Relational Interpretations of Recursive Types in an Operational Setting*
(Summary)

Lars Birkedal and Robert Harper

birkedal@cs.cmu.edu and rwh@cs.cmu.edu
School of Computer Science, Carnegie Mellon University

Abstract. Relational interpretations of type systems are a useful tool for establishing properties of programming languages. For languages with recursive types the existence of a relational interpretation is often difficult to establish. The most well-known approach is to pass to a domain-theoretic model of the language, using the structure of the domain to define a suitable system of relations. Here we study the construction of relational interpretations for an ML-like language with recursive functions and recursive types in a purely operational setting. The construction is an adaptation of results of Pitts on relational properties of domains to an operational setting, making use of techniques introduced by Mason, Smith, and Talcott for proving operational equivalence of expressions. To illustrate the method we give a relational proof of correctness of the continuation-passing transformation used in some compilers for functional languages.

1 Introduction

The interpretation of types as relations is a fundamental technique in the study of type systems (see, for example, Mitchell's survey [18] and monograph [19] for examples and references to the literature). The general idea is to associate to each type a relation over a suitable value space in such a way that well-typed terms are related appropriately by the interpretation. The construction of relational interpretations of type systems often raises interesting technical problems. For example, Girard's proof of strong normalization for the second-order λ-calculus [10] may be understood as a relational interpretation for a type system with impredicative type quantification.

In this paper we are concerned with the construction of relational interpretations for an ML-like language $\mathcal{L}$ with recursive functions and one recursive type. The operational semantics of the language specifies an "eager" or "call-by-value" evaluation strategy, as in Standard ML [17]. We make no restrictions on the occurrence of the recursively-defined type in its definition — both positive and negative occurrences are permitted. This complicates the construction of a relational interpretation of the language.

* Preprint of an article to appear in *TACS '97*.

The usual approach is to pass to a specific model (for example, a domain-theoretic model such as Scott's D_∞) and to exploit the structure of the model to construct the required system of relations. One disadvantage of this approach is that one must then give a denotational semantics for the language in the model under consideration, and this must be proved adequate with respect to the operational semantics. Another disadvantage is that the interpretation is defined for a specific model, and it is not clear to what extent the result applies to other models of the language. The question of generality was recently addressed by Pitts [22], who exploited Freyd's analysis of solutions of recursive domain equations [8,7,9] to construct relational interpretations over domain models of recursive types. In particular, Pitts showed that the existence of a relational interpretation can, under very general conditions, be reduced to the *minimal invariant* property of solutions to recursive domain equations given by Freyd. Very roughly, the minimal invariant property characterizes the "minimal" solution to a recursive domain equation by a universal property.

The starting point for our work is the observation that the minimal invariant property for a model of $\mathcal{L}$ can be stated entirely in terms of the constructs of the language itself. This opens the way to carrying out the construction of a relational interpretation of the type system in a purely operational setting — that is, without consideration of a domain-theoretic denotational semantics of the language. The key is to establish a "syntactic minimal invariant" property for terms of the language taken modulo a suitable notion of operational equivalence. With this in hand we may adapt Pitts's results to construct relational interpretations of types over operational equivalence classes of closed terms. The choice of operational equivalence is guided by the requirements of the proof. Candidates for operational equivalence include contextual equivalence [20,23] (coincidence of evaluation in all program contexts), bisimilarity [13,21] (existence of a correspondence between evaluation steps), and experimental equivalence [15] (coincidence of closed instances in all evaluation contexts). It turns out that all three notions coincide for the language $\mathcal{L}$, so our decision to work with experimental equivalence is entirely pragmatic — it supported a relatively straightforward proof of the critical minimal invariant property for $\mathcal{L}$.

Relational interpretations of types have a number of applications. Pitts [22] uses relations to characterize the approximation relation in minimal domain models of FPC and to give a proof of adequacy of the denotational semantics for FPC in a minimal domain model relative to an operational semantics for it. Here we focus on the application to the correctness of program transformations used in compilers of functional languages. In particular, we consider the correctness of the translation into *continuation-passing style* [6,23], called the *cps transform*. The proof relies on the construction of a relational interpretation of $\mathcal{L}$ that establishes a correspondence between the evaluation of a program and its continuation-passing transform. The result generalizes Reynolds's proof [25] of the relation between direct and continuation semantics for an untyped language to the case of a typed language with a recursive type (which could be taken to be the recursive type corresponding to the untyped λ-calculus). In constrast to

Reynolds' proof we do not rely on a specific domain-theoretic interpretation of $\mathcal{L}$, but instead work directly over the operational semantics.

This paper is organized as follows. In Section 2 we define the syntax of the language $\mathcal{L}$, define the operational semantics and show some standard typing properties, including type soundness. Then in Section 3 we define the notion of operational equivalence, with which we shall be working in the remainder of the paper. The main result of this section is the proof of syntactic minimal invariance based on a technique introduced by Mason, Talcott, and Smith [15]. In Section 4 we define a universe of admissible relations over operational equivalence classes of closed expressions. We also define relational operators corresponding to the type constructors of the language and show that they preserve admissibility. The relational constructors are used in Section 5 where we construct a relational interpretation of types using the method described above. In Section 6 we apply the method to give a proof of correctness of the cps transformation. Finally, in Section 7 we discuss related work, and in Section 8 we conclude.

2 The Language

The language, $\mathcal{L}$, is a simply-typed fragment of ML with one top-level recursive type. We let x and f range over a set Var of program variables. The syntax of the language is given by the following grammar:

$$
\begin{array}{lll}
\textit{Types} & \tau ::= & 0 \mid 1 \mid \rho \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2 \mid \tau_1 \rightharpoonup \tau_2 \\
\textit{Expressions} & e ::= & v \mid \mathsf{in}e \mid \mathsf{out}e \mid \mathsf{inl}_\tau e \mid \mathsf{inr}_\tau e \mid \mathsf{case}(e_1, e_2, e_3) \mid \\
 & & (e_1, e_2) \mid \mathsf{fst}e \mid \mathsf{snd}e \mid e_1\, e_2 \mid \\
\textit{Values} & v ::= & * \mid \mathsf{in}v \mid \mathsf{inl}_\tau v \mid \mathsf{inr}_\tau v \mid \\
 & & (v_1, v_2) \mid \mathsf{fix}f(x{:}\tau).e \\
\textit{Evaluation} & E ::= & _ \mid \mathsf{in}E \mid \mathsf{out}E \mid \mathsf{inl}_\tau E \mid \mathsf{inr}_\tau E \mid \mathsf{case}(E, e, e') \mid \\
\textit{Contexts} & & (E, e) \mid (v, E) \mid \mathsf{fst}E \mid \mathsf{snd}E \mid E\, e \mid v\, E
\end{array}
$$

The $\mathcal{L}$ *raw terms* are given by the syntax trees generated by the grammar above, with e as start symbol, modulo α-equivalence, as usual. Alpha-equivalence is denoted $\equiv_\alpha$. Observe that ρ is a type constant. Distinguish a fixed type expression τ_ρ, the intuition being that ρ is a recursive type isomorphic to τ_ρ; in and out are used to mediate the isomorphism.

A *typing context* is a finite function from variables to types; we use Γ to range over typing contexts. If $x \notin \mathrm{Dom}(\Gamma)$, then $\Gamma[x : \tau]$ denotes the typing context that assigns τ to x and $\Gamma(y)$ to all other variables $y \neq x$. A typing judgment has the form $\Gamma \vdash e : \tau$. The typing rules are given in Figure 1. We write $\vdash e : \tau$ for $\emptyset \vdash e : \tau$. The $\mathcal{L}$ *terms* is the set of raw terms e for which there exists, for each e, a typing context Γ and a type τ such that $\Gamma \vdash e : \tau$.

Note that, even though there is no explicit introduction rule for the type 0, there are terms of this type, for instance $(\mathsf{fix}f(x{:}1).f\, x)\, *$.

The set of expressions of type τ with free variables given types by Γ, denoted $\mathrm{Exp}_\tau(\Gamma)$ is defined as follows.

$$\mathrm{Exp}_\tau(\Gamma) \stackrel{\mathrm{def}}{=} \{\, e \mid \Gamma \vdash e : \tau \,\}$$

$$\Gamma \vdash x : \tau \quad (\Gamma(x) = \tau) \qquad \text{(T-VAR)}$$

$$\Gamma \vdash * : 1 \qquad \text{(T-ONE)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \qquad \text{(T-PROD)}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst} e : \tau_1} \qquad \text{(T-FST)}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{snd} e : \tau_2} \qquad \text{(T-SND)}$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}_{\tau_2} e : \tau_1 + \tau_2} \qquad \text{(T-INL)}$$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}_{\tau_1} e : \tau_1 + \tau_2} \qquad \text{(T-INR)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_1 \rightharpoonup \tau \qquad \Gamma \vdash e_3 : \tau_1 \rightharpoonup \tau}{\Gamma \vdash \mathsf{case}(e_1, e_2, e_3) : \tau} \qquad \text{(T-CASE)}$$

$$\frac{\Gamma[f : \tau_1 \rightharpoonup \tau_2][x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash \mathsf{fix} f(x{:}\tau_1).e : \tau_1 \rightharpoonup \tau_2} \quad (f, x \notin \mathrm{Dom}(\Gamma)) \qquad \text{(T-FIX)}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \rightharpoonup \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-APP)}$$

$$\frac{\Gamma \vdash e : \rho}{\Gamma \vdash \mathsf{out} e : \tau_\rho} \qquad \text{(T-OUT)}$$

$$\frac{\Gamma \vdash e : \tau_\rho}{\Gamma \vdash \mathsf{in} e : \rho} \qquad \text{(T-IN)}$$

Fig. 1. Typing Rules

Further define

$$\mathrm{Exp}_\tau \stackrel{\mathrm{def}}{=} \mathrm{Exp}_\tau(\emptyset)$$

Likewise, we define sets for values as follows.

$$\mathrm{Val}_\tau(\Gamma) \stackrel{\mathrm{def}}{=} \{\, v \mid \Gamma \vdash v : \tau \,\}$$

and

$$\mathrm{Val}_\tau \stackrel{\mathrm{def}}{=} \mathrm{Val}_\tau(\emptyset)$$

Substitution of a value v for free occurrences of x in e is written $[v/x]e$. We let $\mathrm{FV}(e)$ denote the set of free variables in e. We use $\lambda x{:}\tau.e$ as an abbreviation for $\mathsf{fix} f(x{:}\tau).e$ when $f \notin \mathrm{FV}(e)$.

2.1 Contexts

The $\mathcal{L}$ contexts, ranged over by C, are the syntax tree generated by the grammar for e augmented by the clause

$$C ::= \cdots \mid \mathsf{p}$$

where p ranges over some fixed set of parameters. Note that the syntax trees of $\mathcal{L}$ terms are contexts, namely the ones with no occurrence of parameters. $[C/\mathsf{p}]C'$ denotes the context obtained from context C' by replacing all occurrences of p in C' with C. This may involve capture of variables.

Notation Most of the time we will only use contexts involving a single parameter which we will write as $_$. We write $C\{_\}$ to indicate that C is a context containing no parameters other than $_$ (note that it may contain no parameters at all). If e is an $\mathcal{L}$ term, then $C\{e\}$ denotes the raw term resulting from choosing a representative syntax tree for e, substituting it for the parameter in c and forming the α-equivalence class of the resulting $\mathcal{L}$ syntax tree (which by the remarks above is independent of the choice of representative for e).

2.2 Typed Contexts

We will assume given a funtion that assigns types to parameters. We write $_\tau$ to indicate that a parameter $_$ has type τ.

The relation $\Gamma \vdash C : \tau$ is inductively generated by axioms and rules just like those defining $\Gamma \vdash e : \tau$ together with the following axiom for parameters.

$$\Gamma \vdash _\tau : \tau \qquad \text{(T-PAR)}$$

The set of contexts of type τ with free variables given types by Γ, denoted $\mathrm{Ctx}_\tau(\Gamma)$ is defined as follows.

$$\mathrm{Ctx}_\tau(\Gamma) \stackrel{\mathrm{def}}{=} \{\, C \mid \Gamma \vdash C : \tau \,\}$$

$$\mathrm{Ctx}_\tau \stackrel{\mathrm{def}}{=} \mathrm{Ctx}_\tau(\emptyset)$$

2.3 Evaluation

The operational semantics will be given by term rewriting and will be defined for all closed terms (not only those of ground type).

The set of *evaluation contexts* are the syntax trees generated by the grammar for E. Note that this is clearly a subset of the set of contexts (with parameters including $_$). Hence we shall use the notation associated with contexts for evaluation contexts also. In addition we define

$$\mathrm{ECtx}_\tau(\Gamma) \stackrel{\text{def}}{=} \{\, E \mid \Gamma \vdash E : \tau \,\}$$

and

$$\mathrm{ECtx}_\tau \stackrel{\text{def}}{=} \mathrm{ECtx}_\tau(\emptyset)$$

Note that evaluation contexts are not capturing.

Redices are generated by the following grammar.

$$\begin{array}{rl} \textit{Redices } r ::= & (\mathsf{fix} f(x{:}\tau).e)\, v \mid \mathsf{fst}(v_1, v_2) \mid \mathsf{snd}(v_1, v_2) \mid \\ & \mathsf{out}(\mathsf{in} v) \mid \mathsf{case}(\mathsf{inl}_\tau v, e_1, e_2) \mid \mathsf{case}(\mathsf{inr}_\tau v, e_1, e_2) \end{array}$$

Note that the set of redices is a subset of the set of expressions. We define

$$\mathrm{Rexp}_\tau(\Gamma) \stackrel{\text{def}}{=} \{\, r \mid \Gamma \vdash r : \tau \,\}$$

and

$$\mathrm{Rexp}_\tau \stackrel{\text{def}}{=} \mathrm{Rexp}_\tau(\emptyset)$$

The reduction rules for redices are as follows.

$$\begin{array}{lll} (\mathsf{fix} f(x{:}\tau).e)\, v & \rightsquigarrow [\mathsf{fix} f(x{:}\tau).e, v/f, x]e & (\text{R-BETA}) \\ \mathsf{fst}(v_1, v_2) & \rightsquigarrow v_1 & (\text{R-FST}) \\ \mathsf{snd}(v_1, v_2) & \rightsquigarrow v_2 & (\text{R-SND}) \\ \mathsf{out}(\mathsf{in} v) & \rightsquigarrow v & (\text{R-OUT}) \\ \mathsf{case}(\mathsf{inl}_\tau v, e_1, e_2) & \rightsquigarrow e_1\, v & (\text{R-CASE-INL}) \\ \mathsf{case}(\mathsf{inr}_\tau v, e_1, e_2) & \rightsquigarrow e_2\, v & (\text{R-CASE-INR}) \end{array}$$

Further we define, for closed expressions e and e', $e \mapsto e'$ if and only if $e = E\{r\}$ and $e' = E\{e_1\}$ and $r \rightsquigarrow e_1$.

Definition 1. The reflexive and transitive closure of $\mapsto$ is denoted $\mapsto^*$. For $n \geq 0$, we define $e \mapsto^n e'$ iff $e = e_0 \mapsto e_1 \mapsto \cdots e_{n-1} \mapsto e_n = e'$. Further, we write $e \Uparrow$ iff whenever $e \mapsto^* e'$, there exists an e'' such that $e' \mapsto e''$. Finally, we write $e \Downarrow$ iff there exists a v such that $e \mapsto^* v$.

Remark 2. Note that evaluation is only defined for closed expressions and that during evaluation we will only ever substitute closed values for variables.

Theorem 3 (Preservation).
If $e \mapsto e'$ and $\vdash e : \tau$, then $\vdash e' : \tau$.

Theorem 4 (Progress). *If $\vdash e : \tau$, then either e is a value or there exists an e' such that $e \mapsto e'$.*

3 Experimental Equivalence

For closed expressions of base type 1, we define a notion of Kleene approximation and Kleene equivalence as follows.

Definition 5 (Kleene Approximation and Equivalence). For all $e, e' \in \mathrm{Exp}_1$, we define $e \preceq^k e'$ iff $e \mapsto^* * \Rightarrow e' \mapsto^* *$ and $e \approx^k e'$ iff $e \mapsto^* * \iff e' \mapsto^* *$.

For closed expressions we define notions of experimental approximation and experimental equivalence as follows.

Definition 6 (Experimental Approximation and Equivalence). For all $e, e' \in \mathrm{Exp}_\tau$, we define

$$\vdash e \preceq e' : \tau \iff \forall E\{__\tau\} \in \mathrm{ECtx}_1 : E\{e\} \preceq^k E\{e'\}$$
$$\vdash e \approx e' : \tau \iff \forall E\{__\tau\} \in \mathrm{ECtx}_1 : E\{e\} \approx^k E\{e'\}$$

Notation When τ is clear from context we write $e \preceq e'$ for $\vdash e \preceq e' : \tau$ and $e \approx e'$ for $\vdash e \approx e' : \tau$.

We now state some basic properties of experimental equivalence and evaluation.

Lemma 7. *If $\vdash e_1 \approx e_2 : \tau$ then $e_1 \Downarrow$ iff $e_2 \Downarrow$.*

Lemma 8. *For all $e \in Exp_{\tau_1}$ and for all $E\{__{\tau_1}\} \in ECtx_\tau$, $\vdash E\{e\} \approx (\lambda x{:}\tau.E\{x\})\, e : \tau$.*

Lemma 9. *Experimental equivalence, $\approx$, is an equivalence relation.*

Lemma 10.

1. *If $\vdash e \approx (e_1, e_2) : \tau_1 \times \tau_2$ then $e \Downarrow$ iff $e_1 \Downarrow$ and $e_2 \Downarrow$.*
2. *$\vdash (e_1, e_2) \approx (e_1', e_2') : \tau_1 \times \tau_2$ iff $\vdash e_1 \approx e_1' : \tau_1$ and $\vdash e_2 \approx e_2' : \tau_2$.*
3. *If $\vdash e \approx (e_1, e_2) : \tau_1 \times \tau_2$ and $e \Downarrow$, then $\vdash \mathsf{fst} e \approx e_1 : \tau_1$ and $\vdash \mathsf{snd} e \approx e_2 : \tau_2$.*

Lemma 11.

1. *If $\vdash e \approx \mathsf{inl}_{\tau_2} e' : \tau_1 + \tau_2$ or $\vdash e \approx \mathsf{inr}_{\tau_1} e' : \tau_1 + \tau_2$, then $e \Downarrow$ iff $e' \Downarrow$.*
2. *If $\vdash e \approx \mathsf{inl}_{\tau_2} e' : \tau_1 + \tau_2$ and $\vdash e' \approx e'' : \tau_1$, then $\vdash e \approx \mathsf{inl}_{\tau_2} e'' : \tau_1 + \tau_2$. If $\vdash e \approx \mathsf{inr}_{\tau_1} e' : \tau_1 + \tau_2$ and $\vdash e' \approx e'' : \tau_2$, then $\vdash e \approx \mathsf{inr}_{\tau_1} e'' : \tau_1 + \tau_2$.*
3. *If $\vdash e \approx \mathsf{inl}_{\tau_2} e' : \tau_1 + \tau_2$ and $e \Downarrow$, then there exists a v' such that $\vdash e \approx \mathsf{inl}_{\tau_2} v' : \tau_1 + \tau_2$ and $\vdash e' \approx v' : \tau_1$. If $\vdash e \approx \mathsf{inr}_{\tau_1} e' : \tau_1 + \tau_2$ and $e \Downarrow$, then there exists a v' such that $\vdash e \approx \mathsf{inr}_{\tau_1} v' : \tau_1 + \tau_2$ and $\vdash e' \approx v' : \tau_2$.*
4. *$\vdash \mathsf{inl}_{\tau_2} e \approx \mathsf{inl}_{\tau_2} e' : \tau_1 + \tau_2$ iff $\vdash e \approx e' : \tau_1$. $\vdash \mathsf{inr}_{\tau_1} e \approx \mathsf{inr}_{\tau_1} e' : \tau_1 + \tau_2$ iff $\vdash e \approx e' : \tau_2$.*

Lemma 12.

1. *If $\vdash e \approx \mathsf{in} e' : \rho$, then $e \Downarrow$ iff $e' \Downarrow$.*

2. *If* $\vdash e \approx \mathsf{in}e' : \rho$ *and* $\vdash e' \approx e'' : \tau_\rho$, *then* $\vdash e \approx \mathsf{in}e'' : \rho$.
3. $\vdash \mathsf{in}e \approx \mathsf{in}e' : \rho$ *iff* $\vdash e \approx e' : \tau_\rho$.

The next lemma establishes some useful sufficient conditions for establishing experimental approximation.

Lemma 13.

1. *To show* $\vdash v \preceq v' : \tau_1 \rightharpoonup \tau_2$, *it suffices to show*
$$\forall E\{__{\tau_1 \rightharpoonup \tau_2}\, v_1\} \in ECtx_1 : E\{v\} \preceq^k E\{v'\}$$
2. *To show* $\vdash v \preceq v' : \tau_1 \times \tau_2$, *it suffices to show*
$$\forall E\{\mathsf{fst}__{\tau_1 \times \tau_2}\} \in ECtx_1 : E\{v\} \preceq^k E\{v'\}$$
and
$$\forall E\{\mathsf{snd}__{\tau_1 \times \tau_2}\} \in ECtx_1 : E\{v\} \preceq^k E\{v'\}$$
3. *To show* $\vdash v \preceq v' : \tau_1 + \tau_2$, *it suffices to show*
$$\forall E\{\mathsf{case}(__{\tau_1 + \tau_2}, e_1, e_2)\} \in ECtx_1 : E\{v\} \preceq^k E\{v'\}$$
4. *To show* $\vdash v \preceq v' : \rho$, *it suffices to show*
$$\forall E\{\mathsf{out}__\rho\} \in ECtx_1 : E\{v\} \preceq^k E\{v'\}$$

3.1 Compactness of Evaluation

In this section we show that a fix-term is approximated, in the experimental approximation pre-order, by its finite unrollings. Further, we show that to fill a context is a monotone operation with respect to the experimental pre-order and we use this to show that a fix-term is the least upper bound of its finite unrollings. These properties are also referred to as compactness of evaluation. Finally, we show that to fill a context is a continous operation with respect to the approximation pre-order. We shall only be concerned with closed fix-terms, as this suffices for our purposes.

Our development of compactness of evaluation follows the approach of Pitts [21, Section 5] quite closely but there are some technical differences due to the fact that we use a reduction semantics rather than a natural semantics as employed by Pitts. We have chosen this formulation, using cofinal sets, because it fits nicely with our formulation of admissible relations, for which a formulation based on cofinal sets suffices (see Section 4).

Throughout this section we shall consider a particular fixed term $F = \mathsf{fix}f(x{:}\tau_1).e$ satisfying $F \in \mathrm{Exp}_{\tau_1 \rightharpoonup \tau_2}$, and use the following abbreviations:

$$\begin{aligned} F_0 &\stackrel{\text{def}}{=} \mathsf{fix}f^0(x{:}\tau_1).e \stackrel{\text{def}}{=} \mathsf{fix}f(x{:}\tau_1).f\,x \\ F_{n+1} &\stackrel{\text{def}}{=} \mathsf{fix}f^{n+1}(x{:}\tau_1).e \stackrel{\text{def}}{=} \lambda x{:}\tau_1.[F_n/f]e \\ F_\omega &\stackrel{\text{def}}{=} F \end{aligned}$$

Note that we here simply define some abbreviations of expressions already in the language. This is opposed to introducing new labelled expressions and new notions of reduction for labelled expressions as, e.g., done by Gunter [11].

We will only consider contexts involving parameters of type $\tau_1 \rightharpoonup \tau_2$. We write $C\{\mathsf{p}\}$ for such a context whose parameters are included in the list p. Given an k-tuple $\mathbf{n} = (n_1, \ldots, n_k)$ of natural numbers, then we make the following abbreviations.

$$C\{F_{\mathbf{n}}\} \stackrel{\text{def}}{=} C\{F_{n_1}, \ldots, F_{n_k}\}$$
$$C\{F_{\boldsymbol{\omega}}\} \stackrel{\text{def}}{=} C\{F_{\omega}, \ldots, F_{\omega}\}$$

The length of a list of parameter p will be denoted $|\mathsf{p}|$.

Definition 14. For each k, we partially order the set N^k by

$$\mathbf{n} \leq \mathbf{n}' \iff (n_1 \leq n_1' \wedge \cdots \wedge n_k \leq n_k')$$

Definition 15. A subset $I \subseteq N^k$ is said to be cofinal in N^k if and only if, for all $\mathbf{n} \in N^k$, $\exists \mathbf{n'} \in I : \mathbf{n} \leq \mathbf{n'}$. We write $\mathcal{P}_{\text{cof}}(N^k)$ for the set of all such cofinal subsets of N^k.

We introduce the following definitions of sets of value contexts

$$\text{VCtx}_\tau(\Gamma) \stackrel{\text{def}}{=} \{\, C \in \text{Ctx}_\tau(\Gamma) \mid C \text{ is a value or } C \text{ is a parameter} \,\}$$

$$\text{VCtx}_\tau \stackrel{\text{def}}{=} \text{VCtx}_\tau(\emptyset)$$

We use V to range over value contexts. We say that a value context is *proper* if it is not a parameter.

Remark 16. Note that, if $V\{__\tau\} \in \text{VCtx}_{\tau'}$ is a proper value context and $e \in \text{Exp}_\tau$, then $V\{e\}$ is a value. Also, if $V\{__\tau\} \in \text{VCtx}_{\tau'}$ and $v \in \text{Val}_\tau$, then $V\{v\}$ is a value.

Notation We abbreviate $V\{F_{\mathbf{m}}\}$ and $V\{F_{\boldsymbol{\omega}}\}$ analogously to $C\{F_{\mathbf{m}}\}$ and $C\{F_{\boldsymbol{\omega}}\}$.

Definition 17. If $C\{\mathsf{p}\}$ is a context and $V\{\mathsf{p}'\}$ is a value context, then we write $C\{\mathsf{p}\} \Downarrow^F V\{\mathsf{p}'\}$ to mean that for all $I \in \mathcal{P}_{\text{cof}}(N^{|\mathsf{p}|})$

$$\{\, \mathbf{mm}' \mid \mathbf{m} \in I \wedge C\{F_{\mathbf{m}}\} \mapsto^* V\{F_{\mathbf{m}'}\} \,\} \in \mathcal{P}_{\text{cof}}(N^{|\mathsf{p}|+|\mathsf{p}'|})$$

Note that the relation $C\{\mathsf{p}\} \Downarrow^F V\{\mathsf{p}'\}$ is preserved under renaming of the parameters p and, independently, the parameters p'.

Lemma 18.

1. *If $V\{p\}$ is a proper value context, then $V\{p\} \Downarrow^F V\{p\}$.*

2. *If* $E'\{V\}\{p\} \Downarrow^F V''\{p''\}$ *and* $V'\{pp'\}$ *is a value context, then* $E'\{\mathsf{fst}(V, V')\}\{pp'\} \Downarrow^F V''\{p''\}$.
3. *If* $E'\{V\}\{p\} \Downarrow^F V''\{p''\}$ *and* $V'\{pp'\}$ *is a value context, then* $E'\{\mathsf{snd}(V, V')\}\{pp'\} \Downarrow^F V''\{p''\}$.
4. *If* $E'\{V\}\{p\} \Downarrow^F V'\{p'\}$, *then* $E'\{\mathsf{out}(\mathsf{in}V)\}\{p\} \Downarrow^F V'\{p'\}$.
5. *If* $E'\{e_1\, v\}\{p\} \Downarrow^F V\{p'\}$ *and* $e_2 = C_2\{F_\omega\}$ *for some* $C_2\{pp'\}$, *then* $E'\{\mathsf{case}(\mathsf{inl}_{\tau_2} v, e_1, e_2)\}\{pp'\} \Downarrow^F V\{p'\}$
6. *If* $E'\{e_2\, v\}\{p\} \Downarrow^F V\{p'\}$, *and* $e_1 = C_1\{F_\omega\}$ *for some* $C_1\{pp'\}$, *then* $E'\{\mathsf{case}(\mathsf{inr}_{\tau_1} v, e_1, e_2)\}\{pp'\} \Downarrow^F V\{p'\}$

Lemma 19 (Compactness of Evaluation). *For all* $C\{p\} \in Ctx_\tau$, *if* $C\{F_\omega\} \mapsto^* v$, *then there exists a* $V\{p'\} \in VCtx_\tau$ *such that* $v = V\{F_\omega\}$ *and* $C\{p\} \Downarrow^F V\{p'\}$.

The following lemma expresses that the finite unrollings of a fix-term form a chain with respect to the approximation order and that the fix-term itself is an upper bound of this sequence. We shall soon see that it is in fact the *least* upper bound.

Lemma 20. *For all* $i \in N$, $\vdash F_i \preceq F_{i+1} : \tau_1 \rightharpoonup \tau_2$ *and* $\vdash F_i \preceq F_\omega : \tau_1 \rightharpoonup \tau_2$.

To show that a fix-term is the *least* upper bound of its finite unrollings we shall need that the operation of filling a context is a monotone operation with respect to the experimental pre-order (in other words, the experimental pre-order is a pre-congruence). To this end we shall first generalize the experimental pre-order to open expressions in the following way.

Definition 21. An *expression substitution* γ *for a type environment* Γ is a finite map from variables to closed expressions satisfying the following two conditions.

1. $\mathrm{Dom}(\gamma) = \mathrm{Dom}(\Gamma)$.
2. $\forall x \in \mathrm{Dom}(\gamma) : \emptyset \vdash \gamma(x) : \Gamma(x)$.

Definition 22. A *value substitution* γ *for* Γ is an expression substitution for Γ satisfying $\forall x \in \mathrm{Dom}(\Gamma) : \gamma(x) \Downarrow$.

Definition 23. Let γ and γ' be expression substitutions for Γ. Then γ approximates γ', written $\vdash \gamma \preceq \gamma' : \Gamma$, if and only if $\forall x \in \mathrm{Dom}(\Gamma) : \vdash \gamma(x) \preceq \gamma'(x) : \Gamma(x)$. Likewise, we write $\vdash \gamma \approx \gamma' : \Gamma$, if and only if $\forall x \in \mathrm{Dom}(\Gamma) : \vdash \gamma(x) \approx \gamma'(x) : \Gamma(x)$.

Note that this definition also expresses when a *value* substitution γ approximates another value substituition γ' (both for some Γ) as a value substition is just a special expression substitution (we need a notion of expression substition in Section 3.2, which is why we have chosen this formulation).

Definition 24 (Open Experimental Approximation and Equivalence). For all e and e', if $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$, then we define $\Gamma \vdash e \preceq e' : \tau$ if and only if for all value substitutions γ and γ' for Γ such that $\vdash \gamma \preceq \gamma' : \Gamma$, $\vdash \gamma(e) \preceq \gamma'(e') : \tau$. Moreover, we define $\Gamma \vdash e \approx e' : \tau$ if and only if $\Gamma \vdash e \preceq e' : \tau$ and $\Gamma \vdash e' \preceq e : \tau$.

An alternative definition of open experimental approximation would be to say that $\Gamma \vdash e \preceq e' : \tau$ if and only if, for all expression substitutions γ for Γ, $\vdash \gamma(e) \preceq \gamma(e') : \tau$. However, we need the more general definition (specifically it is used in the proof of Lemma 25below).

Lemma 25. *If* $[f : \tau_1 \rightharpoonup \tau_2, x : \tau_1] \vdash e \preceq e' : \tau_2$ *then* $\vdash \mathsf{fix} f(x{:}\tau_1).e \preceq \mathsf{fix} f(x{:}\tau_1).e' : \tau_1 \rightharpoonup \tau_2$.

Lemma 26. *If* $\Gamma, \Gamma' \vdash e \preceq e' : \tau$, $C\{__\tau\} \in Ctx_{\tau'}(\Gamma)$, $\Gamma \vdash C\{e\} : \tau'$, *and* $\Gamma \vdash C\{e'\} : \tau'$, *then* $\Gamma \vdash C\{e\} \preceq C\{e'\} : \tau'$.

The following corollary expresses the monotonicity of contexts with respect to the experimental pre-order — in other words, the experimental pre-order is a pre-congruence. We shall subsequently show that contexts are not only monotone, but also continuous (in an appropriate sense).

Corollary 27 (Context Monotonicity). *If* $\vdash e \preceq e' : \tau_1$ *and* $C\{__{\tau_1}\} \in Ctx_\tau$, *then* $\vdash C\{e\} \preceq C\{e'\} : \tau$.

Lemma 28. *If* $\vdash e_1 \preceq e_1' : \tau_1, \ldots, \vdash e_k \preceq e_k' : \tau_k$ *and* $C\{__1, \ldots, __k\} \in Ctx_\tau$ *with* $__i$ *of type* τ_i, *for all* $1 \leq i \leq k$, *then* $\vdash C\{e_1, \ldots, e_k\} \preceq C\{e_1', \ldots, e_k'\} : \tau$.

Corollary 29. *Experimental equivalence is a congruence relation: if* $\vdash e_1 \approx e_1' : \tau_1 \ldots, \vdash e_k \approx e_k' : \tau_k$ *and* $C\{__1, \ldots, __k\} \in Ctx_\tau$ *with* $__i$ *of type* τ_i, *for all* $1 \leq i \leq k$, *then* $\vdash C\{e_1, \ldots, e_k\} \approx C\{e_1', \ldots, e_k'\} : \tau$.

Before embarking on the theorem from which it follows that a fix-term is the least upper bound of its finite unrollings (Theorem 33), the proof of which will make use of context monotonicity, we shall first make another use of context monotonicity. We shall show that experimental approximation can be used to give an alternative characterization of the usual definition of contextual equivalence — via this alternative characterization, the proof principles for establishing experimental equivalence that are developed in this paper can be used also to establish results of contextual equivalence. This theorem (Theorem 31) is reminiscent of the CIU Theorem of Mason, Smith and Talcott [15].

Definition 30 (Contextual Approximation and Equivalence). If $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$, we define

$$\vdash e \preceq^c e' : \tau \iff \forall C\{__\tau\} \in \mathrm{Ctx}_1 : C\{e\} \preceq^k C\{e'\}$$
$$\vdash e \approx^c e' : \tau \iff \forall C\{__\tau\} \in \mathrm{Ctx}_1 : C\{e\} \approx^k C\{e'\}$$

Theorem 31. *If* $\Gamma \vdash e : \tau$ *and* $\Gamma \vdash e' : \tau$, *then* $\vdash e \preceq^c e' : \tau$ *iff* $\Gamma \vdash e \preceq e' : \tau$.

Corollary 32. *If* $\Gamma \vdash e : \tau$ *and* $\Gamma \vdash e' : \tau$*, then* $\vdash e \approx^c e' : \tau$ *iff* $\Gamma \vdash e \approx e' : \tau$*.*

Theorem 33. *For all* $C\{\mathsf{p}\} \in \mathit{Ctx}_\tau$*, the following three propositions are equivalent.*

1. $\vdash C\{F_{\boldsymbol{\omega}}\} \preceq e : \tau$
2. $\exists I \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{p}|}) : \forall \mathbf{m} \in I : \quad \vdash C\{F_{\mathbf{m}}\} \preceq e : \tau$
3. $\forall I \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{p}|}) : \forall \mathbf{m} \in I : \quad \vdash C\{F_{\mathbf{m}}\} \preceq e : \tau$

Corollary 34 (Context Continuity). *For all* $C\{\mathsf{p}\} \in \mathit{Ctx}_\tau$*,*

$$\vdash C\{F_{\boldsymbol{\omega}}\} \preceq e : \tau \iff \forall n \in N : \quad \vdash C\{F_n, \ldots, F_n\} \preceq e : \tau$$

Let $C\{\mathsf{p}\} = _^{\tau_1 \to \tau_2}$ in Corollary 34. Then the corollary together with Lemma 20 says that F_ω is the least upper bound of the chain of its finite unrollings: By Lemma 20,

$$F_0 \preceq F_1 \preceq F_2 \preceq \cdots$$

is a chain with upper bound F_ω. By Corollary 34, if e is an upper bound of the same chain, then $F_\omega \preceq e$, so F_ω is a least upper bound of the chain:

$$F_\omega = \bigsqcup \{ F_0, F_1, F_2, \ldots \}$$

Furthermore, by Lemma 27,

$$C\{F_0\} \preceq C\{F_1\} \preceq C\{F_2\} \preceq \cdots$$

is again a chain with upper bound $C\{F_\omega\}$, and by Corollary 34, if e is an upper bound of the same chain, then $C\{F_\omega\} \preceq e$, so $C\{F_\omega\}$ is a least upper bound of the chain:

$$C\{F_\omega\} = \bigsqcup \{ C\{F_0\}, C\{F_1\}, C\{F_2\}, \ldots \}$$

In other words, to fill a context is a continuous operation for chains of finite unrollings of fix terms with respect to the approximation order.

As explained by Mason, Smith, and Talcott [15] arbitrary chains of terms do not always have a least upper bound. This leads Mason, Smith, and Talcott to develop a notion of ordering between sets of terms, for which arbitrary chains *do* have a least upper bound, [15, Lemma 4.31]. Here, however, we shall only ever consider chains of the form

$$C\{F_0\} \preceq C\{F_1\} \preceq C\{F_2\} \preceq \cdots$$

for some given closed fix-term F and thus the chains, which we shall consider, will always have a least upper bound. Hence we do not need to develop more complicated notions of approximation à la the set ordering developed by Mason, Smith, and Talcott [15].

3.2 Syntactic Projections

In this section we introduce syntactic projection terms which are the syntactic counterpart of the semantic projection functions known from domain theory. These syntactic projections will be used in the construction of the desired relations in Section 5.

Let π be a variable. For all types τ, we define terms $\Pi_\tau : \tau \rightharpoonup \tau$ (given $\pi : \rho \rightharpoonup \rho$) by induction on τ as follows.

$$\begin{aligned}
\Pi_\rho &\stackrel{\text{def}}{=} \lambda x{:}\rho.\pi\, x \\
\Pi_0 &\stackrel{\text{def}}{=} \lambda x{:}0.x \\
\Pi_1 &\stackrel{\text{def}}{=} \lambda x{:}1.x \\
\Pi_{\tau_1 \times \tau_2} &\stackrel{\text{def}}{=} \lambda x{:}\tau_1 \times \tau_2.(\Pi_{\tau_1}\,(\mathsf{fst}x), \Pi_{\tau_2}\,(\mathsf{snd}x)) \\
\Pi_{\tau_1 + \tau_2} &\stackrel{\text{def}}{=} \lambda x{:}\tau_1 + \tau_2.\mathsf{case}(x, \lambda x{:}\tau_1.\mathsf{inl}_{\tau_2}(\Pi_{\tau_1}\, x), \lambda x{:}\tau_2.\mathsf{inr}_{\tau_1}(\Pi_{\tau_2}\, x)) \\
\Pi_{\tau_1 \rightharpoonup \tau_2} &\stackrel{\text{def}}{=} \lambda f{:}\tau_1 \rightharpoonup \tau_2.\lambda x{:}\tau_1.\Pi_{\tau_2}\,(f\,(\Pi_{\tau_1}\, x))
\end{aligned}$$

Note that π is possibly free in these so defined terms. Further, define terms $\pi^i : \rho \rightharpoonup \rho$, for all $i \geq 0$, by induction on i as follows.

$$\begin{aligned}
\pi^0 &\stackrel{\text{def}}{=} \mathsf{fix}\pi(x{:}\rho).\pi\, x \\
\pi^{i+1} &\stackrel{\text{def}}{=} \lambda x{:}\rho.[\pi^i/\pi]\mathsf{in}(\Pi_{\tau_\rho}\,(\mathsf{out}x))
\end{aligned}$$

and define

$$\pi^\infty \stackrel{\text{def}}{=} \mathsf{fix}\pi(x{:}\rho).\mathsf{in}(\Pi_{\tau_\rho}\,(\mathsf{out}x)) : \rho \rightharpoonup \rho$$

Observe that π^i and also π^∞ are values.

Note that the π^i's are the finite unrollings of the fix-term π^∞ so, as explained in the previous subsection, π^∞ is the least upper bound of the chain of π^i's. The π^∞ term corresponds to the least fixed point $fix(\delta)$ of the continuous function $\delta(e) = iF(e,e)i^{-1}$ in [22, Definition 3.2]. We shall show that π^∞ is experimentally equivalent to the identity function (more precisely, the term $\lambda x{:}\rho.x$); this corresponds to the minimal invariant property in [22, Definition 3.2].

Example 35. Assume $\tau_\rho = 1 + \rho$. Intuitively, our recursive type then corresponds to the type of natural numbers. Then π^∞ is equal to

$$\mathsf{fix}\pi(x{:}\rho).\mathsf{in}((\lambda x{:}1 + \rho.\mathsf{case}(x, \lambda x{:}1.\mathsf{inl}_\rho((\lambda x{:}1.x)\, x), \lambda x{:}\rho.\mathsf{inr}_1((\lambda x{:}\rho.\pi\, x)\, x)))\,(\mathsf{out}x))$$

Intuitively, it is clear that this is equivalent to the identify function.

For all τ and all $i \geq 0$, we define

$$\Pi_\tau^i \stackrel{\text{def}}{=} [\pi^i/\pi]\Pi_\tau : \tau \rightharpoonup \tau$$

Finally, for all τ, we define

$$\Pi_\tau^\infty \stackrel{\text{def}}{=} [\pi^\infty/\pi]\Pi_\tau : \tau \rightharpoonup \tau$$

It is easy to show that the above definitions do indeed define terms, i.e., for all τ, $[\pi : \rho \rightharpoonup \rho] \vdash \Pi_\tau : \tau \rightharpoonup \tau$ and $\vdash \Pi_\tau^\infty : \tau \rightharpoonup \tau$; $\vdash \pi^\infty : \rho \rightharpoonup \rho$; and for all τ, for all $i \geq 0$, $\vdash \Pi_\tau^i : \tau \rightharpoonup \tau$; and for all $i \geq 0$, $\vdash \pi^i : \rho \rightharpoonup \rho$.

We aim to show that π^∞ is operationally equivalent to the identity function $\lambda x{:}\rho.x$. To this end we need a series of simple lemmas which we now state.

Lemma 36. *If* $\vdash e \approx * : 1$ *then*

1. *For all* $i \geq 0$, $\vdash \Pi_1^i\, e \approx * : 1$.
2. $\vdash \Pi_1^\infty\, e \approx * : 1$

Lemma 37. *If* $\vdash e \approx (v_1, v_2) : \tau_1 \times \tau_2$, *then*

1. *For all* $i \geq 0$, $\vdash \Pi^i_{\tau_1\times\tau_2}\, e \approx (\Pi^i_{\tau_1}\, v_1, \Pi^i_{\tau_2}\, v_2) : \tau_1 \times \tau_2$.
2. $\vdash \Pi^\infty_{\tau_1\times\tau_2}\, e \approx (\Pi^\infty_{\tau_1}\, v_1, \Pi^\infty_{\tau_2}\, v_2) : \tau_1 \times \tau_2$.

Lemma 38.

1. *If* $\vdash e \approx \mathsf{inl}_{\tau_2} v : \tau_1 + \tau_2$, *then*
   (a) *For all* $i \geq 0$, $\vdash \Pi^i_{\tau_1+\tau_2}\, e \approx \mathsf{inl}_{\tau_2}(\Pi^i_{\tau_1}\, v) : \tau_1 + \tau_2$.
   (b) $\vdash \Pi^\infty_{\tau_1+\tau_2}\, e \approx \mathsf{inl}_{\tau_2}(\Pi^\infty_{\tau_1}\, v) : \tau_1 + \tau_2$.
2. *If* $\vdash e \approx \mathsf{inr}_{\tau_1} v : \tau_1 + \tau_2$, *then*
   (a) *For all* $i \geq 0$, $\vdash \Pi^i_{\tau_1+\tau_2}\, e \approx \mathsf{inr}_{\tau_1}(\Pi^i_{\tau_2}\, v) : \tau_1 + \tau_2$.
   (b) $\vdash \Pi^\infty_{\tau_1+\tau_2}\, e \approx \mathsf{inr}_{\tau_1}(\Pi^\infty_{\tau_2}\, v) : \tau_1 + \tau_2$.

Lemma 39. *If* $\vdash e \approx v : \tau_1 \rightharpoonup \tau_2$, *then*

1. *For all* $i \geq 0$, $\vdash \Pi^i_{\tau_1\rightharpoonup\tau_2}\, e \approx \lambda x{:}\tau_1.\Pi^i_{\tau_2}\,(v\,(\Pi^i_{\tau_1}\, x)) : \tau_1 \rightharpoonup \tau_2$.
2. $\vdash \Pi^\infty_{\tau_1\rightharpoonup\tau_2}\, e \approx \lambda x{:}\tau_1.\Pi^\infty_{\tau_2}\,(v\,(\Pi^\infty_{\tau_1}\, x)) : \tau_1 \rightharpoonup \tau_2$.

Lemma 40. *If* $\vdash e \approx \mathsf{in} v : \rho$, *then*

1. $\Pi^0_\rho\, e \Uparrow$;
2. *For all* $i \geq 1$, $\vdash \Pi^i_\rho\, e \approx \mathsf{in}(\Pi^{i-1}_{\tau_\rho}\, v) : \rho$;
3. $\vdash \Pi^\infty_\rho\, e \approx \mathsf{in}(\Pi^\infty_{\tau_\rho}\, v) : \rho$;
4. *For all* $i \geq 1$, $\vdash \pi^i\, e \approx \mathsf{in}(\Pi^{i-1}_{\tau_\rho}\, v) : \rho$;
5. $\vdash \pi^\infty\, e \approx \mathsf{in}(\Pi^\infty_{\tau_\rho}\, v) : \rho$.

Lemma 41. *For all* τ *and for all* $i \geq 0$, $\vdash \Pi_\tau^i \preceq \lambda x{:}\tau.x : \tau \rightharpoonup \tau$.

Proof (Sketch) By induction on (i, τ) ordered lexicographically, using the previous list of lemmas. □

We are now in a position to show one half of the operational equivalence of π^∞ and the identity function, namely that π^∞ approximates the identity function.

Lemma 42. $\vdash \pi^\infty \preceq \lambda x{:}\rho.x : \rho \rightharpoonup \rho$

Proof (Sketch) We show by induction on $i \geq 0$ that

$$\forall i \in N : \quad \vdash \pi^i \preceq \lambda x{:}\rho.x : \rho \rightharpoonup \rho \tag{1}$$

The result follows by Corollary 34. □

Next we aim to show the other half of the operational equivalence of π^∞ and the identity function, that is, that the identity function operationally approximates π^∞. We shall employ an idea of Mason, Smith, and Talcott [15].

We first establish the idempotency of Π_τ^∞ and π^∞. We proceed by establishing lemmas for Π_τ^i and π^i and then use compactness of evaluation to get the desired results.

Lemma 43. *For all $i \geq 0$ and for all τ, $\vdash \Pi_\tau^i \preceq \lambda x{:}\tau.\Pi_\tau^\infty\,(\Pi_\tau^\infty\,x) : \tau \rightharpoonup \tau$.*

Lemma 44. *For all $i \geq 0$ and for all τ, $\vdash \lambda x{:}\tau.\Pi_\tau^i\,(\Pi_\tau^i\,x) \preceq \Pi_\tau^\infty : \tau \rightharpoonup \tau$.*

Proof (Sketch) By Corollary 13 it suffices to show, for all $i \geq 0$, for all $v \in \mathrm{Val}_\tau$, and for all $E\{__{\tau \rightharpoonup \tau}\,v\} \in \mathrm{ECtx}_1$,

$$E\{\Pi_\tau^i\,v\} \preceq^k E\{(\lambda x{:}\tau.\Pi_\tau^\infty\,(\Pi_\tau^\infty\,x))\,v\}$$

This can shown by induction on (i, τ) ordered lexicographically. □

Lemma 45. *For all τ, $\vdash \Pi_\tau^\infty \preceq \lambda x{:}\tau.\Pi_\tau^\infty\,(\Pi_\tau^\infty\,x) : \tau \rightharpoonup \tau$.*

Lemma 46. *For all τ, $\vdash \lambda x{:}\tau.\Pi_\tau^\infty\,(\Pi_\tau^\infty\,x) \preceq \Pi_\tau^\infty : \tau \rightharpoonup \tau$.*

Lemma 47. *For all $e \in Exp_\tau$ and for all $E\{__\tau\} \in ECtx_{\tau'}$, $\vdash E\{\Pi_\tau^\infty\,(\Pi_\tau^\infty\,e)\} \approx E\{\Pi_\tau^\infty\,e\} : \tau'$.*

Lemma 48. *For all $e \in Exp_\rho$ and for all $E\{__\rho\} \in ECtx_\tau$, $\vdash E\{\pi^\infty\,(\pi^\infty\,e)\} \approx E\{\pi^\infty\,e\} : \tau$.*

We then define a "compilation" relation for expressions that annotates terms with syntactic projections. The relation $\Gamma \vdash e : \tau \Rightarrow |e|$ is defined by induction on $\Gamma \vdash e : \tau$ by the axioms and inference rules in Figure 2.

Lemma 49. *If $\Gamma \vdash e : \tau \Rightarrow |e|$, then $\Gamma \vdash |e| : \tau$.*

For any $E\{__\tau\} \in \mathrm{ECtx}_{\tau'}$, we define $|E|$ as follows. Clearly, $[z : \tau] \vdash E\{z\} : \tau'$. Thus for some e', $[z : \tau] \vdash E\{z\} : \tau' \Rightarrow e'$. By induction on the derivation there will be one free occurrence of z in e'. We define $|E| \stackrel{\mathrm{def}}{=} [__\tau/z]e'$, and by the remarks given here and Lemma 49, $|E|\{__\tau\} \in \mathrm{ECtx}_{\tau'}$.

Lemma 50. *For all $e \in Exp_\tau(\Gamma)$ and for all expression substitutions γ for Γ, if $\Gamma \vdash e : \tau \Rightarrow |e|$, then $\vdash \Pi_\tau^\infty\,(\gamma|e|) \approx \gamma|e| : \tau$.*

$$\Gamma \vdash x : \tau \Rightarrow \Pi^{\infty}_{\tau}\, x \quad (\Gamma(x) = \tau) \qquad \text{(TR-VAR)}$$

$$\Gamma \vdash * : 1 \Rightarrow \Pi^{\infty}_{1}\, * \qquad \text{(TR-ONE)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \Rightarrow |e_1| \qquad \Gamma \vdash e_2 : \tau_2 \Rightarrow |e_2|}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \Rightarrow \Pi^{\infty}_{\tau_1 \times \tau_2}\,(|e_1|, |e_2|)} \qquad \text{(TR-PROD)}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \Rightarrow |e|}{\Gamma \vdash \mathsf{fst} e : \tau_1 \Rightarrow \mathsf{fst}|e|} \qquad \text{(TR-FST)}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \Rightarrow |e|}{\Gamma \vdash \mathsf{snd} e : \tau_2 \Rightarrow \mathsf{snd}|e|} \qquad \text{(TR-SND)}$$

$$\frac{\Gamma \vdash e : \tau_1 \Rightarrow |e|}{\Gamma \vdash \mathsf{in}|_{\tau_2} e : \tau_1 + \tau_2 \Rightarrow \Pi^{\infty}_{\tau_1 + \tau_2}\,(\mathsf{in}|_{\tau_2}|e|)} \qquad \text{(TR-INL)}$$

$$\frac{\Gamma \vdash e : \tau_2 \Rightarrow |e|}{\Gamma \vdash \mathsf{inr}_{\tau_1} e : \tau_1 + \tau_2 \Rightarrow \Pi^{\infty}_{\tau_1 + \tau_2}\,(\mathsf{inr}_{\tau_1}|e|)} \qquad \text{(TR-INR)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \Rightarrow |e_1| \qquad \Gamma \vdash e_2 : \tau_1 \rightharpoonup \tau \Rightarrow |e_2| \qquad \Gamma \vdash e_3 : \tau_1 \rightharpoonup \tau \Rightarrow |e_3|}{\Gamma \vdash \mathsf{case}(e_1, e_2, e_3) : \tau \Rightarrow \mathsf{case}(|e_1|, |e_2|, |e_3|)} \qquad \text{(TR-CASE)}$$

$$\frac{\Gamma[f : \tau_1 \rightharpoonup \tau_2][x : \tau_1] \vdash e : \tau_2 \Rightarrow |e|}{\Gamma \vdash \mathsf{fix} f(x{:}\tau_1).e : \tau_1 \rightharpoonup \tau_2 \Rightarrow \Pi^{\infty}_{\tau_1 \rightharpoonup \tau_2}\,(\mathsf{fix} f(x{:}\tau_1).|e|)} \quad (f, x \notin \mathrm{Dom}(\Gamma)) \qquad \text{(TR-FIX)}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \rightharpoonup \tau \Rightarrow |e_1| \qquad \Gamma \vdash e_2 : \tau_2 \Rightarrow |e_2|}{\Gamma \vdash e_1\, e_2 : \tau \Rightarrow |e_1|\,|e_2|} \qquad \text{(TR-APP)}$$

$$\frac{\Gamma \vdash e : \rho \Rightarrow |e|}{\Gamma \vdash \mathsf{out} e : \tau_\rho \Rightarrow \mathsf{out}|e|} \qquad \text{(TR-OUT)}$$

$$\frac{\Gamma \vdash e : \tau_\rho \Rightarrow |e|}{\Gamma \vdash \mathsf{in} e : \rho \Rightarrow \Pi^{\infty}_{\rho}\,(\mathsf{in}|e|)} \qquad \text{(TR-IN)}$$

Fig. 2. Definition of $\Gamma \vdash e : \tau \Rightarrow |e|$.

Lemma 51. *For all $e \in Exp_\tau(\Gamma)$ and for all expression substitutions γ, γ' for Γ, if $\vdash \gamma \preceq \gamma' : \Gamma$ and $\Gamma \vdash e : \tau \Rightarrow |e|$, then $\vdash \gamma|e| \preceq \gamma'(e) : \tau$.*

Corollary 52. *If $\emptyset \vdash e : \tau \Rightarrow |e|$, then $\vdash |e| \preceq e : \tau$.*

Corollary 53. *For all $E\{__\tau\} \in ECtx_{\tau'}$ and for all expression substitutions γ for $\Gamma = [z : \tau]$, if $[z : \tau] \vdash E\{z\} \Rightarrow |E\{z\}|$, then $\vdash \gamma|E\{z\}| \preceq \gamma(E\{z\}) : \tau'$.*

Lemma 54. *For all $e \in Exp_\tau$ and for all $E\{__\tau\} \in ECtx_{\tau'}$*

1. *If $[x_1 : \tau_1, \ldots, x_k : \tau_k] \vdash e \Rightarrow |e|$ and $\emptyset \vdash e_1 : \tau_1, \ \ldots, \emptyset \vdash e_k : \tau_k$, then*

$$\vdash |[e_1, \ldots, e_k/x_1, \ldots, x_k]e| \approx [|e_1|, \ldots, |e_k|/x_1, \ldots, x_k]|e| : \tau$$

2. $\vdash |E\{e\}| \approx |E|\{|e|\} : \tau'$.

Lemma 55. *For all τ and for all $v \in Val_\tau$ the following holds.*

1. $|v| \Downarrow$
2. $\Pi_\tau^\infty\, |v| \Downarrow$

Lemma 56. *For all $e \in Exp_\tau$, if $\emptyset \vdash e : \tau \Rightarrow |e|$ and $e \mapsto e'$, then $\vdash |e| \approx |e'| : \tau$, where $\emptyset \vdash e' : \tau \Rightarrow |e'|$*

Lemma 57. $\vdash \lambda x{:}\rho.x \preceq \pi^\infty : \rho \rightharpoonup \rho$

Proof By Corollary 13 it suffices to show, for all $E\{__{\rho \rightharpoonup \rho}\, (\mathsf{in}v)\} \in \mathrm{ECtx}_1$,

$$E\{\lambda x{:}\rho.x\, (\mathsf{in}v)\} \preceq^k E\{\pi^\infty\, (\mathsf{in}v)\}$$

Let $E\{__{\rho \rightharpoonup \rho}\, (\mathsf{in}v)\} \in \mathrm{ECtx}_1$ be arbitrary. By Lemma 8, it then suffices to show,

$$E\{\mathsf{in}v\} \preceq^k E\{\pi^\infty\, (\mathsf{in}v)\}$$

By Corollary 52 it then suffices to show,

$$E\{\mathsf{in}v\} \preceq^k E\{\pi^\infty\, |\mathsf{in}v|\}$$

Since clearly $\vdash \pi^\infty \approx \Pi_\rho^\infty : \rho \rightharpoonup \rho$, by Lemma 50 it then suffices to show,

$$E\{\mathsf{in}v\} \preceq^k E\{|\mathsf{in}v|\} \tag{2}$$

Suppose that

$$E\{\mathsf{in}v\} \preceq^k |E\{\mathsf{in}v\}| \tag{3}$$

holds. Assuming this, we can reason as follows

$$\begin{array}{rll} E\{\mathsf{in}v\} \mapsto^* * \Rightarrow |E\{\mathsf{in}v\}| \mapsto^* * & & \text{by assumption (3)} \\ \Rightarrow |E|\{|\mathsf{in}v|\} \mapsto^* * & & \text{by Lemma 54, item 2} \\ \Rightarrow E\{|\mathsf{in}v|\} \mapsto^* * & & \text{by Corollary 53} \end{array}$$

which gives (2) as required.

Thus we are left with showing (3). Clearly this follows from showing, for all closed expressions $e \in \mathrm{Exp}_1$,

$$e \mapsto^* * \Rightarrow |e| \mapsto^* *$$

We show this by induction on the length m of the compuation of $e \mapsto^* *$.

Basis $(m = 0)$*:* Then $e = *$, whence $|e| = \Pi_1^\infty * \mapsto^* *$, as required.

Inductive Step: Assume $e \mapsto e' \mapsto^m *$. Then by induction we get that $|e'| \mapsto^* *$. By Lemma 56, also $|e| \mapsto^* *$, as required. □

We are now in a position to establish the following theorem, which we refer to as the syntactic minimal invariant property by analogy to the domain-theoretic work of Pitts [22].

Theorem 58 (Syntactic Minimal Invariance). $\vdash \pi^\infty \approx \lambda x{:}\rho.x : \rho \rightharpoonup \rho$

4 Relations

In this and the following section we shall show how to construct a relational interpretation of types over an operational semantics. We shall end up by showing "The Fundamental Theorem of Logical Relations" which states that the relational interpretation of types is sound in the sense that well-typed terms are related to themselves by the relation associated to their type. The constructed relations can be seen to provide a notion of equality of terms — and will therefore be called "relations for equality" — and can in fact be used to reason about contextual equivalence [2]. However, we will not pursue that direction in this extended abstract. Instead we shall view these two sections as providing the necessary understanding for constructing a relational interpretation, which we then use to show the correctness of cps transformation in Section 6.

In this section we define a universe of relations over equivalence classes of closed expressions, with respect to operational equivalence. Further, we define a notion of admissibility for relations. This corresponds to the notion of admissibility (also known as inclusiveness or completeness) used in domain theory, and is also here used as a condition on relations, which, loosely speaking, allows one to show that a fix-term is in a relation by showing that its approximants are in the relation. Next we show that admissible relations equipped with the obvious ordering form a complete lattice, define relational constructors corresponding to the type constructors of the language, and show that these constructors preserve admissibility.

Throughout this section we will let $n \in N$ be an arbitrary but fixed natural number, that is, we will consider n-ary relations for a fixed, but arbitrary $n \in N$. We will use the same abbreviations for terms involving fix and for contexts as in Section 3.1. For any set A and natural number m we write A^m for the m-ary cartesian product of A. For any set A and any equivalence relation $\equiv$ on A, we write $A/\!\equiv$ for the set of equivalence classes of A with respect to $\equiv$. To

simplify notation we denote each equivalence class by one of its representatives. Moreover, we will simply use $\approx$ for the operational equivalence relation at type τ (i.e., $(e, e') \in \approx \iff \vdash e \approx e' : \tau$) when τ is clear from context.

Definition 59. For all τ, we define a universe of n-ary relations Rel_τ as follows.

$$Rel_\tau \stackrel{\text{def}}{=} \mathcal{P}\left((\text{Exp}_\tau /\approx)^n\right)$$

We use R to range over Rel_τ.

Definition 60. A relation $R \in Rel_\tau$ is *admissible* if and only if it satisfies both of the following two conditions.

Strictness: $(e_1, \ldots, e_n) \in R$ if and only if $((\forall i \in 1..n : e_i \Uparrow) \vee (\forall i \in 1..n : \exists v_i : e_i \mapsto^* v_i \wedge (v_1, \ldots, v_n) \in R))$

Completeness: For all $i \in 1..n$ and for all $C_i\{\mathsf{p}\} \in \text{Ctx}_\tau$ with all parameters in p of type $\tau_1 \rightharpoonup \tau_2$ and for all $F^i_\omega = \mathsf{fix} f(x{:}\tau_1).e_i \in \text{Exp}_{\tau_1 \rightharpoonup \tau_2}$

$$\left(\forall \mathbf{m} \in I \in \mathcal{P}_{\text{cof}}(N^{|\mathsf{p}|}) : (C_1\{F^1_{\mathbf{m}}\}, \ldots, C_n\{F^n_{\mathbf{m}}\}) \in R\right) \Rightarrow$$

$$\left((C_1\{F^1_\omega\}, \ldots, C_n\{F^n_\omega\}) \in R\right)$$

Recall that $C\{\mathsf{p}\}$ means that all of the parameters of C are *included* in p, that is, in the completeness condition the contexts C_i are not required to all have the same number of parameters.

Definition 61. For all τ, we define a universe of admissible n-ary relations $Radm_\tau$ as follows.

$$Radm_\tau \stackrel{\text{def}}{=} \{\, R \in Rel_\tau \mid R \text{ is admissible} \,\}$$

We also use R to range over $Radm_\tau$.

We now define a series of relational constructors corresponding to the syntactic type constructors. For each of these constructors it is easy to verify that the definition does not depend on the choice of representative of an operational equivalence class.

Definition 62.

$$R_0 \stackrel{\text{def}}{=} \{\, (e_1, \ldots, e_n) \in (\text{Exp}_0 /\approx)^n \mid \forall i \in 1..n : e_i \Uparrow \,\}$$

Definition 63.

$$R_1 \stackrel{\text{def}}{=} \{\, (e_1, \ldots, e_n) \in (\text{Exp}_1 /\approx)^n \mid (\forall i \in 1..n : e_i \Uparrow) \vee (\forall i \in 1..n : e_i \mapsto^* *) \,\}$$

Definition 64. For all $R_1 \in Rel_{\tau_1}$ and $R_2 \in Rel_{\tau_2}$,

$$\begin{aligned} R_1 \times R_2 \stackrel{\text{def}}{=} \{\, & (e_1, \ldots, e_n) \in (\text{Exp}_{\tau_1 \times \tau_2} /\approx)^n \mid \\ & (\forall i \in 1..n : e_i \Uparrow) \vee \\ & (\forall i \in 1..n : \exists v_i, v'_i : \vdash e_i \approx (v_i, v'_i) : \tau_1 \times \tau_2 \\ & \qquad \wedge (v_1, \ldots, v_n) \in R_1 \wedge (v'_1, \ldots, v'_n) \in R_2) \,\} \end{aligned}$$

Definition 65. For all $R_1 \in Rel_{\tau_1}$ and $R_2 \in Rel_{\tau_2}$,

$$\begin{aligned} R_1 + R_2 \stackrel{\text{def}}{=} \{\, (e_1, \ldots, e_n) \in (\mathrm{Exp}_{\tau_1+\tau_2}/\approx)^n \mid \\ (\forall i \in 1..n : e_i \Uparrow) \vee \\ (\forall i \in 1..n : \exists v_i : \vdash e_i \approx \mathsf{inl}_{\tau_2} v_i : \tau_1 + \tau_2 \wedge (v_1, \ldots, v_n) \in R_1) \\ (\forall i \in 1..n : \exists v_i : \vdash e_i \approx \mathsf{inr}_{\tau_1} v_i : \tau_1 + \tau_2 \wedge (v_1, \ldots, v_n) \in R_2) \,\} \end{aligned}$$

Definition 66. For all $R_1 \in Rel_{\tau_1}$ and $R_2 \in Rel_{\tau_2}$,

$$\begin{aligned} R_1 \rightharpoonup R_2 \stackrel{\text{def}}{=} \{\, (e_1, \ldots, e_n) \in (\mathrm{Exp}_{\tau_1 \rightharpoonup \tau_2}/\approx)^n \mid \\ (\forall i \in 1..n : e_i \Uparrow) \vee \\ (\forall i \in 1..n : \exists v_i : \vdash e_i \approx v_i : \tau_1 \rightharpoonup \tau_2 \wedge ((e'_1, \ldots, e'_n) \in R_1 \Rightarrow \\ (v_1\, e'_1, \ldots, v_n\, e'_n) \in R_2)) \,\} \end{aligned}$$

Lemma 67. *For all τ, $(Radm_\tau, \subseteq)$ is a complete lattice.*

The relational constructors all preserve admissiblity.

Lemma 68.

1. *For all $R_1 \in Radm_{\tau_1}$ and all $R_2 \in Radm_{\tau_2}$, $R_1 \times R_2 \in Radm_{\tau_1 \times \tau_2}$;*
2. *For all $R_1 \in Radm_{\tau_1}$ and all $R_2 \in Radm_{\tau_2}$, $R_1 + R_2 \in Radm_{\tau_1+\tau_2}$;*
3. *For all $R_1 \in Rel_{\tau_1}$ and all $R_2 \in Radm_{\tau_2}$, $R_1 \rightharpoonup R_2 \in Radm_{\tau_1 \rightharpoonup \tau_2}$;*
4. *$R_1 \in Radm_1$;*
5. *$R_0 \in Radm_0$.*

5 Relational Interpretation

In this section we give a relational interpretation of the types of $\mathcal{L}$, that is, an assignment of admissible relations to each type. To interpret the different type constructors we, of course, make use of the corresponding relational constructors defined in the previous section. Our construction follows along the lines of Pitts [22].

Definition 69. For all τ, define $[\![\tau]\!] : Radm_\rho \to Radm_\tau$ by induction on τ as follows.

$$\begin{aligned} [\![0]\!]R &= R_0 \\ [\![1]\!]R &= R_1 \\ [\![\rho]\!]R &= R \\ [\![\tau_1 \times \tau_2]\!]R &= [\![\tau_1]\!]R \times [\![\tau_2]\!]R \\ [\![\tau_1 + \tau_2]\!]R &= [\![\tau_1]\!]R + [\![\tau_2]\!]R \\ [\![\tau_1 \rightharpoonup \tau_2]\!]R &= [\![\tau_1]\!]R \rightharpoonup [\![\tau_2]\!]R \end{aligned}$$

Note that the operation $[\![\tau]\!]$ is well-defined by induction on τ and Lemma 68.

Definition 70. Define $\Phi : Radm_\rho \to Radm_\rho$ by

$$\begin{aligned} \Phi(R) \stackrel{\text{def}}{=} \{\, (e_1, \ldots, e_n) \in (\mathrm{Exp}_\rho/\approx)^n \mid \\ (\forall i \in 1..n : e_i \Uparrow) \vee (\forall i \in 1..n : \exists v_i : \vdash e_i \approx \mathsf{in} v_i : \rho \wedge \\ (v_1, \ldots, v_n) \in [\![\tau_\rho]\!]R) \,\} \end{aligned}$$

Lemma 71. *Φ is well-defined.*

Lemma 72. *$(Radm^{op} \times Radm)$ ordered componentwise is a complete lattice.*

Definition 73. For all τ, define $[\![\tau]\!]' : (Radm_\rho^{op} \times Radm_\rho) \to Radm_\tau$ by induction on τ as follows.

$$\begin{array}{ll}
[\![0]\!]'(R^-, R^+) & = R_0 \\
[\![1]\!]'(R^-, R^+) & = R_1 \\
[\![\rho]\!]'(R^-, R^+) & = R^+ \\
[\![\tau_1 \times \tau_2]\!]'(R^-, R^+) & = [\![\tau_1]\!]'(R^-, R^+) \times [\![\tau_2]\!]'(R^-, R^+) \\
[\![\tau_1 + \tau_2]\!]'(R^-, R^+) & = [\![\tau_1]\!]'(R^-, R^+) + [\![\tau_2]\!]'(R^-, R^+) \\
[\![\tau_1 \rightharpoonup \tau_2]\!]'(R^-, R^+) & = [\![\tau_1]\!]'(R^+, R^-) \rightharpoonup [\![\tau_2]\!]'(R^-, R^+)
\end{array}$$

Note that the operation $[\![\tau]\!]'$ is well-defined by induction on τ and Lemma 68.

Definition 74. Define $\Psi : (Radm_\rho^{op} \times Radm_\rho) \to Radm_\rho$ by

$$\begin{array}{l}
\Psi(R^-, R^+) \stackrel{\text{def}}{=} \{\, (e_1, \ldots, e_n) \in (\mathrm{Exp}_\rho \,/\!\approx)^n \;\mid \\
\qquad (\forall i \in 1..n : e_i \Uparrow) \vee (\forall i \in 1..n : \exists v_i : \vdash e_i \approx \mathsf{in} v_i : \rho \wedge \\
\qquad\qquad (v_1, \ldots, v_n) \in [\![\tau_\rho]\!]'(R^-, R^+)) \,\}
\end{array}$$

Lemma 75. *Ψ is well-defined.*

Definition 76. Define $\Psi^\S : (Radm_\rho^{op} \times Radm_\rho) \to (Radm_\rho^{op} \times Radm_\rho)$ as follows.

$$\Psi^\S(R^-, R^+) = (\Psi(R^+, R^-), \Psi(R^-, R^+))$$

Lemma 77. *$\Psi^\S$ is monotone.*

Definition 78. By Lemma 77 and 72 and Tarski's fixed point theorem, $\Psi^\S$ has a least fixed point $\mathrm{lfp}(\Psi^\S)$. Define $(\Delta^-, \Delta^+) \stackrel{\text{def}}{=} \mathrm{lfp}(\Psi^\S)$.

Lemma 79. *(Δ^-, Δ^+) satisfies the following properties*

1. *$\Delta^-, \Delta^+ \in Radm_\rho$*
2. *$\Delta^- = \Psi(\Delta^+, \Delta^-)$*
3. *$\Delta^+ = \Psi(\Delta^-, \Delta^+)$*
4. *for all $(R^-, R^+) \in (Radm_\rho^{op} \times Radm_\rho)$, if $\Psi^\S(R^-, R^+) \sqsubseteq (R^-, R^+)$ then $R^- \subseteq \Delta^-$ and $R^+ \supseteq \Delta^+$*
5. *$\Delta^+ \subseteq \Delta^-$*

To simplify notation, we write $e : R \subset R'$ for

$$\forall (e_1, \ldots, e_n) \in R : (e\, e_1, \ldots, e\, e_n) \in R'.$$

Note that this notation does not depend on the chosen equivalence class representative, so the notation is indeed well-defined.

Lemma 80. *For all $i \in N$ and for all τ,*

$$\Pi^i_\tau : [\![\tau]\!]'(\Delta^+, \Delta^-) \subset [\![\tau]\!]'(\Delta^-, \Delta^+)$$

Proof (Sketch) By induction on the pair (i, τ) ordered lexicographically. □

Lemma 81. *For all $i \in N$, $\pi^i : \Delta^- \subset \Delta^+$.*

Proof (Sketch) By induction on i. □

Lemma 82.

$$\pi^\infty : \Delta^- \subset \Delta^+$$

It follows from Lemma 82 and Theorem 58 that $\Delta^- \subseteq \Delta^+$, and hence by Lemma 79 that $\Delta^- = \Delta^+$. This justifies the following definition.

Definition 83. For all τ define $R_\tau \stackrel{\text{def}}{=} [\![\tau]\!]\Delta^+$.

This completes the construction of relations R_τ for all τ.

We now aim to show "The Fundamental Theorem of Logical Relations" which states that the relational interpretation of types is sound in the sense that well-typed terms are related to themselves by the relation associated to their type. To this end we first extend the interpretation of types as relations to type environments.

Definition 84. For all type environments Γ,

$$\begin{aligned} R_\Gamma \stackrel{\text{def}}{=} \{\, (\gamma_1, \ldots, \gamma_n) \mid \\ (\forall i \in 1..n : \gamma_i \text{ is an expression substitution for } \Gamma) \wedge \\ (\forall x \in \mathrm{Dom}(\Gamma) : (\gamma_1(x), \ldots, \gamma_n(x)) \in R_{\Gamma(x)}) \,\} \end{aligned}$$

Theorem 85. *If $\Gamma \vdash e : \tau$ and $(\gamma_1, \ldots, \gamma_n) \in R_\Gamma$, then $(\gamma_1(e), \ldots, \gamma_n(e)) \in R_\tau$.*

Proof (Sketch) By induction on $\Gamma \vdash e : \tau$. In the case for T-FIX, by admissibility of the relations R_τ, for all τ, it suffices to show, for all $i \in N$,

$$(\mathsf{fix} f^i(x{:}\tau_1).\gamma_1(e), \ldots, \mathsf{fix} f^i(x{:}\tau_1).\gamma_n(e)) \in R_{\tau_1 \to \tau_2}$$

but this is easy to show by an inner induction on i using the outer induction hypothesis. □

6 Correctness of CPS Transformation

We define the cps transformation as a relation between a "source" and a "target" language. The source language, $\mathcal{L}^{\rho}$, is just the language $\mathcal{L}$ defined earlier. The target language, $\mathcal{L}^{\rho^*}$, is the variant of $\mathcal{L}$ obtained by replacing the single recursive type ρ by another recursive type ρ^* obtained from ρ by a transformation on types similar to that given by Meyer and Wand [16].

We let Type^{ρ} denote the set of type expressions of $\mathcal{L}^{\rho}$, that is $\mathrm{Type}^{\rho} = \mathrm{Type}$. The set of target type expressions, denoted Type^{ρ^*}, is defined exactly as Type, but with ρ^* for ρ.

Below we define two type translations from Type^{ρ} to Type^{ρ^*}, one for computations, $\overline{\tau}$, and one for values, τ^* and extend the one for values to type environments. Note that the case $(\rho)^* = \rho^*$ is not recursive; it reads: "the value type translation of the source type ρ is the target type ρ^*."

Computations
$$\overline{\tau} = (\tau^* \to 1) \to 1$$

Values
$$\begin{aligned} 0^* &= 0 \\ 1^* &= 1 \\ (\rho)^* &= \rho^* \\ (\tau_1 \times \tau_2)^* &= {\tau_1}^* \times {\tau_2}^* \\ (\tau_1 + \tau_2)^* &= {\tau_1}^* + {\tau_2}^* \\ (\tau_1 \rightharpoonup \tau_2)^* &= {\tau_1}^* \rightharpoonup \overline{\tau_2} \end{aligned}$$

Type Environments
$$\Gamma^*(x) = (\Gamma(x))^* \quad (x \in \mathrm{Dom}(\Gamma))$$

In the target language $\mathcal{L}^{\rho^*}$ we take the recursive type ρ^* to be isomorphic to ${\tau_\rho}^*$.

We shall use the same notation for both the source and target language, but we must take care to remember to which language we are referring. Of course, all the results obtained in previous sections for $\mathcal{L}$ hold analogously for both the source and target language (for the source it is obvious as it is equal to $\mathcal{L}$, for the target, just replace ρ with ρ^* and τ_ρ with ${\tau_\rho}^*$ everywhere) and we will freely refer to these results to reason about both the source and the target language. When we need to distinguish between sets of expressions of the source and the target language, we shall use the notation developed for $\mathcal{L}$ but use a superscript ρ for the source language and a superscript ρ^* for the target language. For example, $\mathrm{Exp}^{\rho}_{\tau}$ denotes the set of closed expressions of type τ of the source language, whereas $\mathrm{Exp}^{\rho^*}_{\tau}$ denotes the set of closed expressions of type τ of the target language. Moreover, we will abuse notation and write $e \approx^k e'$, for $e \in \mathrm{Exp}^{\rho}_1$ and $e' \in \mathrm{Exp}^{\rho^*}_1$, to mean that e evaluates to $*$ in $\mathcal{L}^{\rho}$ if and only if e' evaluates to $*$ in $\mathcal{L}^{\rho^*}$.

The translation relations $\Gamma \vdash v : \tau \rightsquigarrow_v v'$ for values and $\Gamma \vdash e : \tau \rightsquigarrow_c e'$ for computations are inductively defined by the rules in Figure 3.

Lemma 86.

$$\Gamma \vdash x : \tau \rightsquigarrow_v x \quad (\Gamma(x) = \tau) \tag{CPS-VAR}$$

$$\Gamma \vdash * : 1 \rightsquigarrow_v * \tag{CPS-ONE}$$

$$\frac{\Gamma[f : \tau_1 \rightharpoonup \tau_2][x : \tau_1] \vdash e : \tau_2 \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{fix} f(x{:}\tau_1).e : \tau_1 \rightharpoonup \tau_2 \rightsquigarrow_v \mathsf{fix} f(x{:}{\tau_1}^*).e'} \quad (f, x \notin \mathrm{Dom}(\Gamma)) \tag{CPS-FIX}$$

$$\frac{\Gamma \vdash v : \tau \rightsquigarrow_v v'}{\Gamma \vdash v : \tau \rightsquigarrow_c \lambda k{:}\tau^* \rightharpoonup 1.k\, v'} \tag{CPS-VAL}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightsquigarrow_c e_1' \qquad \Gamma \vdash e_2 : \tau_2 \rightsquigarrow_c e_2'}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \rightsquigarrow_c \lambda k{:}(\tau_1 \times \tau_2)^* \rightharpoonup 1.e_1'\,(\lambda x_1{:}{\tau_1}^*.e_2'\,(\lambda x_2{:}{\tau_2}^*.k\,(x_1, x_2)))} \tag{CPS-PROD}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{fst} e : \tau_1 \rightsquigarrow_c \lambda k{:}{\tau_1}^* \rightharpoonup 1.e'\,(\lambda x{:}{\tau_1}^*.k\,(\mathsf{fst} x))} \tag{CPS-FST}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{snd} e : \tau_2 \rightsquigarrow_c \lambda k{:}{\tau_2}^* \rightharpoonup 1.e'\,(\lambda x{:}{\tau_2}^*.k\,(\mathsf{snd} x))} \tag{CPS-SND}$$

$$\frac{\Gamma \vdash e : \tau_1 \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{in}|_{\tau_2} e : \tau_1 + \tau_2 \rightsquigarrow_c \lambda k{:}(\tau_1 + \tau_2)^* \rightharpoonup 1.e'\,(\lambda x{:}{\tau_1}^*.k\,(\mathsf{in}|_{{\tau_2}^*} x))} \tag{CPS-INL}$$

$$\frac{\Gamma \vdash e : \tau_2 \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{inr}_{\tau_1} e : \tau_1 + \tau_2 \rightsquigarrow_c \lambda k{:}(\tau_1 + \tau_2)^* \rightharpoonup 1.e'\,(\lambda x{:}{\tau_2}^*.k\,(\mathsf{inr}_{{\tau_1}^*} x))} \tag{CPS-INR}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightsquigarrow_c e_1' \qquad \Gamma \vdash e_2 : \tau_1 \rightharpoonup \tau \rightsquigarrow_c e_2' \qquad \Gamma \vdash e_3 : \tau_1 \rightharpoonup \tau \rightsquigarrow_c e_3'}{\Gamma \vdash \mathsf{case}(e_1, e_2, e_3) : \tau \rightsquigarrow_c \lambda k{:}\tau^* \rightharpoonup 1.e_1'\,(\lambda x{:}{\tau_1}^*.\mathsf{case}(x, e_2'\,x\,k, e_3'\,x\,k))} \tag{CPS-CASE}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \rightharpoonup \tau \rightsquigarrow_c e_1' \qquad \Gamma \vdash e_2 : \tau_2 \rightsquigarrow_c e_2'}{\Gamma \vdash e_1\, e_2 : \tau \rightsquigarrow_c \lambda k{:}\tau^* \rightharpoonup 1.e_1'\,(\lambda x_1{:}(\tau_2 \rightharpoonup \tau)^*.e_2'\,(\lambda x_2{:}{\tau_2}^*.x_1\, x_2\, k))} \tag{CPS-APP}$$

$$\frac{\Gamma \vdash e : \rho \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{out} e : \tau_\rho \rightsquigarrow_c \lambda k{:}{\tau_\rho}^* \rightharpoonup 1.e'\,(\lambda x{:}\rho^*.k\,(\mathsf{out} x))} \tag{CPS-OUT}$$

$$\frac{\Gamma \vdash e : \tau_\rho \rightsquigarrow_c e'}{\Gamma \vdash \mathsf{in} e : \rho \rightsquigarrow_c \lambda k{:}\rho^* \rightharpoonup 1.e'\,(\lambda x{:}{\tau_\rho}^*.k\,(\mathsf{in} x))} \tag{CPS-IN}$$

Fig. 3. CPS Transformation

1. $\Gamma \vdash e : \tau \rightsquigarrow_c e'$ *for some* e' *iff* $\Gamma \vdash e : \tau$.
2. *If* $\Gamma \vdash v : \tau \rightsquigarrow_v v'$, *then* $\Gamma^* \vdash v' : \tau^*$.
3. *If* $\Gamma \vdash e : \tau \rightsquigarrow_c e'$, *then* $\Gamma^* \vdash e' : \overline{\tau}$.

We extend the notion of experimental equivalence to evaluation contexts as follows.

Definition 87. For all $E\{__\tau\}, E'\{__\tau\} \in \mathrm{ECtx}_{\tau'}$, we define

$$\vdash E\{__\tau\} \approx E'\{__\tau\} : \tau' \iff (\forall e, e' \in \mathrm{Exp}_\tau : \vdash e \approx e' : \tau \Rightarrow \vdash E\{e\} \approx E'\{e'\} : \tau').$$

As in Section 4 we denote equivalence classes by one of their representatives.

Theorem 88. *There exists a* Type^ρ*-indexed family of relations*

$$\Delta^c_\tau \subseteq Exp^\rho_\tau \,/{\approx} \times Exp^{\rho^*}_{\tau^*} \,/{\approx}$$

$$\Delta^v_\tau \subseteq Val^\rho_\tau \,/{\approx} \times Val^{\rho^*}_{\tau^*} \,/{\approx}$$

$$\Delta^k_\tau \subseteq ECtx^\rho_\tau \,/{\approx} \times Val^{\rho^*}_{\tau^* \to 1} \,/{\approx}$$

satisfying

$$e \;\Delta^c_\tau\; e' \iff E\{__\tau\} \;\Delta^k_\tau\; v' \Rightarrow E\{e\} \approx^k e'\, v'$$

$$\begin{array}{rcl}
v \;\Delta^v_1\; v' & \iff & v = *, v' = * \\
v \;\Delta^v_0\; v' & & \textit{never} \\
v \;\Delta^v_\rho\; v' & \iff & \vdash v \approx \mathsf{in} v_1 : \rho,\ \vdash v' \approx \mathsf{in} v'_1 : \rho^*,\ v_1 \;\Delta^v_{\tau_\rho}\; v'_1 \\
v \;\Delta^v_{\tau_1 \times \tau_2}\; v' & \iff & \vdash v \approx (v_1, v_2) : \tau_1 \times \tau_2,\ \vdash v' \approx (v'_1, v'_2) : {\tau_1}^* \times {\tau_2}^*, \\
 & & v_1 \;\Delta^v_{\tau_1}\; v'_1,\ v_2 \;\Delta^v_{\tau_2}\; v'_2 \\
v \;\Delta^v_{\tau_1 + \tau_2}\; v' & \iff & \big(\vdash v \approx \mathsf{inl}_{\tau_2} v_1 : \tau_1 + \tau_2,\ \vdash v' \approx \mathsf{inl}_{\tau_2{}^*} v'_1 : {\tau_1}^* + {\tau_2}^*,\ v_1 \;\Delta^v_{\tau_1}\; v'_1\big) \\
 & \vee & \big(\vdash v \approx \mathsf{inr}_{\tau_1} v_1 : \tau_1 + \tau_2,\ \vdash v' \approx \mathsf{inr}_{\tau_1{}^*} v'_1 : {\tau_1}^* + {\tau_2}^*,\ v_1 \;\Delta^v_{\tau_2}\; v'_1\big) \\
v \;\Delta^v_{\tau_1 \to \tau_2}\; v' & \iff & v_1 \;\Delta^v_{\tau_1}\; v'_1 \Rightarrow v\, v_1 \;\Delta^c_{\tau_2}\; v'\, v'_1
\end{array}$$

$$E\{__\tau\} \;\Delta^k_\tau\; v' \iff v_1 \;\Delta^v_\tau\; v'_1 \Rightarrow E\{v_1\} \approx^k v'\, v'_1,$$

and

$$\big(\forall i \in N : \mathsf{fix} f^i(x{:}\tau_1).e \;\Delta^v_{\tau_1 \to \tau_2}\; \mathsf{fix} f^i(x{:}{\tau_1}^*).e'\big) \Rightarrow \mathsf{fix} f(x{:}\tau_1).e \;\Delta^v_{\tau_1 \to \tau_2}\; \mathsf{fix} f(x{:}{\tau_1}^*).e'.$$

(Note that the conditions satisfied by the relations are all independent of the choice of equivalence class representative and are thus well-defined conditions.)

The proof of this theorem will be postponed until Section 6.1 Now we shall first see how to use the relations that exists by the theorem to prove the correctness of the cps transformation.

Definition 89. Let Δ^c_τ, Δ^v_τ, and Δ^k_τ be relations as in Theorem 88. We then define a source type environment indexed family of relations, Δ^v_Γ, relating source value substitutions for Γ modulo experimental equivalence[1] to target value substitutions for Γ^* modulo experimental equivalence as follows:

$$\gamma\ \Delta^v_\Gamma\ \gamma' \iff \forall x \in \mathrm{Dom}(\Gamma) : \gamma(x)\ \Delta^v_{\Gamma(x)}\ \gamma'(x).$$

Theorem 90.

1. *If* $\Gamma \vdash v : \tau \rightsquigarrow_v v'$ *and* $\gamma\ \Delta^v_\Gamma\ \gamma'$, *then* $\gamma(v)\ \Delta^v_\tau\ \gamma'(v')$.
2. *If* $\Gamma \vdash e : \tau \rightsquigarrow_c e'$ *and* $\gamma\ \Delta^v_\Gamma\ \gamma'$, *then* $\gamma(e)\ \Delta^c_\tau\ \gamma'(e')$.

Corollary 91. *If* $\vdash e : 1 \rightsquigarrow_c e'$, *then* $e' \approx^k e\ (\lambda x{:}1.x)$.

6.1 Construction of Relations for CPS Correctness

In this section we prove Theorem 88. This amounts to constructing relations satisfying the conditions in Theorem 88. The idea is to proceed as in Sections 4 and 5 but, of course, with a different universe of relations and with different relational constructors.

We define a source type indexed family of universes of relations as follows.

Definition 92. For all source types τ, we define a universe of relations

$$Rel_\tau \stackrel{\mathrm{def}}{=} \mathcal{P}\left((\mathrm{Exp}^\rho_\tau/{\approx}) \times (\mathrm{Exp}^{\rho^*}_{\tau^*}/{\approx})\right).$$

We use R to range over Rel_τ.

Notation When $I \in \mathcal{P}_{\mathrm{cof}}(N^{k+l})$ we write "$\mathbf{mm'}$" for "$(i_1, \ldots, i_k, i_{k+1}, \ldots, i_{k+l}) \in I$ and $\mathbf{m} = (i_1, \ldots, i_k)$ and $\mathbf{m'} = (i_{k+1}, \ldots, i_{k+l})$."

As in Section 4, we shall also use a notion admissibility.

Definition 93. A relation $R \in Rel_\tau$ is *admissible* if and only if it satisfies both of the following conditions.

Strictness: $(e, e') \in R$ iff $(e \Uparrow \wedge e' \Uparrow) \vee (\exists v, v' : e \mapsto^* v \wedge e' \mapsto^* v' \wedge (v, v) \in R)$.
Completeness: For all $C\{\mathsf{p}\} \in \mathrm{Ctx}^\rho_\tau$ with all parameters in p of type $\tau_1 \rightharpoonup \tau_2$, for all $C'\{\mathsf{q}\} \in \mathrm{Ctx}^{\rho^*}_{\tau^*}$ with all parameters in q of type $(\tau_1 \rightharpoonup \tau_2)^*$, for all $F_\omega = \mathsf{fix} f(x{:}\tau_1).e \in \mathrm{Exp}^\rho_{\tau_1 \rightharpoonup \tau_2}$, for all $F'_\omega = \mathsf{fix} f(x{:}{\tau_1}^*).e' \in \mathrm{Exp}^{\rho^*}_{(\tau_1 \rightharpoonup \tau_2)^*}$,

$$\left(\forall \mathbf{mm'} \in I \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{p}|+|\mathsf{q}|}) : (C\{F_{\mathbf{m}}\}, C'\{F'_{\mathbf{m'}}\}) \in R\right) \Rightarrow$$

$$((C\{F_\omega\}, C'\{F'_\omega\}) \in R)).$$

[1] Recall the definition of experimental equivalence for substitutions, Definition 23.

Definition 94. For all source types τ, we define a universe of admissible relations $Radm_\tau$ as follows.

$$Radm_\tau \stackrel{\text{def}}{=} \{\, R \in Rel_\tau \mid R \text{ is admissible} \,\}$$

We also use R to range over $Radm_\tau$.

We now define a series of relational type constructors. In each case, one has to check that the definitions we give are independent of the chosen equivalence class representative; this is straightforward in all cases (it is just like in Section 4).

Definition 95.

$$R_0 \stackrel{\text{def}}{=} \{\, (e, e') \in (\mathrm{Exp}_0^\rho / \approx) \times (\mathrm{Exp}_0^{\rho^*} / \approx) \mid e \Uparrow \wedge e' \Uparrow \,\}$$

Definition 96.

$$R_1 \stackrel{\text{def}}{=} \{\, (e, e') \in (\mathrm{Exp}_1^\rho / \approx) \times (\mathrm{Exp}_1^{\rho^*} / \approx) \mid (e \Uparrow \wedge e' \Uparrow) \vee (e \mapsto^* * \wedge\, e' \mapsto^* *) \,\}$$

Definition 97. For all $R_1 \in Rel_{\tau_1}$ and $R_2 \in Rel_{\tau_2}$,

$$\begin{array}{rl} R_1 \times R_2 \stackrel{\text{def}}{=} \{\, & (e, e') \in (\mathrm{Exp}^\rho_{\tau_1 \times \tau_2} / \approx) \times (\mathrm{Exp}^{\rho^*}_{\tau_1 \times \tau_2} / \approx) \mid \\ & (e \Uparrow \wedge e' \Uparrow) \vee \\ & (\exists v_1, v_2, v_1', v_2' : \vdash e \approx (v_1, v_2) : \tau_1 \times \tau_2 \wedge \\ & \quad \vdash e' \approx (v_1', v_2') : {\tau_1}^* \times {\tau_2}^* \wedge \\ & \quad (v_1, v_1') \in R_1 \wedge (v_2, v_2') \in R_2) \,\} \end{array}$$

Definition 98. For all $R_1 \in Rel_{\tau_1}$ and $R_2 \in Rel_{\tau_2}$,

$$\begin{array}{rl} R_1 + R_2 \stackrel{\text{def}}{=} \{\, & (e, e') \in (\mathrm{Exp}^\rho_{\tau_1 + \tau_2} / \approx) \times (\mathrm{Exp}^{\rho^*}_{\tau_1 + \tau_2} / \approx) \mid \\ & (e \Uparrow \wedge e' \Uparrow) \vee \\ & (\exists v, v' : \vdash e \approx \mathsf{inl}_{\tau_2} v : \tau_1 + \tau_2 \wedge \vdash e' \approx \mathsf{inl}_{\tau_2{}^*} v' : {\tau_1}^* + {\tau_2}^* \wedge \\ & \quad (v, v') \in R_1) \vee \\ & (\exists v, v' : \vdash e \approx \mathsf{inr}_{\tau_1} v : \tau_1 + \tau_2 \wedge \vdash e' \approx \mathsf{inr}_{\tau_1{}^*} v' : {\tau_1}^* + {\tau_2}^* \wedge \\ & \quad (v, v') \in R_2) \,\} \end{array}$$

The following relational constructors will be used in the definition of the relational constructor for function types.

Definition 99. For all $R \in Rel_\tau$,

$$\begin{array}{rl} \Delta^k_\tau\ (R) \stackrel{\text{def}}{=} \{\, & (E\{__\tau\}, v') \in (\mathrm{ECtx}^\rho_\tau / \approx) \times (\mathrm{Val}^{\rho^*}_{\tau^* \to 1} / \approx) \mid \\ & \forall (e, e') \in R : E\{e\} \approx^k v'\, e' \,\} \end{array}$$

Definition 100. For all $R \in Rel_\tau$,

$$\begin{array}{rl} \Delta^c_\tau\ (R) \stackrel{\text{def}}{=} \{\, & (e, e') \in (\mathrm{Exp}^\rho_\tau / \approx) \times (\mathrm{Exp}^{\rho^*}_{\overline{\tau}} / \approx) \mid \\ & (e \Uparrow \wedge e' \Uparrow) \vee \\ & (\exists v, v' : \vdash e \approx v : \tau \wedge \vdash e' \approx v' : \overline{\tau} \wedge \\ & \qquad \forall (E_0\{__\tau\}, v_0') \in \Delta^k_\tau\ (R) : E_0\{v\} \approx^k v'\, v_0') \,\} \end{array}$$

Definition 101. For all $R_1 \in Rel_{\tau_1}$ and $R_2 \in Rel_{\tau_2}$,

$$\begin{array}{rl} R_1 \rightharpoonup R_2 \stackrel{\text{def}}{=} \{\, (e,e') \in (\mathrm{Exp}^{\rho}_{\tau_1 \rightharpoonup \tau_2} / \approx) \times (\mathrm{Exp}^{\rho^*}_{{\tau_1}^* \rightharpoonup \overline{\tau_2}} / \approx) \mid \\ \quad (e \Uparrow \wedge e' \Uparrow) \vee \\ \quad (\exists v, v' : \vdash e \approx v : \tau_1 \rightharpoonup \tau_2 \wedge \vdash e' \approx v' : {\tau_1}^* \rightharpoonup \overline{\tau_2} \wedge \\ \qquad \forall (e_1, e_1') \in R_1 : (v\, e_1, v'\, e_1') \in \Delta^c_{\tau_2}\,(R_2)\,\} \end{array}$$

Note that $R_1 \rightharpoonup R_2$ is antimonotone in R_1 and monotone in R_2.

Analogously to Section 4, one can now show that $(Radm_\tau, \subseteq)$ is a complete lattice, for all source types τ; R_0 and R_1 are admissible; and $\times$ and $+$ both preserve admissibility. We now show that $\rightharpoonup$ preserve admissibility:

Lemma 102. *For all $R_1 \in Rel_{\tau_1}$ and all $R_2 \in Radm_{\tau_2}$, $R_1 \rightharpoonup R_2 \in Radm_{\tau_1 \rightharpoonup \tau_2}$.*

Proof The strictness condition is straightforward. For completeness, assume

$$\forall \mathbf{m}\mathbf{m}' \in I \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{p}|+|\mathsf{q}|}) : (C\{F_{\mathbf{m}}\}, C'\{F'_{\mathbf{m}'}\}) \in R. \tag{4}$$

One can show that there are two cases to consider.

Case I: $C\{F_\omega\} \Uparrow \wedge C'\{F'_\omega\} \Uparrow$ Easy.

Case II: $C\{F_\omega\} \mapsto^* v$ and $C'\{F'_\omega\} \mapsto^* v'$. By two applications of Lemma 19, there exist $V\{\mathsf{p_1}\}$ and $V'\{\mathsf{q_1}\}$ such that

$$\begin{array}{ll} v = V\{F_\omega\} & C\{\mathsf{p}\} \Downarrow^F V\{\mathsf{p_1}\} \\ v' = V'\{F'_\omega\} & C'\{\mathsf{q}\} \Downarrow^{F'} V'\{\mathsf{q_1}\} \end{array}$$

so

$$I'_1 \stackrel{\text{def}}{=} \{\, \mathbf{m}\mathbf{m}' \mid \mathbf{m}\mathbf{n} \in I \wedge C\{F_{\mathbf{m}}\} \mapsto^* V\{F_{\mathbf{m}'}\}\,\} \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{p}|+|\mathsf{p_1}|})$$

and

$$I'_2 \stackrel{\text{def}}{=} \{\, \mathbf{n}\mathbf{n}' \mid \mathbf{m}\mathbf{n} \in I \wedge C'\{F_{\mathbf{n}}\} \mapsto^* V'\{F'_{\mathbf{n}'}\}\,\} \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{q}|+|\mathsf{q_1}|}).$$

Thus

$$I'' \stackrel{\text{def}}{=} \{\, \mathbf{m}'\mathbf{n}' \mid \mathbf{m}\mathbf{m}' \in I'_1 \wedge \mathbf{n}\mathbf{n}' \in I'_2 \wedge \mathbf{m}\mathbf{n} \in I\,\}$$

is cofinal, i.e., $I'' \in \mathcal{P}_{\mathrm{cof}}(N^{|\mathsf{p_1}|+|\mathsf{q_1}|})$. By (4), and definition of I'',

$$\forall \mathbf{m}'\mathbf{n}' \in I'' : (V\{F_{\mathbf{m}'}\}, V'\{F'_{\mathbf{n}'}\}) \in R_1 \rightharpoonup R_2.$$

Hence, by definition of $\rightharpoonup$,

$$\forall \mathbf{m}'\mathbf{n}' \in I'' : \forall (e,e') \in R_1 : (V\{F_{\mathbf{m}'}\}\, e, V'\{F'_{\mathbf{n}'}\}\, e') \in \Delta^c_{\tau_2}\,(R_2).$$

Let $\mathbf{m}'\mathbf{n}' \in I''$ and $(e,e') \in R_1$ be arbitrary. By definition of $\Delta^c_{\tau_2}\,(R_2)$ we then have that

$$\forall (E_0\{_{\tau_2}\}, v'_0) \in \Delta^k_{\tau_2}\,(R_2) : E_0\{V\{F_{\mathbf{m}'}\}\, e\} \approx^k V'\{F'_{\mathbf{n}'}\}\, e'\, v'_0. \tag{5}$$

We are to show that

$$\forall (E_0\{_{\tau_2}\}, v'_0) \in \Delta^k_{\tau_2}\,(R_2) : E_0\{V\{F_\omega\}\, e\} \approx^k V'\{F'_\omega\}\, e'\, v'_0. \tag{6}$$

Let $(E_0\{__{\tau_2}\}, v_0') \in \Delta^k_{\tau_2}\ (R_2)$ be arbitrary. Suppose $E_0\{V\{F_\omega\}\, e\} \mapsto^* *$. Let $C_{11}\{\mathsf{p_1}\} = E_0\{V\{\mathsf{p_1}\}\, e\}$. Then by Lemma 19,

$$C_{11}\{\mathsf{p_1}\} \Downarrow^F *.$$

Hence

$$I_{11} \stackrel{\text{def}}{=} \{\, \mathbf{m}'\mathbf{n}' \mid \mathbf{m}'\mathbf{n}' \in I \wedge C_{11}\{F_{\mathbf{m}'}\} \mapsto^* *\,\}$$

is cofinal, thus non-empty. So there exists $\mathbf{m}'\mathbf{n}' \in I$ such that $C_{11}\{F_{\mathbf{m}'}\} \mapsto^* *$, i.e. $E_0\{V\{F_{\mathbf{m}'}\}\, e\} \mapsto^* *$. Hence, by (4), $V'\{F'_{\mathbf{n}'}\}\, e'\, v_0' \mapsto^* *$, from which $V'\{F'_\omega\}\, e'\, v_0' \mapsto^* *$ follows by Lemma 20. The other direction is similar, completing the proof of (6). Thus we conclude that $(C\{F_\omega\}, C'\{F'_\omega\}) \in R_1 \rightharpoonup R_2$, as required, since (e, e') and $(E_0\{__{\tau_2}\}, v_0')$ were arbitray. □

For all source types $\tau \in \mathrm{Type}^\rho$ we define an interpretation $[\![\tau]\!]'$ exactly as in Definition 73.

Definition 103. Define $\Psi : (Radm_\rho^{op} \times Radm_\rho) \to Radm_\rho$ by

$$\begin{aligned}\Psi(R^-, R^+) \stackrel{\text{def}}{=} \{\, &(e, e') \in (\mathrm{Exp}^\rho_\rho /\!\approx) \times (\mathrm{Exp}^{\rho^*}_{\rho^*} /\!\approx) \mid \\ &(e \Uparrow \wedge e' \Uparrow) \vee \\ &(\exists v, v' :\ \vdash e \approx \mathsf{in}v : \rho \wedge\ \vdash e' \approx \mathsf{in}v' : \rho^* \wedge (v, v') \in [\![\tau_\rho]\!]'(R^-, R^+))\,\}\end{aligned}$$

We define $\Psi^{\S} : (Radm_\rho^{op} \times Radm_\rho) \to (Radm_\rho^{op} \times Radm_\rho)$ and as in Section 5 we get that $\Psi^{\S}$ is well-defined and monotone, so that we can define (Δ^-, Δ^+) as the least fixed point of $\Psi^{\S}$. Moreover, Lemma 79 holds also now.

We write $(e, e') : R \subset R'$ for $\forall (e_1, e_1') \in R : (e\, e_1, e'\, e_1') \in R'$.

Lemma 104. *For all $i \in N$, for all $\tau \in \mathrm{Type}^\rho$,*

$$(\Pi^{\rho,i}_\tau, \Pi^{\rho^*,i}_{\tau^*}) : [\![\tau]\!]'(\Delta^+, \Delta^-) \subset [\![\tau]\!]'(\Delta^-, \Delta^+).$$

Lemma 105. *For all $i \in N$, $(\pi^{\rho,i}, \pi^{\rho^*,i}) : \Delta^- \subset \Delta^+$.*

Lemma 106.

$$(\pi^{\rho,\infty}, \pi^{\rho^*,\infty}) : \Delta^- \subset \Delta^+$$

As in Section 5, it now follows that $\Delta^- = \Delta^+$ and we can define $\Delta \stackrel{\text{def}}{=} \Delta^+$.

Definition 107. For all source types $\tau \in \mathrm{Type}^\rho$, we define

$$\Delta^e_\tau \stackrel{\text{def}}{=} [\![\tau]\!]'(\Delta, \Delta).$$

Definition 108. For all source types $\tau \in \mathrm{Type}^\rho$, we define

$$\begin{aligned}\Delta^v_\tau &\stackrel{\text{def}}{=} \{\, (e, e') \in \Delta^e_\tau \mid e \Downarrow \wedge e' \Downarrow \,\} \\ \Delta^k_\tau &\stackrel{\text{def}}{=} \{\, (E\{__\tau\}, v') \in (\mathrm{ECtx}^\rho_\tau /\!\approx) \times (\mathrm{Val}^{\rho^*}_{\tau^* \to 1} /\!\approx) \mid (v_1, v_1') \in \Delta^v_\tau \Rightarrow E\{v\} \approx^k v'\, v \,\} \\ \Delta^c_\tau &\stackrel{\text{def}}{=} \{\, (e, e') \in (\mathrm{Exp}^\rho_\tau /\!\approx) \times (\mathrm{Exp}^{\rho^*}_{\tau^*} /\!\approx) \mid (E\{__\tau\}, v') \in \Delta^k_\tau \Rightarrow E\{e\} \approx^k e'\, v' \,\}\end{aligned}$$

Lemma 109. *The above defined relations satisfy the conditions in Theorem 88.*

Proof All the conditions, except the one for $\Delta^v_{\tau_1 \rightharpoonup \tau_2}$ and the completeness condition, are obvious from the above definitions. By definition of $\Delta^v_{\tau_1 \rightharpoonup \tau_2}$, we have that

$$v \; \Delta^v_{\tau_1 \rightharpoonup \tau_2} \; v' \iff \left(e_1 \; \Delta^e_{\tau_1} \; e_1' \Rightarrow v\, e_1 \; \Delta^c_{\tau_2} \; v'\, e_1'\right),$$

but it is easy to check, using the definition of $\Delta^c_{\tau_2}$, that

$$(e_1 \; \Delta^e_{\tau_1} \; e_1' \Rightarrow v\, e_1 \; \Delta^c_{\tau_2} \; v'\, e_1') \iff (v_1 \; \Delta^v_{\tau_1} \; v_1' \Rightarrow v\, v_1 \; \Delta^c_{\tau_2} \; v'\, v_1')$$

which gives the required. The completeness condition for $\Delta^v_{\tau_1 \rightharpoonup \tau_2}$ follows by admissibility of $\Delta^e_{\tau_1 \rightharpoonup \tau_2}$ (using $I = \{\, (i,i) \mid i \in N \,\}$ a the cofinal set) and the facts that $\mathsf{fix} f(x{:}\tau_1).e \Downarrow$ and $\mathsf{fix} f(x{:}\tau_1{}^*).e' \Downarrow$. □

This completes the construction of relations for CPS correctness.

7 Related Work

Much of our work derives from Pitts's work on relational properties of domains [22]. Pitts's innovation was to observe that relational interpretations can be constructed based only on the minimal invariant property of a domain model. Since the minimal invariant condition can be stated entirely in terms of the language itself, it is possible to carry out the construction of relational interpretations entirely at the level of the syntax, taking expressions modulo a suitable notion of operational equivalence.

The method of establishing the minimal invariant property given here is based on the work of Mason, Smith, and Talcott [15]. (See also Abadi, *et. al.* [1].) Specifically, we define a family of typed finitary projection operations corresponding to the projections in a domain model of the language, and prove that the "infinitary" projection operation on the recursive type is the identity up to operational equivalence. Mason, Smith, and Talcott work with an untyped language that includes primitive "run-time type checking" operations such as testing whether or not a value represents a function. (Lassen considers a similar language for which a minimal invariant property is established [14].) The account given here generalizes their work to the case of a typed language with a recursive type. Their run-time type checking primitives emerge as compositions of recursive unrolling and simple case analyses derived from the description of their untyped value space as a recursive type equation.

Several proofs of correctness of the cps transformation have been given in the literature. Reynolds [25] gives a proof for an untyped language that relies on the construction of a system of relations over a domain model of the language. The construction is based directly on the inverse limit construction of the domain model itself. Meyer and Wand [16] give a proof for the simply-typed λ-calculus without recursive functions or recursive types. Their argument also relies on a relational interpretation of types over a model of the language.

8 Conclusion

We have outlined a method for constructing relational interpretations of recursive types in an operational setting. The key result is the syntactic minimal invariant property up to a suitable notion of operational equivalence. With this in hand we may define relational interpretations of types over operational equivalence classes of closed terms. Using this construction we give a relational proof of correctness of cps conversion, generalizing Reynolds's proof to the typed setting.

The choice of experimental equivalence as our notion of operational equivalence is entirely pragmatic — it facilitated a proof of the minimal invariant property. It turns out that for the language considered here experimental equivalence coincides with contextual equivalence (in the sense of Morris [20] and Plotkin [24]) and with bisimulation equivalence (in the sense of Howe [13] and Pitts [21]). So the choice of relation is a matter of technique, rather than of semantics. It would be interesting to see whether the characterization of operational equivalence as a bisimulation might lead to a more concise or insightful proof of minimal invariance.

One unfortunate consequence of the use of experimental equivalence is that our proof of correctness for cps conversion does not easily extend to control operators such as `call/cc` [3,12]. The reason is that we rely on a "uniformity" property of the evaluation relation which states that evaluation steps are parametric in the evaluation context — if $E\{e\} \mapsto E\{e'\}$, then $E'\{e\} \mapsto E'\{e'\}$. This fails in the presence of `call/cc`, and we were unable to find a simple way to avoid the reliance on this condition. It is also unclear, at present, whether our proof can be extended to a language with mutable storage (*e.g.*, ML reference types). Both of these extensions are important directions for further research.

The treatment of cps conversion given here invites generalization to an arbitrary syntactically-definable monad for the language. Filinski's dissertation [5] is a first step towards a general theory of representation of computational effects. Filinski's work suggests that one could give a fairly general correctness proof along the lines suggested here for a wide variety of definable effects.

Finally, the relational interpretation of types given in Section 5 may be used to provide a "logical" characterization of operational equivalence. In his development of a theory of program equivalence for a lazy language with infinite streams [21] Pitts uses a characterization of operational equivalence at stream types that corresponds to the "logical" equivalence determined by the definition of streams as a recursive type. It would be interesting to work out a general account of logical equivalence and to re-consider some of Pitts's examples using it.

Acknowledgements

We are grateful to John Mitchell and Andy Pitts for many useful discussions and for their suggestions on this paper. We also thank Martín Abadi and Furio Honsell for their helpful comments. The work described here was carried out

in part at the Isaac Newton Institute for Mathematical Sciences of Cambridge University in the autumn of 1995. We are grateful to the Newton Institute and the organizers of the program on Semantics of Computation for their support.

Robert Harper is supported by the National Science Foundation under Grant No. CCR-95-2674. Lars Birkedal is supported by the Danish National Research Council.

References

1. Martín Abadi, Luca Cardelli, Benjamin Pierce, and Gordon Plotkin. Dynamic typing in a statically typed language. *ACM Transactions on Programming Languages and Systems*, 13(2):237–268, April 1991.
2. Lars Birkedal and Robert Harper. Relational interpretations of recursive types in an operational setting. Unpublished manuscript.
3. William Clinger and Jonathan Rees. Revised^4 report on the algorithmic language Scheme. *LISP Pointers*, IV(3):1–55, July-Sep. 1991.
4. Bruce Duba, Robert Harper, and David MacQueen. Typing first-class continuations in ML. In *Eighteenth ACM Symposium on Principles of Programming Languages*, January 1991.
5. Andrzej Filinski. *Controlling Effects.* CMU–CS–96–119, School of Computer Science, Carnegie Mellon University, May 1996.
6. Michael J. Fischer. Lambda-calculus schemata. *LISP and Symbolic Computation*, 6(3/4):259–288, November 1993.
7. Peter Freyd. Algebraically complete categories. In A. Carboni, M. C. Pedicchio, and G. Rosolini, editors, *Category Theory. Proceedings, Como 1990*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer-Verlag, 1990.
8. Peter Freyd. Recursive types reduced to inductive types. In *Proceedings of the fifth IEEE Conference on Logic in Computer Science*, pages 498–507, 1990.
9. Peter Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P.T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science. Proceedings of the LMS Symposium, Durham 1991*, volume 177 of *London Mathematical Society Lecture Note Series*, pages 95–106. Cambridge University Press, 1991.
10. Jean-Yves Girard. *Interprétation Fonctionnelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieure.* PhD thesis, Université Paris VII, 1972.
11. Carl A. Gunter. *Semantics of Programming Languages. Structures and Techniques.* MIT Press, 1992.
12. Robert Harper, Bruce Duba, and David MacQueen. Typing first-class continuations in ML. *Journal of Functional Programming*, 3(4):465–484, October 1993.
13. D. J. Howe. Equality in lazy computation systems. In *4th Annual Symposium on Logic in Computer Science*, pages 198–203. IEEE Computer Society Press, Washington, 1989.
14. Søren Lassen. Relational reasoning about contexts. Unpublished manuscript.
15. Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. From operational semantics to domain theory. *Information and Computation*, 1995. To Appear.
16. Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda calculi (summary). In Rohit Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer-Verlag, 1985.

17. Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.
18. John C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B, Formal Models and Semantics*, chapter 8, pages 366–458. Elsevier Science Publishers B.V., 1990.
19. John C. Mitchell. *Foundations for Programming Languages.* Foundations of Computing. MIT Press, 1996.
20. James H. Morris. *Lambda Calculus Models of Programming Languages.* PhD thesis, MIT, 1968.
21. Andrew M. Pitts. Operationally-based theories of program equivalence. In *In Proc. of Summer School on Semantics and Logics of Computation. ESPRIT CLiCS-II.* University of Cambridge. Isaac Newton Institute for Mathematical Sciences., September 1995.
22. Andrew M. Pitts. Relational properties of domains. *Information and Computation*, 127(2):66–90, June 1996.
23. Gordon Plotkin. Call-by-name, call-by-value, and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.
24. Gordon Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–257, 1977.
25. John C. Reynolds. On the relation between direct and continuation semantics. In J. Loeckx, editor, *Proceedings of the Second Colloquium on Automata, Languages and Programming, Saarbrücken*, volume 174 of *Lecture Notes in Computer Science*, pages 141–156. Springer-Verlag, 1974.