\Gamma, \Gamma' \vdash e[e'/x] \lhd \bar{e}[\bar{e}'/x] : \tau_2$.

The decoration of a value is not a value, but is tantamount to a value.

Lemma 3.8 *If* $\vdash v : \tau$ *then* $\pi_\tau v$ *halts.*

Proof. By induction on the structure of v, with an inner case analysis on τ. Note that τ cannot be α.

Case 1: Suppose τ is $\tau_1 \to \tau_2$, $\forall\alpha.\tau'$ or 1. Then $\pi_\tau v$ halts after a single step.

Case 2: Suppose τ is $\mu\alpha.\tau'$. Then v has the form $\mathtt{in}_\tau v'$, for some v' where $\vdash v' : \tau'[\tau/\alpha]$. Hence:

$$\begin{aligned}
\pi_\tau v = \;& (\mathit{fix}\, f(x{:}\tau){:}\tau.\, \mathtt{in}_\tau((\pi_{\tau'}[\tau, f/\alpha, p_\alpha])(\mathtt{out}\, x)))\, (\mathtt{in}_\tau v') \\
\mapsto^* \;& \mathtt{in}_\tau((\pi_{\tau'}[\tau, \pi_\tau/\alpha, p_\alpha])(\mathtt{out}(\mathtt{in}_\tau v'))) \\
= \;& \mathtt{in}_\tau(\pi_{\tau'[\tau/\alpha]}\, (\mathtt{out}(\mathtt{in}_\tau v'))) \\
\mapsto \;& \mathtt{in}_\tau(\pi_{\tau'[\tau/\alpha]}\, v')
\end{aligned}$$

By induction $\pi_{\tau'[\tau/\alpha]}\, v'$ halts, and therefore $\pi_\tau v$ halts as well. □

Corollary 3.9 *If $v \lhd \bar{e}$ then $\bar{e}$ halts.*

Proof. Observe that $\bar{e}$ must be of the form $\pi_{\vec{\varphi}}\, v'$. The result follows by induction on $\vec{\varphi}$, using Lemma 3.8. □

The next three lemmas are preparation for the coinduction in the proof of Corollary 3.14. We rely on the flexibility in choosing decorations in each case.

Lemma 3.10 *Suppose $x{:}\tau_1 \vdash e : \tau_2$ and $\vdash v : \tau_1$, and suppose $\vec{\varphi} \rhd \tau_1 \to \tau_2$, and $\bar{e} \rhd e$, and $\bar{e}' \rhd v$. Then there exists $\bar{e}''$ such that $\pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}]\bar{e}' \approx \bar{e}''$ and $\bar{e}'' \rhd e[v/x]$.*

Proof. By induction on $\vec{\varphi}$. Note that by Corollary 3.9 any decoration of a value halts, so we may employ beta reduction when any such appears as an argument.

Case 1: Suppose $\vec{\varphi} = \epsilon$. Then, using beta-reduction:

$$\begin{aligned}
\pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}]\bar{e}' &= (\lambda x{:}\tau_1.\bar{e})\bar{e}' \\
&\approx \bar{e}[\bar{e}'/x]
\end{aligned}$$

and $\bar{e}[\bar{e}'/x] \rhd e[v/x]$.

Case 2: Suppose $\vec{\varphi} = (\alpha, [\tau_1 \to \tau_2/\alpha]), \vec{\varphi}'$. Then, using beta-reduction:

$$\begin{aligned}
\pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}]\bar{e}' &= (id_{\tau_1\to\tau_2}\, \pi_{\vec{\varphi}'}[\lambda x{:}\tau_1.\bar{e}])\, \bar{e}' \\
&\approx \pi_{\vec{\varphi}'}[\lambda x{:}\tau_1.\bar{e}]\, \bar{e}'
\end{aligned}$$

The result follows immediately by induction.

Case 3: Suppose $\vec{\varphi} = (\tau_1' \to \tau_2', \sigma), \vec{\varphi}'$. Then $(\tau_1', \sigma) \rhd \tau_1$ and $(\tau_2', \sigma) \rhd \tau_2$. There-

fore, using beta-reduction:

$$\begin{aligned}\pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}]\bar{e}' &= (\pi_{(\tau_1'\to\tau_2',\sigma)}\,\pi_{\vec{\varphi}'}[\lambda x{:}\tau_1.\bar{e}])\,\bar{e}' \\ &\approx \pi_{(\tau_2',\sigma)}\,(\pi_{\vec{\varphi}'}[\lambda x{:}\tau_1.\bar{e}]\,(\pi_{(\tau_1',\sigma)}\bar{e}'))\end{aligned}$$

Since $\pi_{(\tau_1',\sigma)}\bar{e}' \rhd v$, by induction and congruence the latter is equivalent to $\pi_{(\tau_2',\sigma)}\,\bar{e}''$ for some $\bar{e}'' \rhd e[v/x]$. Finally, $\pi_{(\tau_2',\sigma)}\bar{e}'' \rhd e[v/x]$. □

Lemma 3.11 *Suppose* $\alpha \vdash e : \tau$ *and* $\vdash \tau'\mathrm{type}$*, and suppose* $\vec{\varphi} \rhd \forall\alpha.\tau$ *and* $\alpha \vdash \bar{e} \rhd e$*. Then there exists* $\bar{e}'$ *such that* $\pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}][\tau'] \approx \bar{e}'$ *and* $\bar{e}' \rhd e[\tau'/\alpha]$*.*

Proof. By induction on $\vec{\varphi}$. Note that by Corollary 3.9 any decoration of a value halts, so we may employ beta reduction when any such appears as an argument.

Case 1: Suppose $\vec{\varphi} = \epsilon$. Then, using beta-reduction:

$$\begin{aligned}\pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}][\tau'] &= (\Lambda\alpha.\bar{e})[\tau'] \\ &\approx \bar{e}[\tau'/\alpha]\end{aligned}$$

and $\bar{e}[\tau'/\alpha] \rhd e[\tau'/\alpha]$.

Case 2: Suppose $\vec{\varphi} = (\beta, [\forall\alpha.\tau/\beta]), \vec{\varphi}'$. Then, using beta-reduction:

$$\begin{aligned}\pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}][\tau'] &= (id_{\forall\alpha.\tau}\,\pi_{\vec{\varphi}'}[\Lambda\alpha.\bar{e}])[\tau'] \\ &\approx \pi_{\vec{\varphi}'}[\Lambda\alpha.\bar{e}][\tau']\end{aligned}$$

The result follows immediately by induction.

Case 3: Suppose $\vec{\varphi} = (\forall\alpha.\tau'', \sigma), \vec{\varphi}'$. Then, using beta-reduction:

$$\begin{aligned}\pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}][\tau'] &= (\pi_{(\forall\alpha.\tau'',\sigma)}\,\pi_{\vec{\varphi}'}[\Lambda\alpha.\bar{e}])[\tau'] \\ &\approx (\pi_{(\tau'',\sigma)}[\tau', id_{\tau'}/\alpha, p_\alpha])\,(\pi_{\vec{\varphi}'}[\Lambda\alpha.\bar{e}][\tau']) \\ &= \pi_{(\tau'',\sigma[\tau'/\alpha])}\,(\pi_{\vec{\varphi}'}[\Lambda\alpha.\bar{e}][\tau'])\end{aligned}$$

By induction and congruence, the latter is equivalent to $\pi_{(\tau'',\sigma[\tau'/\alpha])}\bar{e}'$ for some $\bar{e}' \rhd e[\tau'/\alpha]$. Since $(\tau'', \sigma[\tau'/\alpha])$ factors $\tau[\tau'/\alpha]$, we conclude $\pi_{(\tau'',\sigma[\tau'/\alpha])}\bar{e}' \rhd e[\tau'/\alpha]$. □

Lemma 3.12 *Suppose* $\vdash v : \tau[\mu\alpha.\tau/\alpha]$*, and suppose* $\vec{\varphi} \rhd \mu\alpha.\tau$ *and* $\bar{e} \rhd v$*. Then there exists* $\bar{e}'$ *such that* $\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}] \approx \mathtt{in}_{\mu\alpha.\tau}\bar{e}'$ *and* $\bar{e}' \rhd v$*.*

Proof. By induction on $\vec{\varphi}$. Note that by Corollary 3.9 any decoration of a value halts, so we may employ beta reduction when any such appears as an argument. Also note that $\mathtt{in}_{\mu\alpha.\tau}e\!\downarrow$ whenever $e\!\downarrow$.

Case 1: Suppose $\vec{\varphi} = \epsilon$. Then the result is immediate; choosing $\bar{e}' = \bar{e}$.

Case 2: Suppose $\vec{\varphi} = (\alpha, [\mu\alpha.\tau/\alpha]), \vec{\varphi}'$. Then, using induction, congruence, and beta-reduction:

$$\begin{aligned}\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}] &= id_{\mu\alpha.\tau}\pi_{\vec{\varphi}'}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}] \\ &\approx id_{\mu\alpha.\tau}(\mathtt{in}_{\mu\alpha.\tau}\bar{e}') \\ &\approx \mathtt{in}_{\mu\alpha.\tau}\bar{e}'\end{aligned}$$

for some $\bar{e}' \rhd v$.

Case 3: Suppose $\vec{\varphi} = (\mu\alpha.\tau', \sigma), \vec{\varphi}'$. Then, using induction, congruence, and beta-reduction:

$$\begin{aligned}\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}] &= \pi_{(\mu\alpha.\tau',\sigma)}\,\pi_{\vec{\varphi}'}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}] \\ &\approx \pi_{(\mu\alpha.\tau',\sigma)}\,(\mathtt{in}_{\mu\alpha.\tau}\bar{e}') \\ &\approx \mathtt{in}_{\mu\alpha.\tau}((\pi_{(\tau',\sigma)}[\mu\alpha.\tau, \pi_{(\mu\alpha.\tau',\sigma)}/\alpha, p_\alpha])\,(\mathtt{out}(\mathtt{in}_{\mu\alpha.\tau}\bar{e}'))) \\ &= \mathtt{in}_{\mu\alpha.\tau}(\pi_{(\tau'[\mu\alpha.\tau'/\alpha],\sigma)}\,(\mathtt{out}(\mathtt{in}_{\mu\alpha.\tau}\bar{e}'))) \\ &\approx \mathtt{in}_{\mu\alpha.\tau}(\pi_{(\tau'[\mu\alpha.\tau'/\alpha],\sigma)}\,\bar{e}')\end{aligned}$$

for some $\bar{e}' \rhd v$. Finally, $(\tau'[\mu\alpha.\tau'/\alpha], \sigma) \rhd \tau[\mu\alpha.\tau/\alpha]$ so $\pi_{(\tau'[\mu\alpha.\tau'/\alpha],\sigma)}\bar{e}' \rhd v$. □

It is crucial to the argument that decoration respect evaluation.

Lemma 3.13 *If $e_1 \mapsto e_2$ then for all $\bar{e}_1 \rhd e_1$ there exists $\bar{e}_2 \rhd e_2$ such that $\bar{e}_1 \approx \bar{e}_2$.*

Proof. By induction on e_1.

Case 1: Suppose e_1 is $\lambda x{:}\tau_1.e_1'$, $\Lambda\alpha.e_1'$, or $*$. Then $e_1 \not\mapsto e_2$.

Case 2: Suppose e_1 is $\mathtt{in}_{\mu\alpha.\tau}e_1'$. Then e_2 is $\mathtt{in}_{\mu\alpha.\tau}e_2'$ where $e_1' \mapsto e_2'$. Also, $\bar{e}_1$ is of the form $\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}_1']$ for some $\vec{\varphi} \rhd \mu\alpha.\tau$ and $\bar{e}_1' \rhd e_1'$. By induction there exists $\bar{e}_2' \rhd e_2'$ such that $\bar{e}_1' \approx \bar{e}_2'$. Then $\bar{e}_1 \approx \pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}_2']$ by congruence and $e_2 \lhd \pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}_2']$.

Case 3: Suppose e_1 is $e_1'e$ where e_1' is not a value. Then e_2 is $e_2'e$ where $e_1' \mapsto e_2'$. Let $\vdash e_1 : \tau$. Then $\bar{e}_1$ is of the form $\pi_{\vec{\varphi}}[\bar{e}_1'\bar{e}]$ for some $\vec{\varphi} \rhd \tau$, $\bar{e}_1' \rhd e_1'$, and $\bar{e} \rhd e$. By induction there exists $\bar{e}_2' \rhd e_2'$ such that $\bar{e}_1' \approx \bar{e}_2'$. Then $\bar{e}_1 \approx \pi_{\vec{\varphi}}[\bar{e}_2'\bar{e}]$ by congruence and $e_2 \lhd \pi_{\vec{\varphi}}[\bar{e}_2'\bar{e}]$. The cases where e_1 is $v\,e_1'$, $e_1'[\tau]$, or $\mathtt{out}\,e_1'$, where e_1' is not a value, are similar.

Case 4: Suppose e_1 is $(\lambda x{:}\tau_1.e)v$, where $x{:}\tau_1 \vdash e : \tau_2$. Then e_2 is $e[v/x]$. Also, $\bar{e}_1$ is of the form $\pi_{\vec{\varphi}'}[\pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}]\bar{e}']$ for some $\vec{\varphi} \rhd \tau_1 \to \tau_2$, $\vec{\varphi}' \rhd \tau_2$, $x{:}\tau_1 \vdash \bar{e} \rhd e$, and $\bar{e}' \rhd v$. By Lemma 3.10, $\pi_{\vec{\varphi}}[\lambda x{:}\tau_1.\bar{e}]\bar{e}' \approx \bar{e}''$ for some $\bar{e}'' \rhd e_2$. By congruence, $\bar{e}_1 \approx \pi_{\vec{\varphi}'}[\bar{e}'']$ and $\pi_{\vec{\varphi}'}[\bar{e}''] \rhd e_2$.

Case 5: Suppose e_1 is $\mathtt{out}(\mathtt{in}_{\mu\alpha.\tau}v)$. Then e_2 is v. Also, $\bar{e}_1$ is of the form $\pi_{\vec{\varphi}'}[\mathtt{out}(\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}'])]$ for some $\vec{\varphi} \rhd \mu\alpha.\tau$, $\vec{\varphi}' \rhd \tau[\mu\alpha.\tau/\alpha]$, and $\bar{e} \rhd v$. By Lemma 3.12, $\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}] \approx \mathtt{in}_{\mu\alpha.\tau}\bar{e}'$ for some $\bar{e}' \rhd v$. By congruence and beta-reduction, $\mathtt{out}(\pi_{\vec{\varphi}}[\mathtt{in}_{\mu\alpha.\tau}\bar{e}]) \approx \bar{e}'$. Finally, by congruence, $\bar{e}_1 \approx \pi_{\vec{\varphi}'}[\bar{e}']$ and

$\pi_{\vec{\varphi}'}[\bar{e}'] \rhd v$.

Case 6: Suppose e_1 is $(\Lambda\alpha.e)[\tau']$. Then e_2 is $e[\tau'/\alpha]$. Let $\vdash \Lambda\alpha.e : \forall\alpha.\tau$ and note that τ' is closed (since $\vdash \tau'$ type). Then $\bar{e}_1$ is of the form $\pi_{\vec{\varphi}'}[\pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}][\tau']]$ for some $\vec{\varphi} \rhd \forall\alpha.\tau$, $\vec{\varphi}' \rhd \tau[\tau'/\alpha]$, and $\alpha \vdash \bar{e} \rhd e$. By Lemma 3.11, $\pi_{\vec{\varphi}}[\Lambda\alpha.\bar{e}][\tau'] \approx \bar{e}'$ for some $\bar{e}' \rhd e_2$. By congruence, $\bar{e}_1 \approx \pi_{\vec{\varphi}'}[\bar{e}']$ and $\pi_{\vec{\varphi}'}[\bar{e}'] \rhd e_2$. □

Corollary 3.14 *If $e \lhd \bar{e}$ then $e \preceq \bar{e}$.*

Proof. First we establish that if $e \lhd \bar{e}$ and $e{\downarrow}$ then $\bar{e}{\downarrow}$. Suppose $e \mapsto^* v$. By Lemma 3.13 and an easy induction, there exists $\bar{e}' \rhd v$ such that $\bar{e} \approx \bar{e}'$. By Corollary 3.9, $\bar{e}'{\downarrow}$, and hence $\bar{e}{\downarrow}$.

The proof now proceeds by coinduction (Lemma 2.9). The first condition has just been established; the others are immediate from the compositionality of decoration (Proposition 3.7). □

The main theorem of this section states that the projections associated to each type are the identity at that type. This expresses the universal property of recursive types in an operational setting.

Theorem 3.15 (Syntactic Minimal Invariance) *Suppose $\Gamma \vdash \tau$ type, and let σ be a substitution such that for all $\alpha \in \mathrm{Dom}(\Gamma)$, $\vdash \sigma(\alpha)$ type and $\sigma(p_\alpha) \approx id_{\sigma(\alpha)}$. Then $\sigma(\pi_\tau) \approx id_{\sigma(\tau)}$.*

Proof. By Lemma 3.3, $\sigma(\pi_\tau) \preceq id_{\sigma(\tau)}$. Note that $id_{\sigma(\tau)}$ and $\sigma(\pi_\tau)$ halt. Thus, to show $id_{\sigma(\tau)} \preceq \sigma(\pi_\tau)$ it is sufficient to show that $v \preceq \sigma(\pi_\tau)v$ for all v such that $\vdash v : \sigma(\tau)$. Suppose $\vdash v : \sigma(\tau)$. Clearly (τ, σ) is a factoring of $\sigma(\tau)$, so $v \lhd (\sigma(\pi_\tau))v$. By Corollary 3.14, $v \preceq (\sigma(\pi_\tau))v$. □

4 The Logical Interpretation

The method of logical relations associates a relational action to each type constructor in such a way that (a) every type is assigned a relational interpretation, and (b) every well-typed term stands in the relation assigned to its type. In the presence of impredicative polymorphism and unrestricted recursion the assignment of the relational action requires a combination of Girard's Method [8] and Pitts's analysis of relational properties of domains [19], adapted to the operational setting [4].

4.1 Construction of the Relational Interpretation

The type-pair-indexed, partially ordered sets of admissible relations and birelations are defined as follows. Admissible relations are ordered by inclusion as usual, and $(-)^{\mathrm{op}}$ reverses a set's ordering. For notational convenience, we restrict our attention to binary relations.

$$ARel_{\tau_1,\tau_2} \stackrel{\mathrm{def}}{=} \{R \subseteq ECV_{\tau_1,\tau_2} \mid R \text{ is strict and admissible}\}$$
$$Birel_{\tau_1,\tau_2} \stackrel{\mathrm{def}}{=} ARel^{\mathrm{op}}_{\tau_1,\tau_2} \times ARel_{\tau_1,\tau_2}$$

Note that $ARel_{\tau_1,\tau_2}$ and $Birel_{\tau_1,\tau_2}$ both form complete lattices.

Definition 4.1 *Suppose S is a set of type variables. A* type environment *over S is a function from S to pairs of well-formed (closed) types. A* relation environment *over S is a function from S to strict, admissible relations and a* birelation environment *over S is a function from S to birelations. A relation environment χ over S* respects *a type environment δ over S if, for all $\alpha \in S$, $\chi(\alpha) \in ARel_{\delta(\alpha)}$. Similarly, a birelation environment η over S* respects *a type environment δ over S if, for all $\alpha \in S$, $\eta(\alpha) \in Birel_{\delta(\alpha)}$. Relation and birelation environments are ordered pointwise.*

Definition 4.2 *If S is a set of type variables, then $TEnv_S$ is defined to be the set of type environments over S. We will use contexts as sets of type variables by ignoring their value variables. If δ is a type environment, then $REnv_\delta$ is defined to be the set of relation environments over* $\mathrm{Dom}(\delta)$ *that respect δ, and $BEnv_\delta$ is defined to be the set of birelation environments over* $\mathrm{Dom}(\delta)$ *that respect δ.*

Notation Type environments are used as pairs of substitutions over types (resulting in pairs of types) in the obvious manner. When δ is a type environment over S, we also write δ_{left} and δ_{right} for the substitutions returning the left and right components of $\delta(\alpha)$ on each $\alpha \in S$.

Notation Suppose η is a birelation environment and, for all $\alpha \in \mathrm{Dom}(\eta)$, $\eta(\alpha) = (R^-_\alpha, R^+_\alpha)$. Then η^{op} is the birelation environment defined by $\eta^{\mathrm{op}}(\alpha) \stackrel{\mathrm{def}}{=} (R^+_\alpha, R^-_\alpha)$, and η^+ and η^- are the relation environments defined by $\eta^\pm(\alpha) \stackrel{\mathrm{def}}{=} R^\pm_\alpha$.

We can now define the primary tool for building logical relations:

Definition 4.3 *Suppose $\Gamma \vdash \tau$ type, $\delta \in TEnv_\Gamma$, and $\eta \in BEnv_\delta$. Then the* relational interpretation *of τ, written $[\![\tau]\!]^\delta_\eta$ (and intended to belong to $ARel_{\delta(\tau)}$, according to Lemma 4.4), is defined as in Figure 6.*

In the definition of the relational interpretation we permit η to be any member of $BEnv_\delta$; in particular, we permit η^- and η^+ to differ. However, we are ultimately interested in the logical interpretation only in the case when $\eta = \eta^{\mathrm{op}}$. With the exception of a few early lemmas, we will restrict our attention to that case.

Lemma 4.4 *Suppose $\Gamma \vdash \tau$ type, $\delta \in TEnv_\Gamma$, and $\eta \in BEnv_\delta$. Then $[\![\tau]\!]^\delta_\eta$ is well-defined, respects applicative equivalence, and belongs to $ARel_{\delta(\tau)}$. Moreover, if $\eta' \in BEnv_\delta$ and $\eta \sqsubseteq \eta'$ then $[\![\tau]\!]^\delta_\eta \sqsubseteq [\![\tau]\!]^\delta_{\eta'}$.*

Proof. By induction on τ. □

Definition 4.5 *Suppose $\tau, \tau' \in Type$ and $R_1, R_2 \in Rel_{\tau,\tau'}$. Then $f, f' : R_1 \sqsubseteq R_2$ if $\vdash f : \tau \to \tau$ and $\vdash f' : \tau' \to \tau'$ and (for all $e \in Exp_\tau$, $e' \in Exp_{\tau'}$) $e\ R_1\ e'$ implies $f\,e\ R_2\ f'\,e'$.*

Lemma 4.6 *Suppose $\tau, \tau' \in Type$, $R_1, R_2 \in Rel_{\tau,\tau'}$, and R_2 is admissible. Then the set $\{(f, f') \mid f, f' : R_1 \sqsubseteq R_2\}$ is admissible.*

Proof. Let $S = \{(f, f') \mid f, f' : R_1 \sqsubseteq R_2\}$. Suppose $(e, e') \in R_1$. Then $\bot_{\tau\to\tau}\, e \approx \bot_\tau$ and $\bot_{\tau'\to\tau'}\, e' \approx \bot_{\tau'}$. Therefore $(\bot_{\tau\to\tau} e, \bot_{\tau'\to\tau'} e') \in R_2$, since R_2 is pointed. Hence $\bot_{\tau\to\tau}, \bot_{\tau'\to\tau'} : R_1 \sqsubseteq R_2$, so S is pointed.

$$
\begin{array}{lll}
[\![\alpha]\!]^\delta_\eta & \stackrel{\text{def}}{=} & \eta^+(\alpha) \\
[\![1]\!]^\delta_\eta & \stackrel{\text{def}}{=} & \{(e_1,e_2) \in ECV_{1,1} \mid e_1\!\downarrow \Leftrightarrow e_2\!\downarrow\} \\
[\![\tau_1 \to \tau_2]\!]^\delta_\eta & \stackrel{\text{def}}{=} & \{(e_1,e_2) \in ECV_{\delta(\tau_1\to\tau_2)} \mid \\
 & & \qquad e_1\!\downarrow \Leftrightarrow e_2\!\downarrow \ \wedge \\
 & & \qquad \forall e_1', e_2' \in ECV_{\delta(\tau_1)}.\, (e_1', e_2') \in [\![\tau_1]\!]^\delta_{\eta^{\text{op}}} \Rightarrow (e_1 e_1', e_2 e_2') \in [\![\tau_2]\!]^\delta_\eta\} \\
[\![\forall\alpha.\tau]\!]^\delta_\eta & \stackrel{\text{def}}{=} & \{(e_1,e_2) \in ECV_{\delta(\forall\alpha.\tau)} \mid \\
 & & \qquad e_1\!\downarrow \Leftrightarrow e_2\!\downarrow \ \wedge \\
 & & \qquad \forall \tau_1,\tau_2 \in \mathit{Type}.\, \forall R \in ARel_{\tau_1,\tau_2}.\, (e_1[\tau_1], e_2[\tau_2]) \in [\![\tau]\!]^{\delta[\alpha\mapsto(\tau_1,\tau_2)]}_{\eta[\alpha\mapsto(R,R)]}\} \\
 & & \quad \text{where } \alpha \notin \mathrm{Dom}(\delta) \\
[\![\mu\alpha.\tau]\!]^\delta_\eta & \stackrel{\text{def}}{=} & \Delta^+ \\
 & & \quad \text{where } (\Delta^-,\Delta^+) \quad = \mathit{lfp}\, \Psi^{\S} \\
 & & \quad \text{and } \Psi^{\S}(R^-,R^+) = (\Psi_{\eta^{\text{op}}}(R^+,R^-), \Psi_\eta(R^-,R^+)) \\
 & & \quad \text{and } \Psi_\eta(R^-,R^+) = \{(e_1,e_2) \in ECV_{\delta(\mu\alpha.\tau)} \mid \\
 & & \qquad\qquad (\mathtt{out}\, e_1, \mathtt{out}\, e_2) \in [\![\tau]\!]^{\delta[\alpha\mapsto\delta(\mu\alpha.\tau)]}_{\eta[\alpha\mapsto(R^-,R^+)]}\} \\
 & & \quad \text{and } \alpha \notin \mathrm{Dom}(\delta)
\end{array}
$$

Fig. 6. The Relational Interpretation

Let $w{:}\tau_1 \to \tau_2 \vdash g : \tau \to \tau$ and $w{:}\tau_1 \to \tau_2 \vdash g' : \tau' \to \tau'$ and let $\vdash h : (\tau_1 \to \tau_2) \to \tau_1 \to \tau_2$ and $\vdash h' : (\tau_1' \to \tau_2') \to \tau_1' \to \tau_2'$. Suppose for all i there exists $j \geq i$ such that $(g^{h[j]}, g'^{h'[j]}) \in S$. We wish to show that $(g^{h[\omega]}, g'^{h'[\omega]}) \in S$. Suppose $(e, e') \in R_1$. Then, recalling the definition of S, for all i there exists $j \geq i$ such that $(g^{h[j]}e, g'^{h'[j]}e') \in R_2$. Since R_2 is complete, $(g^{h[\omega]}e, g'^{h'[\omega]}e') \in R_2$. Hence $g^{h[\omega]}, g'^{h'[\omega]} : R_1 \sqsubseteq R_2$, as desired. □

Lemma 4.7 *Suppose* $\Gamma \vdash \tau$ type, $\delta \in \mathit{TEnv}_\Gamma$, $\eta_1, \eta_2 \in \mathit{BEnv}_\delta$, *and* $\sigma_{\mathit{left}}, \sigma_{\mathit{right}}$ *are substitutions on terms such that* $\mathrm{Dom}(\sigma_{\mathit{left}}) = \mathrm{Dom}(\sigma_{\mathit{right}}) = \{p_\alpha \mid \alpha \in \mathrm{Dom}(\Gamma)\}$. *Additionally, suppose that, for all* $\alpha \in \mathrm{Dom}(\Gamma)$, $\sigma_{\mathit{left}}(p_\alpha), \sigma_{\mathit{right}}(p_\alpha) : \eta_1^+(\alpha) \sqsubseteq \eta_2^+(\alpha)$ *and* $\sigma_{\mathit{left}}(p_\alpha), \sigma_{\mathit{right}}(p_\alpha) : \eta_2^-(\alpha) \sqsubseteq \eta_1^-(\alpha)$. *Then* $\sigma_{\mathit{left}}(\delta_{\mathit{left}}(\pi_\tau)), \sigma_{\mathit{right}}(\delta_{\mathit{right}}(\pi_\tau)) : [\![\tau]\!]^\delta_{\eta_1} \sqsubseteq [\![\tau]\!]^\delta_{\eta_2}$.

Proof. For any τ, let $\bar{\pi}_\tau = \sigma_{\mathit{left}}(\delta_{\mathit{left}}(\pi_\tau))$ and $\bar{\pi}'_\tau = \sigma_{\mathit{right}}(\delta_{\mathit{right}}(\pi_\tau))$.

Case 1: Suppose τ is α. Then $\bar{\pi}_\alpha = \sigma_{\mathit{left}}(p_\alpha)$ and $\bar{\pi}'_\alpha = \sigma_{\mathit{right}}(p_\alpha)$, and by assumption $\sigma_{\mathit{left}}(p_\alpha), \sigma_{\mathit{right}}(p_\alpha) : \eta_1^+(\alpha) \sqsubseteq \eta_2^+(\alpha)$, but $[\![\alpha]\!]^\delta_{\eta_i} = \eta_i^+(\alpha)$. Therefore $\bar{\pi}_\alpha, \bar{\pi}'_\alpha : [\![\alpha]\!]^\delta_{\eta_1} \sqsubseteq [\![\alpha]\!]^\delta_{\eta_2}$.

Case 2: Suppose τ is 1. Note that $\bar{\pi}_1 = \bar{\pi}'_1 = \pi_1 = \lambda x{:}1.*$. Suppose $(e_1, e_2) \in [\![1]\!]^\delta_{\eta_1}$. Then $e_1\downarrow \Leftrightarrow e_2\downarrow$. Suppose $\bar{\pi}_1 e_1\downarrow$. Then $e_1\downarrow$, so $e_2\downarrow$ so $\bar{\pi}'_1 e_2\downarrow$. Similarly $\bar{\pi}'_1 e_2\downarrow$ implies $\bar{\pi}_1 e_1\downarrow$. Thus $(\bar{\pi}_1 e_1, \bar{\pi}'_1 e_2) \in [\![1]\!]^\delta_{\eta_2}$ and consequently $\bar{\pi}_1, \bar{\pi}'_1 : [\![1]\!]^\delta_{\eta_1} \sqsubseteq [\![1]\!]^\delta_{\eta_2}$.

Case 3: Suppose τ is $\tau_1 \to \tau_2$. Suppose $(e_1, e_2) \in [\![\tau_1 \to \tau_2]\!]^\delta_{\eta_1}$. When called, $\bar{\pi}_{\tau_1\to\tau_2}$ and $\bar{\pi}'_{\tau_1\to\tau_2}$ immediately return values, so $\bar{\pi}_{\tau_1\to\tau_2} e_1\downarrow \Leftrightarrow e_1\downarrow \Leftrightarrow e_2\downarrow \Leftrightarrow \bar{\pi}'_{\tau_1\to\tau_2} e_2\downarrow$. Suppose $(e'_1, e'_2) \in [\![\tau_1]\!]^\delta_{\eta_2^{\mathrm{op}}}$. By induction $\bar{\pi}_{\tau_1}, \bar{\pi}'_{\tau_1} : [\![\tau_1]\!]^\delta_{\eta_2^{\mathrm{op}}} \sqsubseteq [\![\tau_1]\!]^\delta_{\eta_1^{\mathrm{op}}}$, so $(\bar{\pi}_{\tau_1} e'_1, \bar{\pi}'_{\tau_1} e'_2) \in [\![\tau_1]\!]^\delta_{\eta_1^{\mathrm{op}}}$. Therefore $(e_1(\bar{\pi}_{\tau_1} e'_1), e_2(\bar{\pi}'_{\tau_1} e'_2)) \in [\![\tau_2]\!]^\delta_{\eta_1}$. By induction $\bar{\pi}_{\tau_2}, \bar{\pi}'_{\tau_2} : [\![\tau_2]\!]^\delta_{\eta_1} \sqsubseteq [\![\tau_2]\!]^\delta_{\eta_2}$, so $(\bar{\pi}_{\tau_2}(e_1(\bar{\pi}_{\tau_1} e'_1)), \bar{\pi}'_{\tau_2}(e_2(\bar{\pi}'_{\tau_1} e'_2))) \in [\![\tau_2]\!]^\delta_{\eta_2}$. Since $\bar{\pi}_{\tau_1\to\tau_2} e_1 e'_1 \approx \bar{\pi}_{\tau_2}(e_1(\bar{\pi}_{\tau_1} e'_1))$ and $\bar{\pi}'_{\tau_1\to\tau_2} e_2 e'_2 \approx \bar{\pi}'_{\tau_2}(e_2(\bar{\pi}'_{\tau_1} e'_2))$, we may conclude that $(\bar{\pi}_{\tau_1\to\tau_2} e_1 e'_1, \bar{\pi}'_{\tau_1\to\tau_2} e_2 e'_2) \in [\![\tau_2]\!]^\delta_{\eta_2}$, and thus $(\bar{\pi}_{\tau_1\to\tau_2} e_1, \bar{\pi}'_{\tau_1\to\tau_2} e_2) \in [\![\tau_1 \to \tau_2]\!]^\delta_{\eta_2}$. Therefore $\bar{\pi}_{\tau_1\to\tau_2}, \bar{\pi}'_{\tau_1\to\tau_2} : [\![\tau_1 \to \tau_2]\!]^\delta_{\eta_1} \sqsubseteq [\![\tau_1 \to \tau_2]\!]^\delta_{\eta_2}$.

Case 4: Suppose τ is $\forall\alpha.\tau'$ (choosing so that $\alpha \notin \mathrm{Dom}(\Gamma)$). Suppose $(e_1, e_2) \in [\![\forall\alpha.\tau']\!]^\delta_{\eta_1}$. When called, $\bar{\pi}_{\forall\alpha.\tau'}$ and $\bar{\pi}'_{\forall\alpha.\tau'}$ immediately return values, so $\bar{\pi}_{\forall\alpha.\tau'} e_1\downarrow \Leftrightarrow e_1\downarrow \Leftrightarrow e_2\downarrow \Leftrightarrow \bar{\pi}'_{\forall\alpha.\tau'} e_2\downarrow$. Suppose $\tau_1, \tau_2 \in \mathit{Type}$ and $R \in \mathit{ARel}_{\tau_1,\tau_2}$. Let $\delta' = \delta[\alpha \mapsto (\tau_1, \tau_2)]$, $\eta'_i = \eta_i[\alpha \mapsto (R, R)]$, $\sigma'_{\mathit{left}} = \sigma[p_\alpha \mapsto \mathit{id}_{\tau_1}]$, and $\sigma'_{\mathit{right}} = \sigma[p_\alpha \mapsto \mathit{id}_{\tau_2}]$. Then $(e_1[\tau_1], e_2[\tau_2]) \in [\![\tau']\!]^{\delta'}_{\eta'_1}$. Certainly $\mathit{id}_{\tau_1}, \mathit{id}_{\tau_2} : R \sqsubseteq R$, so by induction $\sigma'_{\mathit{left}}(\delta'_{\mathit{left}}(\pi_{\tau'})), \sigma'_{\mathit{right}}(\delta'_{\mathit{right}}(\pi_{\tau'})) : [\![\tau']\!]^{\delta'}_{\eta'_1} \sqsubseteq [\![\tau']\!]^{\delta'}_{\eta'_2}$. Then $(\sigma'_{\mathit{left}}(\delta'_{\mathit{left}}(\pi_{\tau'}))(e_1[\tau_1]), \sigma'_{\mathit{right}}(\delta'_{\mathit{right}}(\pi_{\tau'}))(e_2[\tau_2])) \in [\![\tau']\!]^{\delta'}_{\eta'_2}$. Rearranging, $(\bar{\pi}_{\forall\alpha.\tau'}\, e_1[\tau_1], \bar{\pi}'_{\forall\alpha.\tau'}\, e_2[\tau_2]) \in [\![\tau']\!]^{\delta'}_{\eta'_2}$, and thus $(\bar{\pi}_{\forall\alpha.\tau'}\, e_1, \bar{\pi}'_{\forall\alpha.\tau'}\, e_2) \in [\![\forall\alpha.\tau']\!]^\delta_{\eta_2}$. Therefore $\bar{\pi}_{\forall\alpha.\tau'}, \bar{\pi}'_{\forall\alpha.\tau'} : [\![\forall\alpha.\tau']\!]^\delta_{\eta_1} \sqsubseteq [\![\forall\alpha.\tau']\!]^\delta_{\eta_2}$.

Case 5: Suppose τ is $\mu\alpha.\tau'$ (choosing so that $\alpha \notin \mathrm{Dom}(\Gamma)$). Let Ψ_η be defined as in Definition 4.3, let $\Psi^{\S}_i(R^-, R^+) = (\Psi_{\eta_i^{\mathrm{op}}}(R^+, R^-), \Psi_{\eta_i}(R^-, R^+))$, and let $(\Delta_i^-, \Delta_i^+) = \mathit{lfp}\, \Psi^{\S}_i$. Note that $[\![\mu\alpha.\tau']\!]^\delta_{\eta_i} = \Delta_i^+$. We show by fixed point induction that $\bar{\pi}_{\mu\alpha.\tau'}, \bar{\pi}'_{\mu\alpha.\tau'} : \Delta_1^+ \sqsubseteq \Delta_2^+$ and $\bar{\pi}_{\mu\alpha.\tau'}, \bar{\pi}'_{\mu\alpha.\tau'} : \Delta_2^- \sqsubseteq \Delta_1^-$. (Note that the relation $\{(f, g) \mid f, g : \Delta_1^+ \sqsubseteq \Delta_2^+ \text{ and } f, g : \Delta_2^- \sqsubseteq \Delta_1^-\}$ is admissible by Lemma 4.6 since the set of admissible relations is closed under intersection.) Certainly $\lambda x.\bot, \lambda x.\bot : \Delta_1^+ \sqsubseteq \Delta_2^+$ and $\lambda x.\bot, \lambda x.\bot : \Delta_2^- \sqsubseteq \Delta_1^-$, since Δ_2^+ and Δ_1^- are pointed. Suppose, for fixed point induction, that $f, g : \Delta_1^+ \sqsubseteq \Delta_2^+$ and $f, g : \Delta_2^- \sqsubseteq \Delta_1^-$. Let $f' = \lambda x{:}\delta_{\mathit{left}}(\tau).\mathtt{in}_{\delta_{\mathit{left}}(\tau)}(\bar{\pi}_{\tau'}[\delta_{\mathit{left}}(\tau), f/\alpha, p_\alpha](\mathtt{out}\, x))$ and $g' = \lambda x{:}\delta_{\mathit{right}}(\tau).\mathtt{in}_{\delta_{\mathit{right}}(\tau)}(\bar{\pi}'_{\tau'}[\delta_{\mathit{right}}(\tau), g/\alpha, p_\alpha](\mathtt{out}\, x))$. We wish to show that $f', g' : \Delta_1^+ \sqsubseteq \Delta_2^+$ and $f', g' : \Delta_2^- \sqsubseteq \Delta_1^-$.

Suppose $(e_1, e_2) \in \Delta_1^+$. Let $\delta' = \delta[\alpha \mapsto \delta(\mu\alpha.\tau')]$, $\eta'_i = \eta_i[\alpha \mapsto (\Delta_i^-, \Delta_i^+)]$, $\sigma'_{\mathit{left}} = \sigma_{\mathit{left}}[p_\alpha \mapsto f]$ and $\sigma'_{\mathit{right}} = \sigma_{\mathit{right}}[p_\alpha \mapsto g]$. Since $\Delta_1^+ = \Psi_{\eta_1}(\Delta_1^-, \Delta_1^+)$, it follows that $(\mathtt{out}\, e_1, \mathtt{out}\, e_2) \in [\![\tau']\!]^{\delta'}_{\eta'_1}$. Now let $\bar{\pi} = \sigma'_{\mathit{left}}(\delta'_{\mathit{left}}(\pi_{\tau'}))$ and $\bar{\pi}' = \sigma'_{\mathit{right}}(\delta'_{\mathit{right}}(\pi_{\tau'}))$. By induction, $\bar{\pi}, \bar{\pi}' : [\![\tau']\!]^{\delta'}_{\eta'_1} \sqsubseteq [\![\tau']\!]^{\delta'}_{\eta'_2}$. It follows that $(\bar{\pi}(\mathtt{out}\, e_1), \bar{\pi}'(\mathtt{out}\, e_2)) \in [\![\tau']\!]^{\delta'}_{\eta'_2}$, so $(\mathtt{out}(\mathtt{in}_{\delta_{\mathit{left}}(\tau)}(\bar{\pi}(\mathtt{out}\, e_1))), \mathtt{out}(\mathtt{in}_{\delta_{\mathit{right}}(\tau)}(\bar{\pi}'(\mathtt{out}\, e_2)))) \in [\![\tau']\!]^{\delta'}_{\eta'_2}$. Thus $(\mathtt{in}_{\delta_{\mathit{left}}(\tau)}(\bar{\pi}(\mathtt{out}\, e_1)), \mathtt{in}_{\delta_{\mathit{right}}(\tau)}(\bar{\pi}'(\mathtt{out}\, e_2))) \in \Delta_2^+$. Rearranging, $(f' e_1, g' e_2) \in \Delta_2^+$. Thus, $f', g' : \Delta_1^+ \sqsubseteq \Delta_2^+$.

Symmetrically, suppose $(e_1, e_2) \in \Delta_2^-$. Since $\Delta_2^- = \Psi_{\eta_2^{\mathrm{op}}}(\Delta_2^+, \Delta_2^-)$, it follows that $(\mathtt{out}\, e_1, \mathtt{out}\, e_2) \in [\![\tau']\!]^{\delta'}_{\eta_2'^{\mathrm{op}}}$. Again by induction, $\bar{\pi}, \bar{\pi}' : [\![\tau']\!]^{\delta'}_{\eta_2'^{\mathrm{op}}} \sqsubseteq [\![\tau']\!]^{\delta'}_{\eta_1'^{\mathrm{op}}}$. Then $(\bar{\pi}(\mathtt{out}\, e_1), \bar{\pi}'(\mathtt{out}\, e_2)) \in [\![\tau']\!]^{\delta'}_{\eta_1'^{\mathrm{op}}}$. Thus $(\mathtt{in}_{\delta_{left}(\tau)}(\bar{\pi}(\mathtt{out}\, e_1)), \mathtt{in}_{\delta_{right}(\tau)}(\bar{\pi}'(\mathtt{out}\, e_2))) \in \Delta_1^-$. Rearranging, $(f'e_1, g'e_2) \in \Delta_1^-$. Thus $f', g' : \Delta_2^- \sqsubseteq \Delta_1^-$.

By fixed point induction we may conclude that $\bar{\pi}_{\mu\alpha.\tau'}, \bar{\pi}'_{\mu\alpha.\tau'} : \Delta_1^+ \sqsubseteq \Delta_2^+$. (Also that $\bar{\pi}_{\mu\alpha.\tau'}, \bar{\pi}'_{\mu\alpha.\tau'} : \Delta_2^- \sqsubseteq \Delta_1^-$, but we do not need this fact.) That is, $\bar{\pi}_{\mu\alpha.\tau'}, \bar{\pi}'_{\mu\alpha.\tau'} : [\![\mu\alpha.\tau']\!]^{\delta}_{\eta_1} \sqsubseteq [\![\mu\alpha.\tau']\!]^{\delta}_{\eta_2}$, as desired. □

As a corollary we can obtain the main lemma needed to prove the unrolling theorem. It states that the syntactic projection function π_τ takes $[\![\tau]\!]_{\eta^{\mathrm{op}}}$ to $[\![\tau]\!]_\eta$. Combined with Syntactic Minimal Invariance, this gives that $[\![\tau]\!]_{\eta^{\mathrm{op}}} \sqsubseteq [\![\tau]\!]_\eta$. For convenience, the lemma is stated in a slightly stronger fashion than needed. It allows η^- and η^+ to differ on any argument, but in our use of the lemma, they differ on at most a single argument. (In fact, when all is said and done they can be shown to agree on that argument as well.)

Corollary 4.8 (Main Lemma) *Suppose* $\Gamma \vdash \tau$ type, $\delta \in TEnv_\Gamma$, $\eta \in BEnv_\delta$, *and* $\sigma_{left}, \sigma_{right}$ *are substitutions on terms such that* $\mathrm{Dom}(\sigma_{left}) = \mathrm{Dom}(\sigma_{right}) = \{p_\alpha \mid \alpha \in \mathrm{Dom}(\Gamma)\}$. *Suppose further that, for all* $\alpha \in \mathrm{Dom}(\Gamma)$, $\sigma_{left}(p_\alpha), \sigma_{right}(p_\alpha) : \eta^-(\alpha) \sqsubseteq \eta^+(\alpha)$. *Then* $\sigma_{left}(\delta_{left}(\pi_\tau)), \sigma_{right}(\delta_{right}(\pi_\tau)) : [\![\tau]\!]^{\delta}_{\eta^{\mathrm{op}}} \sqsubseteq [\![\tau]\!]^{\delta}_{\eta}$.

Proof. Immediate from Lemma 4.7, using $\eta_1 = \eta^{\mathrm{op}}$ and $\eta_2 = \eta$. □

Lemma 4.9 (Compositionality) *Suppose* $\Gamma, \alpha \vdash \tau$ type, $\Gamma \vdash \tau'$ type, $\delta \in TEnv_\Gamma$, *and* $\eta \in BEnv_\delta$. *Then:*

$$[\![\tau[\tau'/\alpha]]\!]^{\delta}_{\eta} = [\![\tau]\!]^{\delta[\alpha \mapsto \delta(\tau')]}_{\eta[\alpha \mapsto ([\![\tau']\!]^{\delta}_{\eta^{\mathrm{op}}}, [\![\tau']\!]^{\delta}_{\eta})]}$$

Proof. By induction on τ. □

Next is the key result of the construction:

Theorem 4.10 (Unrolling) *Suppose* $\Gamma, \alpha \vdash \tau$ type, $\delta \in TEnv_\Gamma$, *and* $\eta \in BEnv_\delta$, *and suppose that* $\eta = \eta^{\mathrm{op}}$. *Then* $[\![\mu\alpha.\tau]\!]^{\delta}_{\eta} = \{(e_1, e_2) \in ECV_{\delta(\mu\alpha.\tau)} \mid (\mathtt{out}\, e_1, \mathtt{out}\, e_2) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\eta}\}$.

Proof. Let Δ^-, Δ^+, $\Psi^{\S}$, and Ψ_η be defined as in Definition 4.3. Note that $(\Delta^-, \Delta^+) = \Psi^{\S}(\Delta^-, \Delta^+) = (\Psi_{\eta^{\mathrm{op}}}(\Delta^+, \Delta^-), \Psi_\eta(\Delta^-, \Delta^+))$. We claim that $\Delta^- = \Delta^+$. It then follows that:

$$\begin{aligned}
[\![\mu\alpha.\tau]\!]^{\delta}_{\eta} &= \Delta^+ \\
&= \Psi_\eta(\Delta^+, \Delta^+) \\
&= \{(e_1, e_2) \in ECV_{\delta(\mu\alpha.\tau)} \mid (\mathtt{out}\, e_1, \mathtt{out}\, e_2) \in [\![\tau]\!]^{\delta[\alpha \mapsto \delta(\mu\alpha.\tau)]}_{\eta[\alpha \mapsto (\Delta^+, \Delta^+)]}\} \\
&= \{(e_1, e_2) \in ECV_{\delta(\mu\alpha.\tau)} \mid (\mathtt{out}\, e_1, \mathtt{out}\, e_2) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\eta}\}
\end{aligned}$$

We freely use the fact that $\eta = \eta^{\mathrm{op}}$. The last line follows by Lemma 4.9. It remains to prove the claim. We will first show that $\Delta^+ \sqsubseteq \Delta^-$ (this will be easy) and then show that $\Delta^- \sqsubseteq \Delta^+$ (this is the main technical point).

For the first inclusion, observe that (Δ^+, Δ^-) is a fixed point of $\Psi^{\S}$:

$$\begin{aligned}\Psi^{\S}(\Delta^+, \Delta^-) &= (\Psi_{\eta^{\mathrm{op}}}(\Delta^-, \Delta^+), \Psi_{\eta}(\Delta^+, \Delta^-)) \\ &= (\Psi_{\eta}(\Delta^-, \Delta^+), \Psi_{\eta^{\mathrm{op}}}(\Delta^+, \Delta^-)) \\ &= (\Delta^+, \Delta^-)\end{aligned}$$

Since (Δ^-, Δ^+) is the *least* fixed point of $\Psi^{\S}$, it follows that $(\Delta^-, \Delta^+) \sqsubseteq (\Delta^+, \Delta^-)$. By the ordering on birelations this means that $\Delta^+ \sqsubseteq \Delta^-$.

For the second inclusion, let $\sigma_{left}(p_\beta) = id_{\delta_{left}(\beta)}$ and $\sigma_{right}(p_\beta) = id_{\delta_{right}(\beta)}$ for all $\beta \in \mathrm{Dom}(\Gamma)$. We show by fixed point induction that $\sigma_{left}(\delta_{left}(\pi_{\mu\alpha.\tau})), \sigma_{right}(\delta_{right}(\pi_{\mu\alpha.\tau})) : \Delta^- \sqsubseteq \Delta^+$. Certainly $\lambda x.\bot : \Delta^- \sqsubseteq \Delta^+$, since Δ^+ is pointed. Suppose, for fixed point induction, that $f, g : \Delta^- \sqsubseteq \Delta^+$. Let $\delta' = \delta[\alpha \mapsto \delta(\mu\alpha.\tau)]$, let $\eta' = \eta[\alpha \mapsto (\Delta^-, \Delta^+)]$, and let $\sigma'_{left} = \sigma_{left}[p_\alpha \mapsto f]$ and $\sigma'_{right} = \sigma_{right}[p_\alpha \mapsto g]$. Suppose $(e, e') \in \Delta^- = \Psi_{\eta^{\mathrm{op}}}(\Delta^+, \Delta^-)$. Then $(\mathtt{out}\, e, \mathtt{out}\, e') \in [\![\tau]\!]^{\delta'}_{\eta'^{\mathrm{op}}}$. Since $\eta = \eta^{\mathrm{op}}$, $id_{\delta_{left}(\beta)}, id_{\delta_{right}(\beta)} : \eta^-(\beta) \sqsubseteq \eta^+(\beta)$, for all $\beta \in \mathrm{Dom}(\Gamma)$. Therefore, by Corollary 4.8, $(\sigma'(\delta'(\pi_\tau))(\mathtt{out}\, e), \sigma'(\delta'(\pi_\tau))(\mathtt{out}\, e')) \in [\![\tau]\!]^{\delta'}_{\eta'}$. It is then easy to show that $(f'e, g'e') \in \Delta^+$ where $f' = \lambda x{:}\delta_{left}(\mu\alpha.\tau).\mathtt{in}_{\delta_{left}(\mu\alpha.\tau)}(\sigma_{left}(\delta_{left}(\pi_\tau[\mu\alpha.\tau, f/\alpha, p_\alpha]))(\mathtt{out}\, x))$ and $g' = \lambda x{:}\delta_{right}(\mu\alpha.\tau).\mathtt{in}_{\delta_{right}(\mu\alpha.\tau)}(\sigma_{right}(\delta_{right}(\pi_\tau[\mu\alpha.\tau, g/\alpha, p_\alpha]))(\mathtt{out}\, x))$. Therefore $f', g' : \Delta^- \sqsubseteq \Delta^+$, and by fixed point induction $\sigma_{left}(\delta_{left}(\pi_{\mu\alpha.\tau})), \sigma_{right}(\delta_{right}(\pi_{\mu\alpha.\tau})) : \Delta^- \sqsubseteq \Delta^+$. Syntactic Minimal Invariance dictates that $\sigma_{left}(\delta_{left}(\pi_{\mu\alpha.\tau})) \approx id_{\delta_{left}(\mu\alpha.\tau)}$ and $\sigma_{right}(\delta_{right}(\pi_{\mu\alpha.\tau})) \approx id_{\delta_{right}(\mu\alpha.\tau)}$. Since Δ^+ must respect applicative equivalence, we conclude that $\Delta^- \sqsubseteq \Delta^+$. □

Notation Suppose χ is a relation environment. We will view χ as a birelation by mapping each α to the pair $(\chi(\alpha), \chi(\alpha))$. Thus we may speak of $[\![\tau]\!]^{\delta}_{\chi}$, the interpretation of a type relative to a relation environment.

4.2 The Fundamental Theorem

Definition 4.11 *We write $\vdash \delta, \chi, \sigma_{left}, \sigma_{right} : \Gamma$ to mean that δ is a type environment over Γ, that χ is a relation environment respecting δ, and that σ_{left} and σ_{right} are substitutions for the term variables bound by Γ such that for every $x{:}\tau \in \Gamma$, $(\sigma_{left}(x), \sigma_{right}(x)) \in [\![\tau]\!]^{\delta}_{\chi}$.*

Definition 4.12 Suppose $\Gamma \vdash e, e' : \tau$. Then e and e' are logically equivalent in Γ and at τ (written $\Gamma \vdash e \Leftrightarrow e' : \tau$) if for any $\vdash \delta, \chi, \sigma_{left}, \sigma_{right} : \Gamma$, $(\sigma_{left}(\delta_{left}(e)), \sigma_{right}(\delta_{right}(e'))) \in [\![\tau]\!]^{\delta}_{\chi}$.

Theorem 4.13 (Fundamental Theorem of Logical Relations)
Suppose $\Gamma \vdash e : \tau$. Then $\Gamma \vdash e \Leftrightarrow e : \tau$.

Proof. By induction on the derivation of $\Gamma \vdash e : \tau$. Let $\vdash \delta, \chi, \sigma_{left}, \sigma_{right} : \Gamma$ be arbitrary.

Case 1: Suppose the last rule applied is:

$$\frac{\vdash \Gamma \text{ context} \quad \Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

The result is immediate from the assumption.

Case 2: Suppose the last rule applied is:

$$\frac{\vdash \Gamma \text{ context}}{\Gamma \vdash * : 1}$$

Then $\sigma_{left}(*) = * = \sigma_{right}(*)$, and $(*, *) \in [\![1]\!]^{\delta}_{\chi}$.

Case 3: Suppose the last rule applied is:

$$\frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.e : \tau \to \tau'}$$

Let $\sigma_{left}(\delta_{left}(\lambda x{:}\tau.e))$ be $\lambda x{:}\tau_1.e_1$, and let $\sigma_{right}(\delta_{right}(\lambda x{:}\tau.e))$ be $\lambda x{:}\tau_2.e_2$. Note that both terms halt. Now suppose $(e'_1, e'_2) \in [\![\tau]\!]^{\delta}_{\chi}$. We wish to show that $((\lambda x{:}\tau_1.e_1)e'_1, (\lambda x{:}\tau_2.e_2)e'_2) \in [\![\tau']\!]^{\delta}_{\chi}$.

Suppose e'_1 diverges. Since $[\![\tau]\!]^{\delta}_{\chi}$ is strict, e'_2 diverges as well. Thus $(\lambda x{:}\tau_1.e_1)e'_1 \approx \bot$ and $(\lambda x{:}\tau_2.e_2)e'_2 \approx \bot$. Since $[\![\tau']\!]^{\delta}_{\chi}$ is admissible (and hence pointed), $(\bot, \bot) \in [\![\tau']\!]^{\delta}_{\chi}$. The result follows since the logical interpretations are closed under applicative equivalence.

Alternatively, suppose e'_1 halts. Then e'_2 halts as well. Thus $(\lambda x{:}\tau_1.e_1)e'_1 \approx e_1[e'_1/x]$ and $(\lambda x{:}\tau_2.e_2)e'_2 \approx e_2[e'_2/x]$. Let σ'_{left} be $\sigma_{left}[x \mapsto e'_1]$ and let σ'_{right} be $\sigma_{right}[x \mapsto e'_2]$. Then $\vdash \delta, \chi, \sigma'_{left}, \sigma'_{right} : (\Gamma, x{:}\tau)$. By induction, $(\sigma'_{left}(\delta_{left}(e)), \sigma'_{right}(\delta_{right}(e))) \in [\![\tau']\!]^{\delta}_{\chi}$. That is, $(e_1[e'_1/x], e_2[e'_2/x]) \in [\![\tau']\!]^{\delta}_{\chi}$. The result follows by closure under applicative equivalence.

Case 4: Suppose the last rule applied is:

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

By induction, $(\sigma_{left}(\delta_{left}(e_1)), \sigma_{right}(\delta_{right}(e_1))) \in [\![\tau_1 \to \tau_2]\!]^{\delta}_{\chi}$ and $(\sigma_{left}(\delta_{left}(e_2)), \sigma_{right}(\delta_{right}(e_2))) \in [\![\tau_1]\!]^{\delta}_{\chi}$. Therefore $(\sigma_{left}(\delta_{left}(e_1))\sigma_{left}(\delta_{left}(e_2)), \sigma_{right}(\delta_{right}(e_1))\sigma_{right}(\delta_{right}(e_2))) \in [\![\tau_2]\!]^{\delta}_{\chi}$. That is, $(\sigma_{left}(\delta_{left}(e_1 e_2)), \sigma_{right}(\delta_{right}(e_1 e_2))) \in [\![\tau_2]\!]^{\delta}_{\chi}$.

Case 5: Suppose the last rule applied is:

$$\frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}$$

Let $\sigma_{left}(\delta_{left}(\Lambda\alpha.e))$ be $\Lambda\alpha.e_1$, and let $\sigma_{right}(\delta_{right}(\Lambda\alpha.e))$ be $\Lambda\alpha.e_2$. Note that both terms halt. Now suppose $\tau_1, \tau_2 \in \mathit{Type}$, and suppose $R \in \mathit{ARel}_{\tau_1,\tau_2}$. Let δ' be $\delta[\alpha \mapsto (\tau_1, \tau_2)]$ and let χ' be $\chi[\alpha \mapsto R]$. Then we wish to show that $((\Lambda\alpha.e_1)[\tau_1], (\Lambda\alpha.e_2)[\tau_2]) \in [\![\tau]\!]^{\delta'}_{\chi'}$.

Observe that $(\Lambda\alpha.e_1)[\tau_1] \approx e_1[\tau_1/\alpha]$ and $(\Lambda\alpha.e_2)[\tau_2] \approx e_2[\tau_2/\alpha]$. Also, δ' is a type environment over (Γ, α), and χ' respects δ', so $\vdash \delta', \chi', \sigma_{left}, \sigma_{right} : (\Gamma, \alpha)$. By induction, $(\sigma_{left}(\delta'_{left}(e)), \sigma_{right}(\delta'_{right}(e))) \in [\![\tau]\!]^{\delta'}_{\chi'}$. That is, $(e_1[\tau_1/\alpha], e_2[\tau_2/\alpha]) \in [\![\tau]\!]^{\delta'}_{\chi'}$. The result then follows by closure under applicative equivalence.

Case 6: Suppose the last rule applied is:

$$\frac{\Gamma \vdash e : \forall\alpha.\tau' \quad \Gamma \vdash \tau \text{ type}}{\Gamma \vdash e[\tau] : \tau'[\tau/\alpha]}$$

By induction, $(\sigma_{left}(\delta_{left}(e)), \sigma_{right}(\delta_{right}(e))) \in [\![\forall\alpha.\tau']\!]^{\delta}_{\chi}$. Therefore:

$$(\sigma_{left}(\delta_{left}(e))[\delta_{left}(\tau)], \sigma_{right}(\delta_{right}(e))[\delta_{right}(\tau)]) \in [\![\tau']\!]^{\delta[\alpha\mapsto\delta(\tau)]}_{\chi[\alpha\mapsto[\![\tau]\!]^{\delta}_{\chi}]}$$

That is, using substitution and rearranging:

$$(\sigma_{left}(\delta_{left}(e[\tau])), \sigma_{right}(\delta_{right}(e[\tau]))) \in [\![\tau'[\tau/\alpha]]\!]^{\delta}_{\chi}$$

Case 7: Suppose the last rule applied is:

$$\frac{\Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \mathtt{in}_{\mu\alpha.\tau} e : \mu\alpha.\tau}$$

Let $\sigma_{left}(\delta_{left}(e))$ be e_1, and let $\sigma_{right}(\delta_{right}(e))$ be e_2. Then, using the Unrolling Lemma, it suffices to show that $(\mathtt{out}(\mathtt{in}\, e_1), \mathtt{out}(\mathtt{in}\, e_2)) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\chi}$. Observe that $\mathtt{out}(\mathtt{in}\, e_1) \approx e_1$ and $\mathtt{out}(\mathtt{in}\, e_2) \approx e_2$. By induction, $(e_1, e_2) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\chi}$. The result then follows by closure under applicative equivalence.

Case 8: Suppose the last rule applied is:

$$\frac{\Gamma \vdash e : \mu\alpha.\tau}{\Gamma \vdash \mathtt{out}\, e : \tau[\mu\alpha.\tau/\alpha]}$$

By induction, $(\sigma_{left}(\delta_{left}(e)), \sigma_{right}(\delta_{right}(e))) \in [\![\mu\alpha.\tau]\!]^{\delta}_{\chi}$. Using the Unrolling Lemma, $(\mathtt{out}\, \sigma_{left}(\delta_{left}(e)), \mathtt{out}\, \sigma_{right}(\delta_{right}(e))) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\chi}$. The result then follows by rearrangement. □

Theorem 4.14 (Applicative Equivalence implies Logical Equivalence) *If* $\Gamma \vdash e \approx e' : \tau$ *then* $\Gamma \vdash e \Leftrightarrow e' : \tau$.

Proof. Suppose $\Gamma \vdash e \approx e' : \tau$ and let $\vdash \delta, \chi, \sigma_{left}, \sigma_{right} : \Gamma$ be arbitrary. By the Fundamental Theorem, $(\sigma_{left}(\delta_{left}(e)), \sigma_{right}(\delta_{right}(e))) : [\![\tau]\!]^{\delta}_{\chi}$. By the definition of applicative equivalence on open terms, $\sigma_{right}(\delta_{right}(e)) \approx \sigma_{right}(\delta_{right}(e'))$. Since

$$\boxed{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau)}$$

$$\frac{}{[\,] : (\Gamma \vdash \tau) \Rightarrow (\Gamma \vdash \tau)} \qquad \frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma, x{:}\tau_1 \vdash \tau_2)}{\lambda x{:}\tau_1.C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau_1 \to \tau_2)}$$

$$\frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau_1 \to \tau_2) \quad \Gamma \vdash e : \tau_1}{C\,e : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau_2)}$$

$$\frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau_1) \quad \Gamma \vdash e : \tau_1 \to \tau_2}{e\,C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau_2)}$$

$$\frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma, \alpha \vdash \tau)}{\Lambda\alpha.C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \forall\alpha.\tau)} \qquad \frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \forall\alpha.\tau_1) \quad \Gamma \vdash \tau_2 \text{ type}}{C[\tau_2] : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau_1[\tau_2/\alpha])}$$

$$\frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau[\mu\alpha.\tau/\alpha])}{\mathtt{in}_{\mu\alpha.\tau}C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \mu\alpha.\tau)} \qquad \frac{C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \mu\alpha.\tau)}{\mathtt{out}\,C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau[\mu\alpha.\tau/\alpha])}$$

Fig. 7. Context Typing

$[\![\tau]\!]^{\delta}_{\chi}$ respects applicative equivalence, $(\sigma_{left}(\delta_{left}(e)), \sigma_{right}(\delta_{right}(e'))) : [\![\tau]\!]^{\delta}_{\chi}$. Hence $\Gamma \vdash e \Leftrightarrow e' : \tau$. □

5 Contextual Equivalence

Two open expressions of the same type are *contextually equivalent* [18] if and only if they are indistinguishable by closing contexts of unit type in that the result closed programs either both halt or both diverge. Contextual equivalence, logical equivalence, and applicative equivalence coincide for our language.

To begin with we define the syntax of contexts by the following grammar:

$$\textit{Contexts} \quad C ::= [\,] \mid \lambda x{:}\tau.C \mid Ce \mid eC \mid \Lambda\alpha.C \mid C[\tau] \mid \mathtt{in}_{\mu\alpha.\tau}C \mid \mathtt{out}\,C$$

Instantiation of contexts (written $C[e]$) and composition of contexts (written $C \circ C'$) are defined in the usual manner. Typing rules for contexts are given in Figure 7.

Proposition 5.1

- *If* $C : (\Gamma' \vdash \tau') \Rightarrow (\Gamma \vdash \tau)$ *and* $\Gamma' \vdash e : \tau'$ *then* $\Gamma \vdash C[e] : \tau$.
- *If* $C : (\Gamma_2 \vdash \tau_2) \Rightarrow (\Gamma_1 \vdash \tau_1)$ *and* $C' : (\Gamma_3 \vdash \tau_3) \Rightarrow (\Gamma_2 \vdash \tau_2)$ *then* $C \circ C' : (\Gamma_3 \vdash \tau_3) \Rightarrow (\Gamma_1 \vdash \tau_1)$.

Two terms are contextually equivalent if no type-appropriate context can distinguish them:

Definition 5.2 (Contextual Equivalence) Suppose $\Gamma \vdash e, e' : \tau$. Then e and e' are contextually equivalent in Γ and at τ (written $\Gamma \vdash e \cong e' : \tau$) if $C[e]$ halts if and only if $C[e']$ halts, for every $C : (\Gamma \vdash \tau) \Rightarrow (\epsilon \vdash 1)$.

Proposition 5.3 *Contextual equivalence is reflexive (over appropriately typed terms), symmetric, and transitive.*

A type-indexed equivalence is *consistent* iff it relates two closed expressions of unit type only if they either both diverge or both converge.

Proposition 5.4 *Contextual equivalence is the coarsest consistent congruence on terms.*

5.1 Contextual Equivalence implies Applicative Equivalence

The conditions defining applicative equivalence amount to consideration of particular contexts. It is therefore no finer than contextual equivalence, and can be no coarser, since it is a consistent congruence.

Lemma 5.5 *If* $\vdash e \cong e' : \tau$ *then* $\vdash e \approx e' : \tau$.

Proof. By Lemma 2.9, it suffices to check four conditions:

- Suppose $\vdash e \cong e' : \tau$ and e halts. Observe that $(\lambda x{:}\tau.*)[\,] : (\epsilon \vdash \tau) \Rightarrow (\epsilon \vdash 1)$, so $(\lambda x{:}\tau.*)e$ halts if and only if $(\lambda{:}\tau.*)e'$ halts. Therefore e' halts.
- Suppose $\vdash e \cong e' : \tau_1 \to \tau_2$ and $\vdash v : \tau_1$. Let $C : (\epsilon \vdash \tau_2) \Rightarrow (\epsilon \vdash 1)$ be arbitrary. Then $C \circ ([\,]v) : (\epsilon \vdash \tau_1 \to \tau_2) \Rightarrow (\epsilon \vdash 1)$. By contextual equivalence of e and e', it follows that $C[ev]$ halts if and only if $C[e'v]$ halts.
- The remaining cases are similar

Thus $\vdash e \cong e' : \tau$ implies $\vdash e \preceq e' : \tau$. The result follows by the symmetry of contextual equivalence. □

Lemma 5.6 *Suppose σ is a substitution, and let C_Γ be defined as follows:*

$$\begin{aligned} C_\epsilon &\stackrel{\text{def}}{=} [\,] \\ C_{\Gamma,\alpha} &\stackrel{\text{def}}{=} C_\Gamma[\Lambda\alpha.[\,]][\sigma(\alpha)] \\ C_{\Gamma,x:\tau} &\stackrel{\text{def}}{=} C_\Gamma[\lambda x{:}\tau.[\,]]\sigma(x) \end{aligned}$$

If $\vdash \sigma : \Gamma$ *then*

(i) *for any* $\Gamma \vdash \tau$ type, $C_\Gamma : (\Gamma \vdash \tau) \Rightarrow (\epsilon \vdash \sigma(\tau))$, *and*

(ii) *for any* $\Gamma \vdash e : \tau$, $C_\Gamma[e] \approx \sigma(e)$.

Proof. By induction on Γ. □

Theorem 5.7 (Contextual Equivalence implies Applicative Equivalence) *If* $\Gamma \vdash e \cong e' : \tau$ *then* $\Gamma \vdash e \approx e' : \tau$.

Proof. Let $\vdash \sigma : \Gamma$ be arbitrary. By definition, we wish to show that $\vdash \sigma(e) \approx \sigma(e') : \sigma(\tau)$. Let C_Γ be defined as in Lemma 5.6. Since $C_\Gamma[e] \approx \sigma(e)$ and $C_\Gamma[e'] \approx \sigma(e')$, it suffices to show that $\vdash C_\Gamma[e] \approx C_\Gamma[e'] : \sigma(\tau)$. Finally, by Lemma 5.5, it suffices to show that $\vdash C_\Gamma[e] \cong C_\Gamma[e'] : \sigma(\tau)$.

Thus, let $C : (\epsilon \vdash \sigma(\tau)) \Rightarrow (\epsilon \vdash 1)$ be arbitrary. By Lemma 5.6, $C_\Gamma : (\Gamma \vdash \tau) \Rightarrow (\epsilon \vdash \sigma(\tau))$. Hence $C \circ C_\Gamma : (\Gamma \vdash \tau) \Rightarrow (\epsilon \vdash 1)$. Since e and e' are contextually equivalent, we may conclude that $C[C_\Gamma[e]]$ halts if and only if $C[C_\Gamma[e']]$ halts. Therefore $\vdash C_\Gamma[e] \cong C_\Gamma[e'] : \sigma(\tau)$. □

5.2 Logical Equivalence implies Contextual Equivalence

Logical equivalence is a congruence, and is consistent by definition. The reader may note that the proof of this is very similar to that of the Fundamental Theorem.

Lemma 5.8 *If* $\hat{\Gamma} \vdash e \Leftrightarrow e' : \hat{\tau}$ *and* $C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau)$ *then* $\Gamma \vdash C[e] \Leftrightarrow C[e'] : \tau$.

Proof. By induction on the derivation of $C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau)$. Suppose $\hat{\Gamma} \vdash e \Leftrightarrow e' : \hat{\tau}$ and let $\vdash \delta, \chi, \sigma_{left}, \sigma_{right} : \Gamma$ be arbitrary.

Case 1: Suppose the last rule applied is:

$$\frac{}{[\,] : (\Gamma \vdash \tau) \Rightarrow (\Gamma \vdash \tau)}$$

Then the result is immediate.

Case 2: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma, x{:}\tau \vdash \tau')}{\lambda x{:}\tau.C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau \to \tau')}$$

Let $\sigma_{left}(\delta_{left}(\lambda x{:}\tau.C[e]))$ be $\lambda x{:}\tau_1.e_1$, and let $\sigma_{right}(\delta_{right}(\lambda x{:}\tau.C[e']))$ be $\lambda x{:}\tau_2.e_2$. Note that both terms halt. Now suppose $(e'_1, e'_2) \in [\![\tau]\!]^\delta_\chi$. We wish to show that $((\lambda x{:}\tau_1.e_1)e'_1, (\lambda x{:}\tau_2.e_2)e'_2) \in [\![\tau']\!]^\delta_\chi$.

Suppose e'_1 diverges. Since $[\![\tau]\!]^\delta_\chi$ is strict, e'_2 diverges as well. Thus $(\lambda x{:}\tau_1.e_1)e'_1 \approx \bot$ and $(\lambda x{:}\tau_2.e_2)e'_2 \approx \bot$. Since $[\![\tau']\!]^\delta_\chi$ is admissible (and hence pointed), $(\bot, \bot) \in [\![\tau']\!]^\delta_\chi$. The result follows since the logical interpretations are closed under applicative equivalence.

Alternatively, suppose e'_1 halts. Then e'_2 halts as well. Thus $(\lambda x{:}\tau_1.e_1)e'_1 \approx e_1[e'_1/x]$ and $(\lambda x{:}\tau_2.e_2)e'_2 \approx e_2[e'_2/x]$. Let σ'_{left} be $\sigma_{left}[x \mapsto e'_1]$ and let σ'_{right} be $\sigma_{right}[x \mapsto e'_2]$. Then $\vdash \delta, \chi, \sigma'_{left}, \sigma'_{right} : (\Gamma, x{:}\tau)$. By induction $\Gamma, x{:}\tau \vdash C[e] \Leftrightarrow C[e'] : \tau'$, so $(\sigma'_{left}(\delta_{left}(C[e])), \sigma'_{right}(\delta_{right}(C[e']))) \in [\![\tau']\!]^\delta_\chi$. That is, $(e_1[e'_1/x], e_2[e'_2/x]) \in [\![\tau']\!]^\delta_\chi$. The result follows by closure under applicative equivalence.

Case 3: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau_1 \to \tau_2) \quad \Gamma \vdash e_2 : \tau_1}{C\, e_2 : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau_2)}$$

By induction, $\Gamma \vdash C[e] \Leftrightarrow C[e'] : \tau_1 \to \tau_2$, so $(\sigma_{left}(\delta_{left}(C[e])), \sigma_{right}(\delta_{right}(C[e'])) \in [\![\tau_1 \to \tau_2]\!]^{\delta}_{\chi}$. By the Fundamental Theorem, $(\sigma_{left}(\delta_{left}(e_2)), \sigma_{right}(\delta_{right}(e_2))) \in [\![\tau_1]\!]^{\delta}_{\chi}$. Therefore $(\sigma_{left}(\delta_{left}(C[e]))\sigma_{left}(\delta_{left}(e_2)), \sigma_{right}(\delta_{right}(C[e']))\sigma_{right}(\delta_{right}(e_2))) \in [\![\tau_2]\!]^{\delta}_{\chi}$. That is, $(\sigma_{left}(\delta_{left}(C[e]e_2)), \sigma_{right}(\delta_{right}(C[e']e_2))) \in [\![\tau_2]\!]^{\delta}_{\chi}$.

Case 4: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau_1) \quad \Gamma \vdash e_1 : \tau_1 \to \tau_2}{e_1\, C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau_2)}$$

By induction, $\Gamma \vdash C[e] \Leftrightarrow C[e'] : \tau_1$, so $(\sigma_{left}(\delta_{left}(C[e])), \sigma_{right}(\delta_{right}(C[e'])) \in [\![\tau_1]\!]^{\delta}_{\chi}$. By the Fundamental Theorem, $(\sigma_{left}(\delta_{left}(e_1)), \sigma_{right}(\delta_{right}(e_1))) \in [\![\tau_1 \to \tau_2]\!]^{\delta}_{\chi}$. Therefore $(\sigma_{left}(\delta_{left}(e_1))\sigma_{left}(\delta_{left}(C[e])), \sigma_{right}(\delta_{right}(e_1))\sigma_{right}(\delta_{right}(C[e']))) \in [\![\tau_2]\!]^{\delta}_{\chi}$. That is, $(\sigma_{left}(\delta_{left}(e_1 C[e])), \sigma_{right}(\delta_{right}(e_1 C[e']))) \in [\![\tau_2]\!]^{\delta}_{\chi}$.

Case 5: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma, \alpha \vdash \tau)}{\Lambda\alpha.C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \forall\alpha.\tau)}$$

Let $\sigma_{left}(\delta_{left}(\Lambda\alpha.C[e]))$ be $\Lambda\alpha.e_1$, and let $\sigma_{right}(\delta_{right}(\Lambda\alpha.C[e']))$ be $\Lambda\alpha.e_2$. Note that both terms halt. Now suppose $\tau_1, \tau_2 \in \mathit{Type}$, and suppose $R \in \mathit{ARel}_{\tau_1,\tau_2}$. Let δ' be $\delta[\alpha \mapsto (\tau_1, \tau_2)]$ and let χ' be $\chi[\alpha \mapsto R]$. Then we wish to show that $((\Lambda\alpha.e_1)[\tau_1], (\Lambda\alpha.e_2)[\tau_2]) \in [\![\tau]\!]^{\delta'}_{\chi'}$.

Observe that $(\Lambda\alpha.e_1)[\tau_1] \approx e_1[\tau_1/\alpha]$ and $(\Lambda\alpha.e_2)[\tau_2] \approx e_2[\tau_2/\alpha]$. Also, δ' is a type environment over (Γ, α), and χ' respects δ', so $\vdash \delta', \chi', \sigma_{left}, \sigma_{right} : (\Gamma, \alpha)$. By induction, $\Gamma, \alpha \vdash C[e] \Leftrightarrow C[e'] : \tau$, so $(\sigma_{left}(\delta'_{left}(C[e])), \sigma_{right}(\delta'_{right}(C[e']))) \in [\![\tau]\!]^{\delta'}_{\chi'}$. That is, $(e_1[\tau_1/\alpha], e_2[\tau_2/\alpha]) \in [\![\tau]\!]^{\delta'}_{\chi'}$. The result then follows by closure under applicative equivalence.

Case 6: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \forall\alpha.\tau') \quad \Gamma \vdash \tau \text{ type}}{C[\tau] : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau'[\tau/\alpha])}$$

By induction, $\Gamma \vdash C[e] \Leftrightarrow C[e'] : \forall\alpha.\tau'$, so $(\sigma_{left}(\delta_{left}(C[e])), \sigma_{right}(\delta_{right}(C[e'])) \in [\![\forall\alpha.\tau']\!]^{\delta}_{\chi}$. Therefore:

$$(\sigma_{left}(\delta_{left}(C[e]))[\delta_{left}(\tau)], \sigma_{right}(\delta_{right}(C[e']))[\delta_{right}(\tau)]) \in [\![\tau']\!]^{\delta[\alpha \mapsto \delta(\tau)]}_{\chi[\alpha \mapsto [\![\tau]\!]^{\delta}_{\chi}]}$$

That is, using substitution and rearranging:

$$(\sigma_{left}(\delta_{left}(C[e][\tau])), \sigma_{right}(\delta_{right}(C[e'][\tau]))) \in [\![\tau'[\tau/\alpha]]\!]^{\delta}_{\chi}$$

Case 7: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau[\mu\alpha.\tau/\alpha])}{\mathtt{in}_{\mu\alpha.\tau} C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \mu\alpha.\tau)}$$

Let $\sigma_{left}(\delta_{left}(C[e]))$ be e_1, and let $\sigma_{right}(\delta_{right}(C[e']))$ be e_2. Then, using the Unrolling Lemma, it suffices to show that $(\mathtt{out}(\mathtt{in}\, e_1), \mathtt{out}(\mathtt{in}\, e_2)) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\chi}$. Observe that $\mathtt{out}(\mathtt{in}\, e_1) \approx e_1$ and $\mathtt{out}(\mathtt{in}\, e_2) \approx e_2$. By induction, $\Gamma \vdash C[e] \Leftrightarrow C[e'] : \tau[\mu\alpha.\tau/\alpha]$, so $(e_1, e_2) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\chi}$. The result then follows by closure under applicative equivalence.

Case 8: Suppose the last rule applied is:

$$\frac{C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \mu\alpha.\tau)}{\mathtt{out}\, C : (\hat{\Gamma} \vdash \hat{\tau}) \Rightarrow (\Gamma \vdash \tau[\mu\alpha.\tau/\alpha])}$$

By induction, $\Gamma \vdash C[e] \Leftrightarrow C[e'] : \mu\alpha.\tau$, so $(\sigma_{left}(\delta_{left}(C[e])), \sigma_{right}(\delta_{right}(C[e']))) \in [\![\mu\alpha.\tau]\!]^{\delta}_{\chi}$. Using the Unrolling Lemma, $(\mathtt{out}\, \sigma_{left}(\delta_{left}(C[e])), \mathtt{out}\, \sigma_{right}(\delta_{right}(C[e']))) \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]^{\delta}_{\chi}$. The result then follows by rearrangement. □

Theorem 5.9 (Logical Equivalence implies Contextual Equivalence) *If* $\Gamma \vdash e \Leftrightarrow e' : \tau$ *then* $\Gamma \vdash e \cong e' : \tau$.

Proof. Suppose $\Gamma \vdash e \Leftrightarrow e' : \tau$ and let $C : (\Gamma \vdash \tau) \Rightarrow (\epsilon \vdash 1)$ be arbitrary. By Lemma 5.8, $\vdash C[e] \Leftrightarrow C[e'] : 1$. Let δ_0, χ_0, σ_0 be the empty type environment, relation environment, and substitution. Then $\vdash \delta_0, \chi_0, \sigma_0, \sigma_0 : \epsilon$, so $(C[e], C[e']) \in [\![1]\!]^{\delta_0}_{\chi_0}$. Hence $C[e]$ halts if and only $C[e']$ halts. □

Corollary 5.10 *Applicative, logical, and contextual equivalence coincide.*

Proof. Immediate from Theorems 4.14, 5.9, and 5.7. □

Note than an immediate consequence of the coincidence of applicative and logical equivalence is that logical equivalence is a congruence, in the sense of Lemma 2.7.

6 Applications

Logical relations may be used to derive equivalences governing well-typed terms. Of particular interest are equivalences arising from parametricity, giving rise to "free theorems" [35,21,20] and consequences of representation independence for abstract types.

6.1 Defined Types

We may extend our results beyond our small set of primitive types using the usual Church encodings. We will omit the type annotations from these derived forms when they are clear from context. The reader is cautioned that the encodings given here do not satisfy the universal characterizations ordinarily associated with these types, essentially because functions in the language are partial, rather than total, and are call-by-value. We will state and prove the properties we need for the examples we consider.

Definition 6.1 (Products)

$$\begin{aligned}
\tau_1 \times \tau_2 &\stackrel{\text{def}}{=} \forall\beta.(\tau_1 \to \tau_2 \to \beta) \to \beta && (\beta \text{ fresh}) \\
\langle e_1 : \tau_1, e_2 : \tau_2 \rangle &\stackrel{\text{def}}{=} \Lambda\beta.\lambda f{:}(\tau_1 \to \tau_2 \to \beta).f\, e_1\, e_2 && (\beta, f \text{ fresh}) \\
\mathtt{prj}_1^{\tau_1 \times \tau_2} e &\stackrel{\text{def}}{=} e[\tau_1](\lambda x{:}\tau_1.\lambda y{:}\tau_2.x) \\
\mathtt{prj}_2^{\tau_1 \times \tau_2} e &\stackrel{\text{def}}{=} e[\tau_2](\lambda x{:}\tau_1.\lambda y{:}\tau_2.y)
\end{aligned}$$

Definition 6.2 (Sums)

$$\begin{aligned}
\tau_1 + \tau_2 &\stackrel{\text{def}}{=} \forall\beta.(\tau_1 \to \beta) \to (\tau_2 \to \beta) \to \beta && (\beta \text{ fresh}) \\
\mathtt{inj}_1^{\tau_1+\tau_2} e &\stackrel{\text{def}}{=} \Lambda\beta.\lambda f{:}(\tau_1 \to \beta).\lambda g{:}(\tau_2 \to \beta).f\, e && (\beta, f, g \text{ fresh}) \\
\mathtt{inj}_2^{\tau_1+\tau_2} e &\stackrel{\text{def}}{=} \Lambda\beta.\lambda f{:}(\tau_1 \to \beta).\lambda g{:}(\tau_2 \to \beta).g\, e && (\beta, f, g \text{ fresh}) \\
\mathtt{case}^{\tau}(e, x{:}\tau_1.e_1, x{:}\tau_2.e_2) &\stackrel{\text{def}}{=} e[\tau](\lambda x{:}\tau_1.e_1)(\lambda x{:}\tau_2.e_2)
\end{aligned}$$

Definition 6.3 (Existentials)

$$\begin{aligned}
\exists\alpha.\tau &\stackrel{\text{def}}{=} \forall\beta.(\forall\alpha.\,\tau \to \beta) \to \beta && (\beta \text{ fresh}) \\
\mathtt{pack}\,(\tau, e)\,\mathtt{as}\,\exists\alpha.\tau_1 &\stackrel{\text{def}}{=} \Lambda\beta.\lambda f{:}(\forall\alpha.\,\tau_1 \to \beta).f[\tau]e && (\beta, f \text{ fresh}) \\
\mathtt{unpack}^{\tau}(\alpha, x{:}\tau_1) = e\,\mathtt{in}\,e' &\stackrel{\text{def}}{=} e[\tau](\Lambda\alpha.\lambda x{:}\tau_1.\,e')
\end{aligned}$$

We may derive logical equivalences over the defined types using the following lemmas:

Lemma 6.4 (Logical Equivalence for Product Introduction)
If $(e_1, e_1') \in [\![\tau_1]\!]^{\delta}_{\chi}$ *and* $(e_2, e_2') \in [\![\tau_2]\!]^{\delta}_{\chi}$ *then* $(\langle e_1, e_2\rangle, \langle e_1', e_2'\rangle) \in [\![\tau_1 \times \tau_2]\!]^{\delta}_{\chi}$.

Proof. Suppose $(e_1, e_1') \in [\![\tau_1]\!]^{\delta}_{\chi}$ and $(e_2, e_2') \in [\![\tau_2]\!]^{\delta}_{\chi}$. Both $\langle e_1, e_2\rangle$ and $\langle e_1', e_2'\rangle$ halt, so let $\tau, \tau' \in \mathit{Type}$ and $R \in \mathit{ARel}_{\tau,\tau'}$ be arbitrary. We wish to show that $(\langle e_1, e_2\rangle[\tau], \langle e_1', e_2'\rangle[\tau']) \in [\![(\tau_1 \to \tau_2 \to \beta) \to \beta]\!]^{\delta[\beta\mapsto(\tau,\tau')]}_{\chi[\beta\mapsto R]}$. Both terms halt, so let $(m, m') \in [\![\tau_1 \to \tau_2 \to \beta]\!]^{\delta[\beta\mapsto(\tau,\tau')]}_{\chi[\beta\mapsto R]}$. It suffices to show that $(\langle e_1, e_2\rangle[\tau]m, \langle e_1', e_2'\rangle[\tau']m') \in R$. We may assume that m and m' halt (otherwise the result is immediate, since R is pointed). Thus, it is sufficient to show that

$(m\,e_1\,e_2, m'\,e_1'\,e_2') \in R$ which follows from the assumption (since β is not free in τ_1 or τ_2). □

Lemma 6.5 (Logical Equivalence for Sum Introduction)

- *If* $(e, e') \in [\![\tau_1]\!]^{\delta}_{\chi}$ *and* $\tau_2 \in \mathit{Type}$ *then* $(\mathtt{inj}_1 e, \mathtt{inj}_1 e') \in [\![\tau_1 + \tau_2]\!]^{\delta}_{\chi}$.
- *If* $(e, e') \in [\![\tau_2]\!]^{\delta}_{\chi}$ *and* $\tau_1 \in \mathit{Type}$ *then* $(\mathtt{inj}_2 e, \mathtt{inj}_2 e') \in [\![\tau_1 + \tau_2]\!]^{\delta}_{\chi}$.

Proof. Similar to Lemma 6.4. □

Lemma 6.6 (Logical Equivalence for Existential Introduction)
If $\tau, \tau' \in \mathit{Type}$ *and* $R \in \mathit{ARel}_{\tau,\tau'}$ *and* $(e, e') \in [\![\tau_1]\!]^{\delta[\alpha \mapsto (\tau,\tau')]}_{\chi[\alpha \mapsto R]}$
then $(\mathtt{pack}\,(\tau, e)\,\mathtt{as}\,\exists\alpha.\tau_1, \mathtt{pack}\,(\tau', e')\,\mathtt{as}\,\exists\alpha.\tau_1) \in [\![\exists\alpha.\tau_1]\!]^{\delta}_{\chi}$.

Proof. Suppose $\tau, \tau' \in \mathit{Type}$ and $R \in \mathit{ARel}_{\tau,\tau'}$ and $(e, e') \in [\![\tau_1]\!]^{\delta[\alpha \mapsto (\tau,\tau')]}_{\chi[\alpha \mapsto R]}$. Both $\mathtt{pack}\,(\tau, e)\,\mathtt{as}\,\exists\alpha.\tau_1$ and $\mathtt{pack}\,(\tau', e')\,\mathtt{as}\,\exists\alpha.\tau_1$ halt. Let $\sigma, \sigma' \in \mathit{type}$ and $Q \in \mathit{ARel}_{\sigma,\sigma'}$ be arbitrary. We wish to show that $((\mathtt{pack}\,(\tau, e)\,\mathtt{as}\,\exists\alpha.\tau_1)[\sigma], (\mathtt{pack}\,(\tau', e')\,\mathtt{as}\,\exists\alpha.\tau_1)[\sigma']) \in [\![(\forall\alpha.\,\tau_1 \to \beta) \to \beta]\!]^{\delta[\beta \mapsto (\sigma,\sigma')]}_{\chi[\beta \mapsto Q]}$. Both terms halt, so let $(m, m') \in [\![\forall\alpha.\,\tau_1 \to \beta]\!]^{\delta[\beta \mapsto (\sigma,\sigma')]}_{\chi[\beta \mapsto Q]}$. It suffices to show that $((\mathtt{pack}\,(\tau, e)\,\mathtt{as}\,\exists\alpha.\tau_1)[\sigma]m, (\mathtt{pack}\,(\tau', e')\,\mathtt{as}\,\exists\alpha.\tau_1)[\sigma']m') \in Q$. We may assume that m and m' halt (otherwise the result is immediate, since Q is pointed). Thus, it is sufficient to show that $(m\,[\tau]\,e, m'\,[\tau']\,e') \in Q$.

Using the definition of the logical relation, we may obtain:

$$(m\,[\tau], m\,[\tau']) \in [\![\tau_1 \to \beta]\!]^{\delta[\beta \mapsto (\sigma,\sigma')][\alpha \mapsto (\tau,\tau')]}_{\chi[\beta \mapsto Q][\alpha \mapsto R]}$$

Using our assumption and the fact that β is not free in τ_1, we have:

$$(e, e') \in [\![\tau_1]\!]^{\delta[\beta \mapsto (\sigma,\sigma')][\alpha \mapsto (\tau,\tau')]}_{\chi[\beta \mapsto Q][\alpha \mapsto R]}$$

Therefore, as desired:

$$(m\,[\tau]\,e, m\,[\tau']\,e') \in [\![\beta]\!]^{\delta[\beta \mapsto (\sigma,\sigma')][\alpha \mapsto (\tau,\tau')]}_{\chi[\beta \mapsto Q][\alpha \mapsto R]} = Q$$

□

In conjunction with Corollary 5.10, Lemma 6.6 gives us a powerful tool for establishing representation independence results.

It is natural to ask whether the converses of the above lemmas hold as well. For products and sums, it is not difficult to prove that the answer is yes.[6] For existentials, however, the answer is probably not, as we illustrate in Section 6.4.

In light of that, it is also natural to wonder whether the constructions in this paper might be carried out with a *primitive* existential type. They cannot, and

[6] Note that this is a distinct question from the one as to whether any value of product (sum) type must take the form on a pair (injection) up to equivalence. We leave the latter question open.

in fact the problem arises quite early. Our proof relies critically on applicative equivalence and it is unclear how even to define it in the presence of existential types. The obvious definition would require that applicatively equivalent existential packages share the same hidden type, but such a definition clearly cannot coincide with contextual equivalence.

6.2 Free Theorems

One of the powers of relational parametricity is to prove *free theorems* [36], theorems regarding the behavior of programs that can be ascertained merely by looking at the program's type. Two simple examples of free theorems are the following, which show that the types $\forall\alpha.\alpha$ and $\forall\alpha.\alpha \to \alpha$ contain only trivial members.

Theorem 6.7 *If* $\vdash e : \forall\alpha.\alpha$ *then* $e \preceq \Lambda\alpha.\bot$.

Proof. Note that $\Lambda\alpha.\bot$ halts. Therefore, suppose $\tau \in \mathit{Type}$. We wish to show that $e[\tau] \preceq \bot$, that is, that $e[\tau]$ diverges. By the Fundamental Theorem, $(e, e) \in [\![\forall\alpha.\alpha]\!]$. Let $R = \{(p : \tau, q : \tau) \mid p{\uparrow} \wedge q{\uparrow}\}$. Observe that R is pointed, complete, and strict. Therefore, $(e[\tau], e[\tau]) \in [\![\alpha]\!]^{[\alpha\mapsto(\tau,\tau)]}_{[\alpha\mapsto R]} = R$. Hence $e[\tau]$ diverges. □

Theorem 6.8 *If* $\vdash e : \forall\alpha.\alpha \to \alpha$ *then* $e \preceq \Lambda\alpha.\lambda x{:}\alpha.x$.

Proof. Note that $\Lambda\alpha.\lambda x{:}\alpha.x$ halts. Therefore, suppose $\tau \in \mathit{Type}$. We wish to show that $e[\tau] \preceq \lambda x{:}\tau.x$. Again, not that $\lambda x{:}\tau.x$ halts. Therefore, suppose $\vdash v : \tau$. We wish to show that $e[\tau]v \preceq v$. By the Fundamental Theorem, $(e, e) \in [\![\forall\alpha.\alpha \to \alpha]\!]$. Let $R = \{(p : \tau, q : \tau) \mid p \approx q \wedge p \preceq v\}$. Observe that R is pointed, complete, and strict. Therefore $(e[\tau], e[\tau]) \in [\![\alpha \to \alpha]\!]^{[\alpha\mapsto(\tau,\tau)]}_{[\alpha\mapsto R]}$. Since $(v, v) \in R = [\![\alpha]\!]^{[\alpha\mapsto(\tau,\tau)]}_{[\alpha\mapsto R]}$, it follows that $(e[\tau]v, e[\tau]v) \in [\![\alpha]\!]^{[\alpha\mapsto(\tau,\tau)]}_{[\alpha\mapsto R]} = R$. Hence $e[\tau]v \preceq v$. □

We can also show that any function with type $\forall\alpha.\alpha \to \tau$ for closed τ is either a constant function or some flavor of nonterminating function.

Theorem 6.9 *If* $\vdash \tau$ type *and* $\vdash e : \forall\alpha.\alpha \to \tau$ *then, up to equivalence,* e *is one of the following:* $\bot$, $\Lambda\alpha.\bot$, $\Lambda\alpha.\lambda x{:}\alpha.\bot$, *or* $\Lambda\alpha.\lambda x{:}\alpha.v$ *(for some* $\vdash v : \tau$*).*

Proof. It suffices to show that (1) if $e[\tau']$ halts for any τ' then it halts for every τ', and (2) if $e[\tau']v$ halts for any τ' and $v \in \mathit{Exp}_{\tau'}$ then it halts and returns an equivalent value for every such τ' and v.

Suppose $e[\tau_1]$ halts, for some $\tau_1 \in \mathit{Type}$. Let $\tau_2 \in \mathit{Type}$ be arbitrary. By the Fundamental Theorem, $(e, e) \in [\![\forall\alpha.\alpha \to \tau]\!]$. Let R be any strict and admissible relation on τ_1 and τ_2. (We are permitted to choose R but the choice does not matter.) Then $(e[\tau_1], e[\tau_2]) \in [\![\alpha \to \tau]\!]^{[\alpha\mapsto(\tau_1,\tau_2)]}_{[\alpha\mapsto R]}$. By Lemma 4.4, the latter relation is strict, so $e[\tau_2]$ halts.

Now suppose $e[\tau_1]v_1 \mapsto^* v_1'$, for some $\tau_1 \in \mathit{Type}$ and $v_1 \in \mathit{Exp}_{\tau_1}$. Let $\tau_2 \in \mathit{Type}$ and $v_2 \in \mathit{Exp}_{\tau_2}$ be arbitrary. By the Fundamental Theorem, $(e, e) \in [\![\forall\alpha.\alpha \to \tau]\!]$. Let $R = \{(p : \tau_1, q : \tau_2) \mid (p{\uparrow} \wedge q{\uparrow}) \vee (p \approx v_1 \wedge q \approx v_2)\}$. Then $(e[\tau_1]v_1, e[\tau_2]v_2) \in$

$[\![\tau]\!]^{[\alpha\mapsto(\tau_1,\tau_2)]}_{[\alpha\mapsto R]}$. Since τ is closed, we may conclude that $e[\tau_1]v_1 \Leftrightarrow e[\tau_2]v_2$, and hence that $e[\tau_1]v_1 \approx e[\tau_2]v_2$. Thus $e[\tau_2]v_2 \mapsto^* v_2'$ for some $v_2' \approx v_1'$. □

For a more interesting example of a free theorem, we borrow from Wadler [35]. Consider the function `head`, which extracts the first element of a list or diverges if the list is empty. One theorem regarding `head` is that mapping a function f over `head`'s argument is equivalent to applying f to `head`'s result. This theorem is free, because it can be ascertained without looking at the code for `head`; it applies to any function with the type $\forall\alpha.\,\alpha\,\texttt{list} \to \alpha$.

Theorem 6.10 *Let us define:*

$$\tau\,\texttt{list} \quad \stackrel{\text{def}}{=} \mu\alpha.1 + (\tau \times \alpha)$$
$$[e_1, \ldots, e_n]_\tau \stackrel{\text{def}}{=} \texttt{in}_{\tau\,\texttt{list}}(\texttt{inj}_2\langle e_1, \ldots \texttt{in}_{\tau\,\texttt{list}}(\texttt{inj}_2\langle e_n, \texttt{in}_{\tau\,\texttt{list}}(\texttt{inj}_1 *)\rangle)\ldots\rangle)$$

Suppose that $\vdash h : \forall\alpha.\,\alpha\,\texttt{list} \to \alpha$. *Suppose further that* $\vdash v_1, \ldots, v_n : \tau$ *and that* $\vdash f : \tau \to \tau'$ *halts and is a total function. Then* $f(h[\tau][v_1, \ldots, v_n]_\tau) \approx h[\tau'][f\,v_1, \ldots, f\,v_n]_{\tau'}$.

Proof. By the Fundamental Theorem, $(h, h) \in [\![\forall\alpha.\,\alpha\,\texttt{list} \to \alpha]\!]$. Let $R = \{(p : \tau, q : \tau') \mid f\,p \approx q\}$. Observe that R is pointed, complete, and strict. Therefore $(h[\tau], h[\tau']) \in [\![\alpha\,\texttt{list} \to \alpha]\!]^{[\alpha\mapsto(\tau,\tau')]}_{[\alpha\mapsto R]}$. Using the Unrolling theorem and Lemmas 6.4 and 6.5, we can show by induction on n that $([v_1, \ldots, v_n]_\tau, [f\,v_1, \ldots, f\,v_n]_{\tau'}) \in [\![\alpha\,\texttt{list}]\!]^{[\alpha\mapsto(\tau,\tau')]}_{[\alpha\mapsto R]}$. Therefore $(h[\tau][v_1, \ldots, v_n]_\tau, h[\tau'][f\,v_1, \ldots, f\,v_n]_{\tau'}) \in R$. By the construction of R, $f(h[\tau][v_1, \ldots, v_n]_\tau) \approx h[\tau'][f\,v_1, \ldots, f\,v_n]_{\tau'}$, as desired. □

6.3 Representation Independence

The use of logical relations to establish representation independence results in the absence of recursive types is well-known. Using our technique we may also obtain results that exploit recursive types, including ones in which the recursive variable is used negatively. To illustrate, we adapt an example from Sumii and Pierce [33].

Let us define $\texttt{nat} \stackrel{\text{def}}{=} \mu\alpha.1 + \alpha$ and $\texttt{bool} \stackrel{\text{def}}{=} 1 + 1$, and suppose that $\texttt{zero} : \texttt{nat}$, $\texttt{succ} : \texttt{nat} \to \texttt{nat}$, $\texttt{even} : \texttt{nat} \to \texttt{bool}$, $\texttt{true} : \texttt{bool}$, $\texttt{false} : \texttt{bool}$, and $\texttt{not} : \texttt{bool} \to \texttt{bool}$ are implemented in the obvious manner. Then consider the following type for flag objects:

$$\texttt{flag} \stackrel{\text{def}}{=} \exists\texttt{st}.\,\mu\texttt{self}.\,\texttt{st} \times ((\texttt{self} \to \texttt{self}) \times (\texttt{self} \to \texttt{bool}))$$

A flag object has an instance variable (belonging to an abstract type `st`), and two methods. The first method returns a new object whose flag is reversed, and the second method returns the state of the flag. Note that both methods access the instance variable only through the recursive `self` variable.

We consider two different implementation of flags, one in which the hidden state is a `bool` and one in which it is a `nat`:

$$\mathtt{fields}_\alpha \stackrel{\text{def}}{=} \mu\mathtt{self}.\,\alpha \times ((\mathtt{self} \to \mathtt{self}) \times (\mathtt{self} \to \mathtt{bool}))$$

$$\mathtt{boolflag} \stackrel{\text{def}}{=} \mathtt{pack}\,(\mathtt{bool}, \mathtt{in}_{\mathtt{fields}_{\mathtt{bool}}}\langle \mathtt{true}, \langle \mathtt{boolflip}, \mathtt{boolret}\rangle\rangle)\,\mathtt{as}\,\mathtt{flag}$$
$$\mathtt{boolflip} \stackrel{\text{def}}{=} \lambda x{:}\mathtt{fields}_{\mathtt{bool}}.\mathtt{in}_{\mathtt{fields}_{\mathtt{bool}}}\langle \mathtt{not}(\mathtt{prj}_1(\mathtt{out}\,x)), \mathtt{prj}_2(\mathtt{out}\,x)\rangle$$
$$\mathtt{boolret} \stackrel{\text{def}}{=} \lambda x{:}\mathtt{fields}_{\mathtt{bool}}.\mathtt{prj}_1(\mathtt{out}\,x)$$

$$\mathtt{natflag} \stackrel{\text{def}}{=} \mathtt{pack}\,(\mathtt{nat}, \mathtt{in}_{\mathtt{fields}_{\mathtt{nat}}}\langle \mathtt{zero}, \langle \mathtt{natflip}, \mathtt{natret}\rangle\rangle)\,\mathtt{as}\,\mathtt{flag}$$
$$\mathtt{natflip} \stackrel{\text{def}}{=} \lambda x{:}\mathtt{fields}_{\mathtt{nat}}.\mathtt{in}_{\mathtt{fields}_{\mathtt{nat}}}\langle \mathtt{succ}(\mathtt{prj}_1(\mathtt{out}\,x)), \mathtt{prj}_2(\mathtt{out}\,x)\rangle$$
$$\mathtt{natret} \stackrel{\text{def}}{=} \lambda x{:}\mathtt{fields}_{\mathtt{nat}}.\mathtt{even}(\mathtt{prj}_1(\mathtt{out}\,x))$$

Using Lemma 6.6 we can show that `boolflag` and `natflag` are logically equivalent. It will then follow by Corollary 5.10 that they are operationally indistinguishable.

Theorem 6.11 $\vdash \mathtt{boolflag} \Leftrightarrow \mathtt{natflag} : \mathtt{flag}$

Proof. Unwinding the definitions, we wish to show that $(\mathtt{boolflag}, \mathtt{natflag}) \in [\![\exists \mathtt{st}.\mathtt{fields}_{\mathtt{st}}]\!]$. By Lemma 6.6, it suffices to exhibit a relation $R \in ARel_{\mathtt{bool},\mathtt{nat}}$ such that:

$$\begin{array}{l}(\mathtt{in}_{\mathtt{fields}_{\mathtt{bool}}}\langle \mathtt{true}, \langle \mathtt{boolflip}, \mathtt{boolret}\rangle\rangle,\\ \mathtt{in}_{\mathtt{fields}_{\mathtt{nat}}}\langle \mathtt{zero}, \langle \mathtt{natflip}, \mathtt{natret}\rangle\rangle) \quad \in [\![\mathtt{fields}_{\mathtt{st}}]\!]^\delta_\chi\end{array}$$

where $\chi = [\mathtt{st} \mapsto R]$ and $\delta = [\mathtt{st} \mapsto (\mathtt{bool}, \mathtt{nat})]$. Let $\overline{n} \stackrel{\text{def}}{=} \underbrace{\mathtt{succ}(\cdots(\mathtt{succ}}_{n \text{ times}}\,\mathtt{zero})\cdots)$, and let:

$$\begin{array}{rl}R = \{(p : \mathtt{bool}, q : \mathtt{nat}) \mid & p{\downarrow} \Leftrightarrow q{\downarrow}\\ & \wedge\, p{\downarrow} \Rightarrow \exists \mathtt{nat} \in \mathbb{N}.\,(p \approx \mathtt{true} \wedge q \approx \overline{2n}) \vee\\ & \qquad (p \approx \mathtt{false} \wedge q \approx \overline{2n+1})\}\end{array}$$

Observe that R is pointed, complete, and strict. By the Unrolling theorem, and cancelling the recursive roll and unroll on each side, it is sufficient to show that:

$$\begin{array}{l}(\langle \mathtt{true}, \langle \mathtt{boolflip}, \mathtt{boolret}\rangle\rangle, \langle \mathtt{zero}, \langle \mathtt{natflip}, \mathtt{natret}\rangle\rangle)\\ \quad \in [\![\mathtt{st} \times ((\mathtt{fields}_{\mathtt{st}} \to \mathtt{fields}_{\mathtt{st}}) \times (\mathtt{fields}_{\mathtt{st}} \to \mathtt{bool}))]\!]^\delta_\chi\end{array}$$

Using Lemma 6.4, it remains to show equivalences for each field:

- Clearly $(\texttt{true}, \texttt{zero}) \in R = [\![\texttt{st}]\!]^{\delta}_{\chi}$.
- We wish to show $(\texttt{boolflip}, \texttt{natflip}) \in [\![\texttt{fields}_{\texttt{st}} \to \texttt{fields}_{\texttt{st}}]\!]^{\delta}_{\chi}$. Both terms halt, so suppose $(m, m') \in [\![\texttt{fields}_{\texttt{st}}]\!]^{\delta}_{\chi}$. By the Unrolling theorem, $(\texttt{out}\, m, \texttt{out}\, m') \in [\![\texttt{st} \times ((\texttt{fields}_{\texttt{st}} \to \texttt{fields}_{\texttt{st}}) \times (\texttt{fields}_{\texttt{st}} \to \texttt{bool}))]\!]^{\delta}_{\chi}$. It follows[7] that $(\texttt{prj}_1(\texttt{out}\, m), \texttt{prj}_1(\texttt{out}\, m')) \in [\![\texttt{st}]\!]^{\delta}_{\chi} = R$ and $(\texttt{prj}_2(\texttt{out}\, m), \texttt{prj}_2(\texttt{out}\, m')) \in [\![(\texttt{fields}_{\texttt{st}} \to \texttt{fields}_{\texttt{st}}) \times (\texttt{fields}_{\texttt{st}} \to \texttt{bool})]\!]^{\delta}_{\chi}$.

  By the construction of R, $(\texttt{not}(\texttt{prj}_1(\texttt{out}\, m)), \texttt{succ}(\texttt{prj}_1(\texttt{out}\, m'))) \in R = [\![\texttt{st}]\!]^{\delta}_{\chi}$. Re-assembling the pieces, we obtain:

$$(\langle \texttt{not}(\texttt{prj}_1(\texttt{out}\, m)), \texttt{prj}_2(\texttt{out}\, m)\rangle, \langle \texttt{succ}(\texttt{prj}_1(\texttt{out}\, m')), \texttt{prj}_2(\texttt{out}\, m')\rangle)$$
$$\in [\![\texttt{st} \times ((\texttt{fields}_{\texttt{st}} \to \texttt{fields}_{\texttt{st}}) \times (\texttt{fields}_{\texttt{st}} \to \texttt{bool}))]\!]^{\delta}_{\chi}$$

  Again using the Unrolling lemma and cancelling rolls and unrolls, we may conclude that $(\texttt{boolflip}\ m, \texttt{natflip}\ m) \in [\![\texttt{fields}_{\texttt{st}}]\!]^{\delta}_{\chi}$. Therefore, $(\texttt{boolflip}, \texttt{natflip}) \in [\![\texttt{fields}_{\texttt{st}} \to \texttt{fields}_{\texttt{st}}]\!]^{\delta}_{\chi}$, as desired.
- We wish to show $(\texttt{boolret}, \texttt{natret}) \in [\![\texttt{fields}_{\texttt{st}} \to \texttt{bool}]\!]^{\delta}_{\chi}$. Both terms halt, so suppose $(m, m') \in [\![\texttt{fields}_{\texttt{st}}]\!]^{\delta}_{\chi}$. By the Unrolling theorem, $(\texttt{out}\, m, \texttt{out}\, m') \in [\![\texttt{st} \times ((\texttt{fields}_{\texttt{st}} \to \texttt{fields}_{\texttt{st}}) \times (\texttt{fields}_{\texttt{st}} \to \texttt{bool}))]\!]^{\delta}_{\chi}$. It follows that $(\texttt{prj}_1(\texttt{out}\, m), \texttt{prj}_1(\texttt{out}\, m')) \in [\![\texttt{st}]\!]^{\delta}_{\chi} = R$.

  By the construction of R, $\texttt{prj}_1(\texttt{out}\, m) \approx \texttt{even}(\texttt{prj}_1(\texttt{out}\, m'))$. Since the logical relation respects applicative equivalence, we may conclude that $(\texttt{boolret}\, m, \texttt{natret}\, m) \in [\![\texttt{bool}]\!]^{\delta}_{\chi}$. Therefore, $(\texttt{boolret}, \texttt{natret}) \in [\![\texttt{fields}_{\texttt{st}} \to \texttt{bool}]\!]^{\delta}_{\chi}$, as desired.

□

Corollary 6.12 `boolflag` *and* `natflag` *are contextually equivalent.*

Proof. Immediate, by Corollary 5.10. □

6.4 Limitations Regarding Existential Types

Corollary 5.10 implies that logical equivalence is a general strategy for proving contextual equivalences. The preceding example illustrates the use of that corollary in conjunction with Lemma 6.6 (logical equivalence for existential introduction) to prove an contextual equivalence result for two existential packages. A natural question to ask is whether this is a general technique for proving contextual equivalences of existential packages.

Unfortunately, the answer is probably not. Consider the following example, due to Pitts [17, 7.7.4]. Let $\texttt{void} \stackrel{\text{def}}{=} \mu\alpha.\alpha$, and note that `void` contains no values, and hence no terms that halt. (Since `in` is strict, it is easy to prove this by induction on

[7] Using an easy argument regarding logical equivalence and product elimination.

typing derivations.) Also, let `if-then-else` and `andalso` be defined in the obvious manner. Then, define two existential packages, `voidpkg` and `boolpkg`:

$$
\begin{array}{lcl}
T & \stackrel{\text{def}}{=} & \exists\alpha.(\alpha \to \texttt{bool}) \to 1 \\
\texttt{voidfn} & \stackrel{\text{def}}{=} & \lambda f{:}\texttt{void} \to \texttt{bool}.\, \bot \\
\texttt{boolfn} & \stackrel{\text{def}}{=} & \lambda f{:}\texttt{bool} \to \texttt{bool}.\texttt{if}\, f\, \texttt{true}\ \texttt{andalso}\ \texttt{not}(f\, \texttt{false}) \\
 & & \qquad\qquad \texttt{then} * \texttt{else}\ \bot \\
\texttt{voidpkg} & \stackrel{\text{def}}{=} & \texttt{pack}\,(\texttt{void}, \texttt{voidfn})\ \texttt{as}\, T \\
\texttt{boolpkg} & \stackrel{\text{def}}{=} & \texttt{pack}\,(\texttt{bool}, \texttt{boolfn})\ \texttt{as}\, T
\end{array}
$$

These two packages appear[8] to be contextually equivalent. Intuitively, the packages should be contextually equivalent because any surrounding context can call the enclosed function only on a parametric function $\alpha \to \texttt{bool}$. Up to equivalence, the only such functions are the two constant functions, the everywhere divergent function, and $\bot$. (Recall Theorem 6.9 above.) For each of these, `voidfn` and `boolfn` behave the same, because `boolfn` is crafted to diverge when given a constant function.

However, the contextual equivalence of these packages cannot be proven using Lemma 6.6. To use the Lemma, we need to exhibit an $R \in ARel_{\texttt{void},\texttt{bool}}$ such that $(\texttt{voidfn}, \texttt{boolfn}) \in [\![(\alpha \to \texttt{bool}) \to 1]\!]^{[\alpha\mapsto(\texttt{void},\texttt{bool})]}_{[\alpha\mapsto R]}$. Since `void` contains no terms that halt, the only strict relation on `void` and `bool` is $\{(p : \texttt{void}, q : \texttt{bool}) \mid p{\uparrow} \wedge q{\uparrow}\}$. Let R be this relation.

The problem is that R provides no assistance in narrowing the set of functions $\alpha \to \texttt{bool}$ that might be used as arguments to `voidfn` and `boolfn`. In particular, observe that $(\lambda x{:}\texttt{void}.\texttt{true}, \lambda x{:}\texttt{bool}.x) \in [\![\alpha \to \texttt{bool}]\!]^{[\alpha\mapsto(\texttt{void},\texttt{bool})]}_{[\alpha\to R]}$. Since `voidfn` and `boolfn` do not behave the same on these two functions, we must conclude that $(\texttt{voidfn}, \texttt{boolfn}) \notin [\![(\alpha \to \texttt{bool}) \to 1]\!]^{[\alpha\mapsto(\texttt{void},\texttt{bool})]}_{[\alpha\mapsto R]}$. Hence, Lemma 6.6 is inapplicable.

It is not clear how important this issue is. The example seems to rely crucially on the fact that the type T contains no strictly positive occurrences of the hidden type variable α. This is a situation that would arise rarely if ever in normal use of representation independence in modularity, for example, in proving equivalence of two implementations of an abstract data type. Thus, we might conjecture that the characterization of existential types in Lemma 6.6 is complete for types $\exists\alpha.\tau$ in which α has a strictly positive occurrence in τ.

7 Related Work

There is a large body of work on the use of logical relations in the study of the syntax and semantics of typed languages. Of most immediate relevance is the work

[8] We have not proven this, but Pitts [17] sketches a proof for a different but similar language.

of Pitts on developing operationally based theories of expression equivalence for PCF-like languages [18]. In that setting, as here, logical, contextual, and applicative equivalence coincide. More recently Pitts has extended this work to polymorphic languages [21] and languages with abstract types [20]. Rather than work with admissible relations as we do here, Pitts relies on a related closure condition that facilitates handling of the continuation-based elimination form for existential types. Using this he obtains a complete characterization of contextual equivalence in terms of logical equivalence, and uses this to obtain examples similar to those considered here.

The methods used here are influenced by Pitts's work on relational properties of domains, and by Birkedal and Harper's [4] operational account of logical relations for a functional language with a single recursive type. The present work generalizes this earlier work to account for impredicative polymorphism and unrestricted recursive types, and, *en passant*, gives a new, streamlined proof of syntactic minimal invariance that may be of use in other settings. The treatment of projections for abstract types as the identity was inspired by Riecke [30].

Ho [9] and Filinski [5] each exploit Pitts's technique in building a domain-theoretic semantics of recursive types for the purpose of proving results in operational semantics. In Ho, the central result is the algebraic compactness of a "syntactic" category, in essence proving syntactic minimal invariance by way of an adequate operational semantics. In Filinski, the central result is a re-presentation of Reynolds's seminal result [27] on the coincidence of direct and continuation semantics. Both find that in the construction of the relational interpretation, it is necessary (or at least beneficial) not to use the entire category $\mathcal{C}^{\mathrm{op}} \times \mathcal{C}$, but its full subcategory consisting of objects along the "diagonal" (that is, objects $\langle A, A \rangle$ for objects A of $\mathcal{C}$). In our setting, this corresponds to the proviso in Theorem 4.10 that $\eta = \eta^{\mathrm{op}}$.

Vouillon and Melliès [34] use a technique similar to Birkedal and Harper to construct an ideal model for quantified types in the presence of subtyping. Like Birkedal and Harper's construction, and in contrast to ours, their construction is based on a single recursive type (representing the universal domain of untyped terms in Vouillon and Melliès's case) rather than a general recursive type operator.

Appel and McAllester have also considered an operationally-based relational interpretation of types, but with the emphasis on proving safety, rather than equivalence, and for low-level, imperative languages, rather than high-level functional languages [3]. Their approach is based on a form of indexed semantics that is broadly similar to our use of projections, but the precise relationship is not clear. In particular Appel and McAllester do not need to quotient terms by an operational congruence as we do here; for us, this is essential to the proof of syntactic minimal invariance. An open question regarding Appel and McAllester's relational interpretation is whether it is actually an equivalence relation; in particular, whether it is transitive. Ahmed [2] gives a related construction that enjoys transitivity by adding additional typing assumptions, and also extends the method to support quantified types.

References

[1] Abadi, M. and G. D. Plotkin, *A per model of polymorphism and recursive types*, in: LICS [13], pp. 355–365.

[2] Ahmed, A., *Step-indexed syntactic logical relations for recursive and quantified types*, in: *Fifteenth European Symposium on Programming*, Vienna, Austria, 2006, pp. 69–83.

[3] Appel, A. W. and D. A. McAllester, *An indexed model of recursive types for foundational proof-carrying code.*, ACM Trans. Program. Lang. Syst. **23** (2001), pp. 657–683.

[4] Birkedal, L. and R. Harper, *Relational interpretations of recursive types in an operational setting*, Information and Computation **155** (1999), pp. 3–63.

[5] Filinski, A., *On the relations between monadic semantics*, in: *John Reynolds Festschrift*, 2007 To appear.

[6] Freyd, P. J., *Recursive types reduced to inductive types*, in: LICS [13], pp. 498–507.

[7] Girard, J.-Y., "Interprétation Fonctionnelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieure," Ph.D. thesis, Université Paris VII (1972).

[8] Girard, J.-Y., Y. Lafont and P. Taylor, "Proofs and Types," Cambridge Tracts in Theoretical Computer Science **7**, Cambridge University Press, Cambridge, England, 1989.

[9] Ho, W. K., *An operational domain-theoretic treatment of recursive types*, in: *Twenty-Second Mathematical Foundations of Programming Semantics*, 2006.

[10] Hoare, C. A. R., *Proof of correctness of data representation*, Artificial Intelligence **1** (1972), pp. 271–281.

[11] Howe, D. J., *Equality in lazy computation systems*, in: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science* (1989), pp. 198–203.
URL citeseer.ist.psu.edu/howe89equality.html

[12] Howe, D. J., *Proving congruence of bisimulation in functional programming languages.*, Inf. Comput. **124** (1996), pp. 103–112.

[13] "Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA," IEEE Computer Society, 1990.

[14] Mason, I. A., S. F. Smith and C. L. Talcott, *From operational semantics to domain theory*, Information and Computation **128** (1996), pp. 26–47.

[15] Mitchell, J. C., *Representation independence and data abstraction*, in: *Thirteenth ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, 1986, pp. 263–276.

[16] Mitchell, J. C. and G. D. Plotkin, *Abstract types have existential type.*, ACM Trans. Program. Lang. Syst. **10** (1988), pp. 470–502.

[17] Pitts, A., *Typed operational reasoning*, in: B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, mit-press, 2005 pp. 245–289.

[18] Pitts, A. M., *Operationally-based theories of program equivalence*, in: P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*, Cambridge University Press, 1995 pp. 241–283.
URL citeseer.ist.psu.edu/113777.html

[19] Pitts, A. M., *Relational properties of domains*, Information and Computation **127** (1996), pp. 66–90.

[20] Pitts, A. M., *Existential types: Logical relations and operational equivalence.*, in: K. G. Larsen, S. Skyum and G. Winskel, editors, *ICALP*, Lecture Notes in Computer Science **1443** (1998), pp. 309–326.

[21] Pitts, A. M., *Parametric polymorphism and operational equivalence.*, Mathematical Structures in Computer Science **10** (2000), pp. 321–359.

[22] Plotkin, G. D., *LCF considered as a programming language.*, Theor. Comput. Sci. **5** (1977), pp. 225–255.

[23] Plotkin, G. D., *Domain theory* (1983), unpublished Notes.
URL http://homepages.inf.ed.ac.uk/gdp/publications/Domains.ps

[24] Plotkin, G. D., *The origins of structural operational semantics.*, J. Log. Algebr. Program. **60-61** (2004), pp. 3–15.

[25] Plotkin, G. D., *A structural approach to operational semantics.*, J. Log. Algebr. Program. **60-61** (2004), pp. 17–139.

[26] Plotkin, G. D. and M. Abadi, *A logic for parametric polymorphism.*, in: M. Bezem and J. F. Groote, editors, *TLCA*, Lecture Notes in Computer Science **664** (1993), pp. 361–375.

[27] Reynolds, J. C., *On the relation between direct and continuation semantics*, in: *Second Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **14** (1974), pp. 141–156.

[28] Reynolds, J. C., *Towards a theory of type structure*, in: *Colloq. sur la Programmation*, Lecture Notes in Computer Science **19** (1974), pp. 408–423.

[29] Reynolds, J. C., *Types, abstraction, and parametric polymorphism*, in: R. E. A. Mason, editor, *Information Processing '83* (1983), pp. 513–523.

[30] Riecke, J. G. and R. Subrahmanyam, *Semantic orthogonality of type disciplines* (1997), (Unpublished manuscript.).
URL citeseer.ist.psu.edu/riecke97semantic.html

[31] Smyth, M. B. and G. D. Plotkin, *The category-theoretic solution of recursive domain equations.*, SIAM J. Comput. **11** (1982), pp. 761–783.

[32] Statman, R., *Logical relations and the typed λ-calculus*, Information and Control **65** (1985), pp. 85–97.

[33] Sumii, E. and B. C. Pierce, *A bisimulation for type abstraction and recursion*, in: *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2005), pp. 63–74.

[34] Vouillon, J. and P.-A. Melliès, *Semantic types: A fresh look at the ideal model for types*, in: *Thirty-First ACM Symposium on Principles of Programming Languages*, Venice, Italy, 2004, pp. 52–63.

[35] Wadler, P., *Theorems for free!*, in: *FPCA '89: Proceedings of the fourth international conference on Functional programming languages and computer architecture* (1989), pp. 347–359.

[36] Wadler, P., *Theorems for free!*, in: *Fourth Conference on Functional Programming Languages and Computer Architecture*, London, 1989.