September 1991

# Fully Abstract Translations Between Functional Languages

Jon G. Riecke
*University of Pennsylvania*



University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-64.

# Fully Abstract Translations Between Functional Languages


## Abstract

We examine the problem of finding fully abstract translations between programming languages, *i.e.*, translations that preserve code equivalence and nonequivalence. We present three examples of fully abstract translations: one from call-by-value to lazy PCF, one from call-by name to call-by-value PCF, and one from lazy to call-by-value PCF. The translations yield upper and lower bounds on decision procedures for proving equivalences of code. We finally define a notion of "functional translation" that captures the essence of the proofs of full abstraction, and show that some languages *cannot* be translated into others.



## Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-64.

# Fully Abstract Translations Between Functional Languages

MS-CIS-91-64
LOGIC & COMPUTATION 38

Jon G. Riecke

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

September 1991

# Fully Abstract Translations between Functional Languages

Jon G. Riecke*

September 16, 1991

**Abstract**

We examine the problem of finding fully abstract translations between programming languages, *i.e.*, translations that preserve code equivalence and nonequivalence. We present three examples of fully abstract translations: one from call-by-value to lazy PCF, one from call-by-name to call-by-value PCF, and one from lazy to call-by-value PCF. The translations yield upper and lower bounds on decision procedures for proving equivalences of code. We finally define a notion of "functional translation" that captures the essence of the proofs of full abstraction, and show that some languages *cannot* be translated into others.

## 1 Introduction

There are many ways to compare the expressive power of programming languages. For instance, for two strongly-typed languages A and B, we might say that language B is more expressive if it can type-check more expressions. Another criterion might be the constructs provided by the programming languages: language A is more expressive than language B if language A can define all of the operators of language B. (This idea of "definable operators" is explored in [7].) This paper explores a third criterion, related to the idea of definable operators: whether a language can be *translated* into another. Here we will be interested in transforming whole programs instead of focusing on a handful of operators.

In general, a **translation** is syntactically-defined, meaning-preserving map from a source language to a target language. A compiler is a familiar example of a translation. A compiler is syntactically-driven, generating target code based on the parse tree of the source code, and compiled code (when interpreted) produces precisely the same results as the source code (when interpreted). This latter property, which captures the notion of compiler correctness, is crucial, since otherwise a "compiler" could be any program that generates code in the target language.

It is useful to formalize this correctness criterion. First we pick a set of **observations**, which are the observable outcomes of computation. For example, the set $\mathcal{O} = \{\text{"evaluates to } n\text{"} : n \in \mathbf{N}\}$ is a natural notion of observing the computation of arithmetic expressions. We will call a translation adequate if it preserves observations [11]:

*Author's current address: Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. The bulk of this work was done at the MIT Laboratory for Computer Science. The author was supported by an NSF Graduate Fellowship, NSF Grant Nos. 8511190-DCR and 8819761-CCR, and ONR grant Nos. N00014-83-K-012 and N00014-88-K-0557.

**Definition 1.1** Suppose the observations of language $L_1$ and $L_2$ are $\mathcal{O}$. A translation $M \mapsto \widetilde{M}$ from $L_1$ to $L_2$ is **adequate** if $M$ yields an observation in $\mathcal{O}$ iff $\widetilde{M}$ yields the same observation.

Adequacy is a minimal connection between source and translated code. Most reasonable translations are adequate.

There are other properties which may hold for a given translation, *e.g.*, the translation may be time- or space-bounded. Another *semantic* criterion requires that a translation preserve equivalences of arbitrary pieces of code (*e.g.*, code in which functions or procedures have not yet been declared). Two pieces of code are said to be equivalent iff they cannot be distinguished when placed into any program context. More specifically,

**Definition 1.2** A term $M$ **observationally approximates** a term $N$ with respect to observations $\mathcal{O}$ and language $L$ (written $M \sqsubseteq_L^{\mathcal{O}} N$) if, for any $L$-context $C[\cdot]$, $C[M]$ yields an observation implies that $C[N]$ yields the same observation. Two terms $M$ and $N$ are **observationally congruent** (written $M \equiv_L^{\mathcal{O}} N$) if both $M \sqsubseteq_L^{\mathcal{O}} N$ and $N \sqsubseteq_L^{\mathcal{O}} M$.

Observational approximation and congruence are important ideas in programming. For example, observational congruence captures the notion of correct optimizations: replacing $M$ by a faster but observationally congruent term $N$ will not change the final answer of the program. A translation will be called fully abstract if it preserves observational approximations (*cf.* [12, 22, 38]):

**Definition 1.3** Let $\mathcal{O}$ be the observations of $L_1$ and $L_2$. A translation $P \mapsto \widetilde{P}$ from $L_1$ to $L_2$ is **inequationally fully abstract** if

$$M \sqsubseteq_{L_1}^{\mathcal{O}} N \iff \widetilde{M} \sqsubseteq_{L_2}^{\mathcal{O}} \widetilde{N}.$$

Likewise, a translation $P \mapsto \widetilde{P}$ is **equationally fully abstract** if $M \equiv_{L_1}^{\mathcal{O}} N \iff \widetilde{M} \equiv_{L_2}^{\mathcal{O}} \widetilde{N}$.

Fully abstract translations are important for a number of reasons. First, fully abstract translations can be used to reduce questions about code equivalence or nonequivalence in one language to another. For example, if there is an effective means of proving equivalences (observational congruences) in language B and there is an effective, fully abstract translation from language A to language B, then there is an effective proof procedure for observational congruences in language A: first translate terms and then reason about them. Moreover, if the translation is time-bounded, we may be able to deduce lower and upper bounds on decision procedures for proving equivalences. Second, the concept of fully abstract translations yields a notion of expressiveness: language A is "no more expressive" than B if there is a fully abstract translation from A to B. This idea is not new; Mitchell [13, 15] uses the idea of compositional, fully abstract translations to compare languages. Others have examined similar ideas. Felleisen's notion of expressiveness [7] based on "definable operators" is a restricted version of fully abstract translations (where some of the operators of a language are not translated). More recently, Shapiro [31] uses a definition of homomorphic translation to derive a theory of expressiveness of concurrent languages.

This paper explores fully abstract translations between functional languages. To keep the study focused, we restrict attention to finding translations between various versions of the simply-typed, functional language PCF. A full definition of the syntax of PCF appears in Section 2, along with the operational and denotational semantics of three particular versions of the language, call-by-name PCF [9, 22, 30], call-by-value PCF [9, 25, 33, 34], and lazy PCF [5, 6, 9].

Section 3 begins with a description of an adequate translation from call-by-value to lazy PCF. It is then shown that the translation is *not* fully abstract. Section 3 then repairs the translation using syntactically-definable retractions, and proves that the translation is fully abstract. Sections 4 and 5 define other fully abstract translations from call-by-name to call-by-value PCF, and from lazy to call-by-value PCF. These translations also rely upon definable retractions, and the proofs of full abstraction use the same basic technique as the call-by-value to lazy case. Section 6 discusses some complexity-theoretic corollaries to the full abstraction theorems.

Other effective, fully abstract translations based on gödelnumberings of terms can be given. Section 7 defines the notion of a **functional translation** that eliminates such gödelnumbering translations from consideration. We then show that lazy and call-by-value PCF cannot be translated into call-by-name PCF via a functional translation. This is evidence that the notion of a functional translation leads to a nontrivial expressiveness theory. Section 8 concludes with a discussion of some open problems.

# 2 PCF and its Interpretation

This section briefly reviews the operational and denotational semantics of call-by-name, call-by-value, and lazy PCF. The reader familiar with these languages may care to skim this section and refer to it when necessary.

## 2.1 Syntax of PCF

PCF (Programming language for Computable Functions) is a simply-typed language which has some simple constructs for computing with integers. The set of PCF-terms is the least set closed under the formation rules of Table 1.[1] A term is a **value** if it is a numeral or $\lambda$-abstraction; values are denoted by $V$.

Most of the operations of PCF behave in the intuitive way. For instance, the sequential conditional $(\mathsf{cond}\ M\ N\ P)$ reduces its first argument, returning the value of the second if the first halts at 0 and the value of the third if the first halts at a numeral greater than 0. The parallel conditional $(\mathsf{pcond}\ M\ N\ P)$, where $M$, $N$, and $P$ have type $\iota$, differs from $(\mathsf{cond}\ M\ N\ P)$ operationally in one respect: if $N$ and $P$ reduce to the same numeral, $(\mathsf{pcond}\ M\ N\ P)$ reduces to that numeral even if $M$ diverges. Parallel conditional is necessary for making the standard denotational models fully abstract.

PCF is the syntax of call-by-name and call-by-value PCF. For lazy PCF, we add the formation rule

$$\text{Convergence-testing} \quad \frac{M : \sigma \qquad N : \tau}{(\mathsf{conv}\, M\ N) : \tau}$$

and call the resultant set of terms LPCF. Informally, $(\mathsf{conv}\, M\ N)$ returns the value of $N$ if the interpretation of $M$ halts, and otherwise diverges.

We adopt many of the standard notational conventions of the $\lambda$-calculus [2]. For instance, terms are denoted by the letters $M$, $N$, $P$, $Q$, $S$, and $T$. Parentheses may be dropped from applications under the assumption that application associates to the left, *i.e.*, $(M\ N\ P)$ is short for $((M\ N)\ P)$.

[1] There are alternative ways of building a syntax of PCF. Most notably, the syntax can be defined using constants instead of term constructors [22]. Using constants for *conditionals*, however, leads to a rather arcane operational semantics of call-by-value PCF (*cf.* [35]).

| | | | |
|---|---|---|---|
| Variables | $x_i^\sigma : \sigma$, where $i \in \mathbb{N}$ | | |
| $\lambda$-abstraction | $\dfrac{M : \sigma}{(\lambda x_i^\tau . M) : (\tau \to \nu)}$ | Application | $\dfrac{M : (\tau \to \nu) \quad N : \tau}{(M\ N) : \nu}$ |
| Numerals | $0, 1, 2, \ldots : \iota$ | Recursion | $\dfrac{M : \sigma}{\mu x_i^\sigma . M : \sigma}$ |
| Successor | $\dfrac{M : \iota}{(\mathsf{succ}\ M) : \iota}$ | Predecessor | $\dfrac{M : \iota}{(\mathsf{pred}\ M) : \iota}$ |
| Conditional | $\dfrac{M : \iota \quad N : \sigma \quad P : \sigma}{(\mathsf{cond}\ M\ N\ P) : \sigma}$ | Parallel conditional | $\dfrac{M : \iota \quad N : \iota \quad P : \iota}{(\mathsf{pcond}\ M\ N\ P) : \iota}$ |

Table 1: Syntactic formation rules for PCF.

We will also drop types from variables whenever the types are unimportant or can be deduced from the context, and use the letters $u$, $v$, $w$, $x$, $y$, and $z$ to denote variables. The usual definitions of free and bound variables apply here, and terms are identified up to renaming of bound variables [2]. Finally, syntactic substitution is written $M[x := N]$, where the substitution renames bound variables to avoid capturing the free variables of $N$ [2].

## 2.2 Operational and Denotational Semantics of PCF

The operational semantics of PCF can be defined by a **deductive semantics.**[2] A deductive semantics defines a binary relation $\Downarrow$ on terms by rules based on the structure of terms; we write $M \Downarrow V$ (read "$M$ halts at value $V$") when there is a proof tree with result $M \Downarrow V$, whose nodes are instances of the rules defining the relation $\Downarrow$. It is important to understand the substantial difference between deductive semantics and rewrite or "structured operational semantics" [2, 23]. In deductive semantics, terms are written to values in one big step, whereas in rewrite semantics, the single-step relation may need to be used multiple times in order to rewrite a term to a value.

Each language has its own $\Downarrow$ relation, called $\Downarrow_n$, $\Downarrow_v$, and $\Downarrow_l$ for call-by-name, call-by-value, and lazy PCF respectively. The rules defining these relations include the rules of Table 2.[3] Rules specific to the three languages appear in Table 3.

The denotational models for the three languages—called $\mathcal{N}$, $\mathcal{V}$, and $\mathcal{L}$—are environment models whose the underlying spaces are Scott domains [10]. For every type $\sigma$, we assign domains $\mathcal{N}^\sigma$, $\mathcal{V}^\sigma$, and $\mathcal{L}^\sigma$. The three denotational models share much of the same structure, *e.g.*, the poset $\mathbb{N}_\perp$ (natural numbers with $\perp$, where $\perp \sqsubseteq n$ for any natural number $n$ and $n \not\sqsubseteq k$ for any distinct

[2]This form of semantics has been given the title "natural semantics" by Gilles Kahn and others; it has also been called an "observation calculus" by Bloom [4]. We call this form of semantics a "deductive semantics" to emphasize the resemblance of the interpretation of terms to proof trees.

[3]The expert reader may recall that Plotkin's interpreter for call-by-name PCF diverges on (pred 0) [22], whereas our interpreter returns 0. This is a minor design change that makes the denotational semantics of the three languages easier to use.

| | |
|---|---|
| $V \Downarrow V$, $V$ a value | $\dfrac{M \Downarrow 0 \qquad N \Downarrow V}{(\mathsf{cond}\ M\ N\ P) \Downarrow V}$ |
| $\dfrac{M \Downarrow n}{(\mathsf{succ}\ M) \Downarrow (n+1)}$ | $\dfrac{M \Downarrow (n+1) \qquad P \Downarrow V}{(\mathsf{cond}\ M\ N\ P) \Downarrow V}$ |
| $\dfrac{M \Downarrow (n+1)}{(\mathsf{pred}\ M) \Downarrow n}$ | $\dfrac{M \Downarrow 0 \qquad N \Downarrow k}{(\mathsf{pcond}\ M\ N\ P) \Downarrow k}$ |
| $\dfrac{M \Downarrow 0}{(\mathsf{pred}\ M) \Downarrow 0}$ | $\dfrac{M \Downarrow (n+1) \qquad P \Downarrow k}{(\mathsf{pcond}\ M\ N\ P) \Downarrow k}$ |
| $\dfrac{M[x := \mu x.\, M] \Downarrow V}{(\mu x.\, M) \Downarrow V}$ | $\dfrac{N \Downarrow k \qquad P \Downarrow k}{(\mathsf{pcond}\ M\ N\ P) \Downarrow k}$ |

Table 2: Deductive semantics rules for applying constants and reducing conditionals.

| | |
|---|---|
| Call-by-name | $\dfrac{M \Downarrow_n \lambda x.\, M' \qquad M'[x := N] \Downarrow_n V}{(M\ N) \Downarrow_n V}$ |
| Lazy | $\dfrac{M \Downarrow_l \lambda x.\, M' \qquad M'[x := N] \Downarrow_l V}{(M\ N) \Downarrow_l V}$ $\qquad$ $\dfrac{M \Downarrow_l V' \qquad N \Downarrow_l V}{(\mathsf{conv}\ M\ N) \Downarrow_l V}$ |
| Call-by-value | $\dfrac{M \Downarrow_v \lambda x.\, M' \qquad N \Downarrow_v V' \qquad M'[x := V'] \Downarrow_v V}{(M\ N) \Downarrow_v V}$ |

Table 3: Deductive semantics rules specific to the three languages.

$$
\begin{array}{rcl}
[\![x^\sigma]\!]\rho & = & \rho(x) \\
[\![n]\!]\rho & = & n \\
[\![\mu x^\sigma.\, M]\!]\rho & = & \bigsqcup_{n\geq 0} f^n(\bot), \quad \text{where } f(d) = [\![M]\!]\rho[x \mapsto d] \\
[\![\mathsf{succ}\ M]\!]\rho & = & \begin{cases} \bot & \text{if } [\![M]\!]\rho = \bot \\ [\![M]\!]\rho + 1 & \text{otherwise} \end{cases} \\
[\![\mathsf{pred}\ M]\!]\rho & = & \begin{cases} \bot & \text{if } [\![M]\!]\rho = \bot \\ 0 & \text{if } [\![M]\!]\rho = 0 \\ [\![M]\!]\rho - 1 & \text{otherwise} \end{cases} \\
[\![\mathsf{cond}\ M\ N\ P]\!]\rho & = & \begin{cases} \bot & \text{if } [\![M]\!]\rho = \bot \\ [\![N]\!]\rho & \text{if } [\![M]\!]\rho = 0 \\ [\![P]\!]\rho & \text{otherwise} \end{cases} \\
[\![\mathsf{pcond}\ M\ N\ P]\!]\rho & = & \begin{cases} [\![N]\!]\rho & \text{if } [\![M]\!]\rho = 0 \\ [\![P]\!]\rho & \text{if } [\![M]\!]\rho > 0 \\ [\![P]\!]\rho & \text{if } [\![N]\!]\rho = [\![P]\!]\rho \\ \bot & \text{otherwise} \end{cases}
\end{array}
$$

Table 4: Equations common to the denotational semantics of call-by-name, lazy, and call-by-value PCF. Here, $\bot$ denotes the least element in the appropriate domain. Abusing notation, we write $n$ for both the numeral denoting $n$ and the natural number $n$ itself.

natural numbers $n$ and $k$) is the domain assigned to $\mathcal{N}^\iota$, $\mathcal{V}^\iota$, and $\mathcal{L}^\iota$. The semantic clauses in Table 4 are shared among the three models as well; clauses specific to the three languages appear in Table 5. The only differences between the three models are the domains assigned to the functional types, and the corresponding semantic clauses for interpreting $\lambda$-abstractions and applications.

## 2.3 Call-by-name PCF

The **call-by-name** interpreter requires one more rule beyond those appearing in Table 2. This is the rule for evaluating applications of $\lambda$-abstractions to arguments, where the arguments are passed call-by-name, *i.e.*, without evaluation. This rule appears in Table 3. For call-by-name PCF, we observe only numerals [22], and hence the observational approximation relation is

**Definition 2.1** $M \sqsubseteq_{name} N$ if for any PCF-context $C[\cdot]$, $C[M] \Downarrow_n k$ implies $C[N] \Downarrow_n k$.

For example, let $\Omega = \mu f^{\iota\to\iota}.\, f$; then $(\lambda x.\, \Omega\ x) \sqsubseteq_{name} \Omega$.

The denotational model of call-by-name PCF is built out of continuous functions. The functional spaces are defined inductively by $\mathcal{N}^{\tau\to\nu} = [\mathcal{N}^\tau \to_c \mathcal{N}^\nu]$ where $[A \to_c B]$ is the Scott domain of continuous functions from domain $A$ to domain $B$ ordered pointwise [10]. The missing semantic

| | |
|---|---|
| Call-by-name | $\mathcal{N}[\![\lambda x.\, M]\!]\rho = f$, where $f(d) = \mathcal{N}[\![M]\!]\rho[x \mapsto d]$<br>$\mathcal{N}[\![M\ N]\!]\rho = (\mathcal{N}[\![M]\!]\rho)\,(\mathcal{N}[\![N]\!]\rho)$ |
| Lazy | $\mathcal{L}[\![\lambda x.\, M]\!]\rho = \mathsf{lift}(g)$, where $g(d) = \mathcal{L}[\![M]\!]\rho[x \mapsto d]$<br>$\mathcal{L}[\![M\ N]\!]\rho = (\mathcal{L}[\![M]\!]\rho) \bullet_l (\mathcal{L}[\![N]\!]\rho)$<br>$\mathcal{L}[\![\mathsf{conv}\, M\, N]\!]\rho = \begin{cases} \bot & \text{if } \mathcal{L}[\![M]\!]\rho = \bot \\ \mathcal{L}[\![N]\!]\rho & \text{otherwise} \end{cases}$ |
| Call-by-value | $\mathcal{V}[\![\lambda x.\, M]\!]\rho = \mathsf{lift}(\mathsf{strict}(g))$, where $g(d) = \mathcal{V}[\![M]\!]\rho[x \mapsto d]$<br>$\mathcal{V}[\![M\ N]\!]\rho = (\mathcal{V}[\![M]\!]\rho) \bullet_v (\mathcal{V}[\![N]\!]\rho)$ |

Table 5: Extra semantic clauses for the three languages.

clauses appear in Table 5. For the purposes of this paper, the most important use of the denotational semantics will be to prove operational facts using the following theorem [22, 28]:

**Theorem 2.2** *Adequacy and inequational full abstraction for $\mathcal{N}$:*

- *Adequacy: For any closed term $M$ of type $\iota$, $M \Downarrow_n k$ iff $\mathcal{N}[\![M]\!] = \mathcal{N}[\![k]\!]$.*
- *Inequational full abstraction: $M \sqsubseteq_{name} N$ iff for all $\rho$, $\mathcal{N}[\![M]\!]\rho \sqsubseteq \mathcal{N}[\![N]\!]\rho$.*

The notions of adequacy and full abstraction for models obviously parallels the notions of adequacy and full abstraction for translations.

### 2.3.1 Lazy PCF

Like the call-by-name interpreter, the **lazy** interpreter also passes arguments by-name. Nevertheless, there is one significant difference between the two languages: lazy PCF includes extra terms for convergence-testing. These extra terms are interpreted by the rule given in Table 3. In lazy PCF we observe numerals, so

**Definition 2.3** $M \sqsubseteq_{lazy} N$ if for any LPCF-context $C[\cdot]$, $C[M] \Downarrow_l k$ implies $C[N] \Downarrow_l k$.

For instance, $(\lambda x.\, P) \not\sqsubseteq_{lazy} \Omega$: the context $(\mathsf{conv}\,[\cdot]\,0)$ distinguishes these terms, since $(\mathsf{conv}\,\Omega\ 0) \Uparrow_l$ but $(\mathsf{conv}\,(\lambda x.\, P)\,0) \Downarrow_l 0$.

It is important to notice that one may observe *termination* and obtain the same observational approximation relation. Suppose, for example, $M \not\sqsubseteq_{lazy} N$ by the context $C[\cdot]$ and $C[M] \Downarrow_l m$ and $C[N] \Downarrow_l n$ with $m < n$. Then the context $D[\cdot] = (\mathsf{cond}\,(\mathsf{pred}^m\ C[\cdot])\,0\,\Omega)$, where

$$(\mathsf{pred}^m\ P) = \underbrace{(\mathsf{pred}\,(\mathsf{pred}\ \ldots(\mathsf{pred}}_{m \text{ times}}\ P)))$$

forces $D[M]$ to converge but $D[N]$ to diverge. Both numeral and termination observations will be important below, even though only one kind of observation is essential in defining the lazy observational congruence relation.

The denotational model reflects the termination properties of lazy PCF using the domain-theoretic operation of *lifting*. Lifting is used at functional type to distinguish between divergent terms, which will mean $\bot$, and convergent terms, which will have denotations in the lifted part of the space. Formally, for any poset $D$ with a least element, $D_\bot$ is the poset consisting of $D$ and a new element $\bot$, with $\bot$ ordered below every element of $D$. Importantly, there are well-defined injection and projection functions $\mathsf{lift} : D \to D_\bot$ and $\mathsf{drop} : D_\bot \to D$, where $\mathsf{drop}(\mathsf{lift}(d)) = d$ and $\mathsf{lift}(d) \neq \bot$ for all $d \in D$. The function spaces for $\mathcal{L}$ are then defined inductively by $\mathcal{L}^{\tau\to\nu} = [\mathcal{L}^\tau \to_c \mathcal{L}^\nu]_\bot$. The semantic clauses for interpreting terms in this structure appear in Tables 4 and 5, where $d \bullet_l e = \mathsf{drop}(d)(e)$ is used to simplify the notation. (Note that $\bullet_l$ is continuous in both arguments.) This model matches the operational semantics of lazy PCF [5].

**Theorem 2.4** *Adequacy and inequational full abstraction for $\mathcal{L}$:*

- *Adequacy: For any closed term $M$ of type $\iota$, $M \Downarrow_l k$ iff $\mathcal{L}[\![M]\!] = \mathcal{L}[\![k]\!]$. Moreover, for any closed term $M$, $M \Downarrow_l$ iff $\mathcal{L}[\![M]\!] \neq \bot$.*
- *Inequational full abstraction: $M \sqsubseteq_{lazy} N$ iff for all environments $\rho$, $\mathcal{L}[\![M]\!]\rho \sqsubseteq \mathcal{L}[\![N]\!]\rho$.*

### 2.3.2 Call-by-value PCF

The call-by-value version of PCF differs little from call-by-name or lazy PCF. The main difference between the languages arises in the parameter-passing mechanism implemented by the interpreter: in call-by-value PCF, all arguments are reduced to values before being substituted for formal parameters. The formal rule appears in Table 3. The observations of call-by-value PCF are numerals, so as before,

**Definition 2.5** $M \sqsubseteq_{val} N$ if for any PCF-context $C[\cdot]$, $C[M] \Downarrow_v k$ implies $C[N] \Downarrow_v k$.

One example of a call-by-value observational congruence is $(\lambda x.\, x) \equiv_{val} (\lambda x.\, \lambda y.\, x\ y)$. Suppose both terms are placed in a context $C[\cdot]$. If $x$ is ever instantiated by a term $N$ during the evaluation of $C[\lambda x.\, x]$, that term $N$ *must* be a value. Since values always halt, $N \equiv_{val} (\lambda y.\, N\ y)$. The observational congruence now follows from this fact.

The denotational model of call-by-value PCF is built out of *strict* continuous functions, *i.e.*, those continuous functions $f$ for which $f(\bot) = \bot$. Let $[A \to_s B]$ denote the Scott domain of strict, continuous functions from $A$ to $B$, ordered pointwise [10]. The functional spaces of $\mathcal{V}$ are defined inductively by $\mathcal{V}^{\tau\to\nu} = [\mathcal{V}^\tau \to_s \mathcal{V}^\nu]_\bot$, and the language is interpreted using the semantic clauses of Tables 4 and 5, where $d \bullet_v e = \mathsf{drop}(d)(e)$ and where

$$\mathsf{strict}(g)(e) = \begin{cases} \bot & \text{if } e = \bot \\ g(e) & \text{otherwise.} \end{cases}$$

(Note that $\bullet_v$ is continuous in both arguments.) This model fits the call-by-value operational semantics [9, 32, 34].

**Theorem 2.6** *Adequacy and inequational full abstraction for $\mathcal{V}$:*

- *Adequacy: For any closed term $M$ of type $\iota$, $M \Downarrow_v k$ iff $\mathcal{V}[\![M]\!] = k$. Moreover, for any closed term $M$, $M \Downarrow_v$ iff $\mathcal{V}[\![M]\!] \neq \bot$.*
- *Inequational full abstraction: $M \sqsubseteq_{val} N$ iff for all environments $\rho$, $\mathcal{V}[\![M]\!]\rho \sqsubseteq \mathcal{V}[\![N]\!]\rho$.*

# 3 Translation from Call-by-Value to Lazy PCF

This section thoroughly explores one translation from call-by-value to lazy PCF. First we define a basic, albeit naive translation. This translation will satisfy the adequacy property but *not* the full abstraction property. The translation is then repaired so that it becomes fully abstract. The proofs of adequacy and full abstraction are developed in detail in this section, since the basic techniques employed, right down to the statements of lemmas, may be carried over for the other translations considered in Sections 4 and 5.

## 3.1 The basic translation

In [20], Ong defines a translation from call-by-value to lazy PCF. The idea behind this translation is familiar and simple. Since the call-by-value interpreter forces evaluation of operands in applications, the translation converts call-by-value functions to lazy functions which check their arguments for convergence. The translation is defined by induction on the structure of terms:[4]

$$\begin{array}{rclcrcl}
\overline{x^\sigma} & = & x^\sigma & & \overline{n} & = & n \\
\overline{(M\ N)} & = & (\overline{M}\ \overline{N}) & & \overline{(\lambda x^\sigma.\, M)} & = & (\lambda x^\sigma.\, \mathsf{conv}\, x\ \overline{M}) \\
\overline{(\mathsf{succ}\, M)} & = & (\mathsf{succ}\, \overline{M}) & & \overline{(\mathsf{cond}\ M\ N\ P)} & = & (\mathsf{cond}\ \overline{M}\ \overline{N}\ \overline{P}) \\
\overline{(\mathsf{pred}\, M)} & = & (\mathsf{pred}\, \overline{M}) & & \overline{(\mathsf{pcond}\ M\ N\ P)} & = & (\mathsf{pcond}\ \overline{M}\ \overline{N}\ \overline{P}) \\
\overline{(\mu x^\sigma.\, M)} & = & (\mu x^\sigma.\, \overline{M}) & & & &
\end{array}$$

## 3.2 Adequacy

Importantly, this translation satisfies the adequacy property, *i.e.*,

**Theorem 3.1** *For any closed term PCF-term $M$,*

- *$M \Downarrow_v n$ iff $\overline{M} \Downarrow_l n$; and*
- *$M \Downarrow_v$ iff $\overline{M} \Downarrow_l$.*

Most proofs of adequacy for translations are based on connections between the interpreters of the language [20, 21]. Ong, for instance, proves the adequacy of his translation by setting up a tight correspondence between steps in the interpreters for his two languages. But a technically simpler, *semantic* proof is also possible for this translation, using the models $\mathcal{V}$ and $\mathcal{L}$ and the fact that these models are *adequate*. This is the approach we will take.

The proof relies upon showing that there are elements of the lazy model (corresponding to strict functions) that represent elements of the call-by-value model. The following inductively-defined

[4] There are a few technical differences between this translation and Ong's: Ong's translation tests for convergence in the application case rather than in the abstraction case, and works with untyped languages without conditionals, arithmetic, or recursion. Nevertheless, the spirit of the translations is the same, and both are adequate but not fully abstract.

relation, an instance of a **logical relation** [14, 37], states this relationship between elements of the two models:

**Definition 3.2** Define the relations $R^\sigma \subseteq \mathcal{V}^\sigma \times \mathcal{L}^\sigma$ by induction on types as follows:

1. $d\ R^\iota\ e$ iff $d = e$; and
2. $f\ R^{\sigma\to\tau}\ g$ iff $(f \neq \bot \iff g \neq \bot)$, and for any $d\ R^\sigma\ e$, $(f \bullet_v d)\ R^\tau\ (g \bullet_l e)$.

This definition should be compared to the definitions of logical relations in [6, 16, 25]. There is an operational justification for this relation. For instance, recall that divergent terms mean $\bot$ in both $\mathcal{V}$ and $\mathcal{L}$. Thus, $R$ relates the meanings of all divergent terms in call-by-value and lazy PCF.

We begin with a technical lemma on the relations that will be needed to handle recursions.

**Lemma 3.3** *Suppose* $d_i\ R^\sigma\ e_i$, *and* $(\bigsqcup d_i)$ *and* $(\bigsqcup e_i)$ *exist. Then* $(\bigsqcup d_i)\ R^\sigma\ (\bigsqcup e_i)$.

**Proof:** By induction on types. The basis is straightforward since $R^\iota$ is the identity relation. Now consider the induction case, and let $d = (\bigsqcup d_i)$ and $e = (\bigsqcup e_i)$. It is not hard to see that $d = \bot$ iff $e = \bot$. Now suppose $d'\ R^\sigma\ e'$. By hypothesis, $(d_i \bullet_v d')\ R^\tau\ (e_i \bullet_l e')$, so by induction

$$\bigsqcup (d_i \bullet_v d')\ R^\tau\ \bigsqcup (e_i \bullet_l e')$$

By the continuity of $\bullet_v$ and $\bullet_l$, $((\bigsqcup d_i) \bullet_v d')\ R^\tau\ ((\bigsqcup e_i) \bullet_l e')$ and so $(d \bullet_v d')\ R^\tau\ (e \bullet_v e')$ as desired. ■

This property which is sometimes called "inclusivity" or "directed completeness," since $R$ preserves least upper bounds of directed sets.

The key lemma needed for the proof of Theorem 3.1—an analog of the Fundamental Theorem of Logical Relations [14, 37]—shows that the meanings of all call-by-value terms are related to their lazy translates. To relate the meanings of *open* terms (which will be encountered inductively), we need a condition on environments: a $\mathcal{V}$-environment $\rho$ and an $\mathcal{L}$-environment $\rho'$ are **compatible** if for any variable $x^\sigma$, $\rho(x^\sigma)\ R^\sigma\ \rho'(x^\sigma)$. Then

**Lemma 3.4** *For any PCF-term* $M$ *and compatible* $\rho$ *and* $\rho'$, $\mathcal{V}[\![M]\!]\rho\ R^\sigma\ \mathcal{L}[\![\overline{M}]\!]\rho'$.

**Proof:** By induction on the structure of terms. In the basis, $M$ is either a variable or numeral. If $M$ is a variable $x^\sigma$, then by hypothesis

$$\mathcal{V}[\![x^\sigma]\!]\rho = \rho(x^\sigma)\ R^\sigma\ \rho'(x^\sigma) = \mathcal{L}[\![\overline{x^\sigma}]\!]\rho'$$

If $M$ is a numeral $n$, then $\mathcal{V}[\![n]\!]\rho = n\ R^\iota\ n = \mathcal{L}[\![\overline{n}]\!]\rho'$.

There are seven cases in the induction step; we consider application, $\lambda$-abstraction, and recursion here and leave the remaining cases dealing with successor, predecessor, and the conditionals to the reader. First, suppose $M = (M_1\ M_2)$. By induction, for any compatible $\rho$ and $\rho'$, $\mathcal{V}[\![M_i]\!]\rho\ R\ \mathcal{L}[\![\overline{M_i}]\!]\rho'$. Thus, by the definition of $R$,

$$\begin{aligned} \mathcal{V}[\![M]\!]\rho &= (\mathcal{V}[\![M_1]\!]\rho) \bullet_v (\mathcal{V}[\![M_2]\!]\rho) \\ R &\quad (\mathcal{L}[\![\overline{M_1}]\!]\rho') \bullet_l (\mathcal{L}[\![\overline{M_2}]\!]\rho') \\ R &\quad \mathcal{L}[\![\overline{M}]\!]\rho' \end{aligned}$$

as desired.

Second, suppose $M = \lambda x^\sigma.\, N$. Let $d = \mathcal{V}[\![M]\!]\rho$ and $e = \mathcal{L}[\![\overline{M}]\!]\rho'$. Obviously $d \neq \bot$ and $e \neq \bot$, since both are the meanings of $\lambda$-abstractions. Now we consider their meanings when applied. Suppose $d'\ R^\sigma\ e'$. If $d' = \bot$, then $e' = \bot$ and thus $d \bullet_v d' = \bot\ R^\tau\ \bot = e \bullet_l e'$ since $e$ checks its argument for convergence. If, on the other hand, $d' \neq \bot$, then $e' \neq \bot$ and so

$$\begin{array}{rcl} d \bullet_v d' & = & \mathcal{V}[\![N]\!]\rho[x \mapsto d'] \\ & R & \mathcal{L}[\![\overline{N}]\!]\rho'[x \mapsto e'] \\ & R & e \bullet_l e' \end{array}$$

where the second line follows from the induction hypothesis and the fact that $\rho[x \mapsto d']$ and $\rho'[x \mapsto e']$ are compatible. Thus, $d\ R\ e$ as desired.

Finally, suppose $M = \mu x^\sigma.\, N$. Let $f(d) = \mathcal{V}[\![N]\!]\rho[x \mapsto d]$ and $g(e) = \mathcal{L}[\![\overline{N}]\!]\rho'[x \mapsto e]$. First, it follows easily from the definition of the relations that $f^0(\bot) = \bot\ R^\sigma\ \bot = g^0(\bot)$. By the induction hypothesis,

$$f^1(\bot) = \mathcal{V}[\![N]\!]\rho[x \mapsto \bot]\ R^\sigma\ \mathcal{L}[\![\overline{N}]\!]\rho'[x \mapsto \bot] = g^1(\bot)$$

since $\rho[x \mapsto \bot]$ and $\rho'[x \mapsto \bot]$ are compatible environments. Thus, using a simple induction on $n$, it is easy to see that $f^n(\bot)\ R^\sigma\ g^n(\bot)$ for all $n$. Since $\{f^n(\bot) : n \geq 0\}$ and $\{g^n(\bot) : n \geq 0\}$ are both chains, their lub's exist and hence by Lemma 3.3 we may conclude

$$\mathcal{V}[\![M]\!]\rho = \bigsqcup_{n \geq 0} f^n(\bot)\ R^\sigma\ \bigsqcup_{n \geq 0} g^n(\bot) = \mathcal{L}[\![\overline{M}]\!]\rho'$$

as desired. ■

**Proof of Theorem 3.1:** Suppose, for instance, that $M$ is a closed PCF-term and $M \Downarrow_v$. Then by the adequacy theorem for $\mathcal{V}$, $\mathcal{V}[\![M]\!] \neq \bot$. Since it follows from Lemma 3.4 that $\mathcal{V}[\![M]\!]\ R^\sigma\ \mathcal{L}[\![\overline{M}]\!]$, it must be the case that $\mathcal{L}[\![\overline{M}]\!] \neq \bot$. Thus, by the adequacy theorem for $\mathcal{L}$, $M \Downarrow_l$. The converse and the case when $M$ is of type $\iota$ follow along similar lines. ■

## 3.3 Failure of full abstraction

Theorem 3.1, together with the fact that the translation is *compositional—i.e.*, the translation of a term is defined by the translation of its components—implies one direction of full abstraction.

**Corollary 3.5** *For any PCF-terms $M$ and $N$, $\overline{M} \sqsubseteq_{lazy} \overline{N}$ implies $M \sqsubseteq_{val} N$.*

**Proof:** Suppose $M \not\sqsubseteq_{val} N$. Then there is a context $C[\cdot]$ in which either

- $C[M] \Downarrow_v$ and $C[N] \Uparrow_v$; or
- $C[M] \Downarrow_v m$ and $C[N] \Downarrow_v n$, where $m$ and $n$ are distinct numerals.

Suppose the former of these cases holds (the latter case can be argued similarly). By Theorem 3.1, it follows that $\overline{C[M]} \Downarrow_l$ and $\overline{C[N]} \Uparrow_l$. Because the translation is compositional, $\overline{C[M]} = \overline{C}[\overline{M}]$, where "holes" in contexts are translated to "holes." Similarly, $\overline{C[N]} = \overline{C}[\overline{N}]$. Thus, this context $\overline{C}[\cdot]$ distinguishes $\overline{M}$ and $\overline{N}$, so $\overline{M} \not\sqsubseteq_{lazy} \overline{N}$. ■

The converse of Corollary 3.5 fails, and hence the translation is not fully abstract. A simple example demonstrates this fact. Consider the PCF-terms $M_1 = \lambda x^{\iota\to\iota}.\, x$ and $M_2 = \lambda x^{\iota\to\iota}.\, \lambda y^{\iota}.\, (x\ y)$. One may verify that $M_1 \equiv_{val} M_2$ using the model $\mathcal{V}$, but the terms

$$\begin{aligned}\overline{M_1} &= \lambda x^{\iota\to\iota}.\, \mathsf{conv}\, x\ x\\ \overline{M_2} &= \lambda x^{\iota\to\iota}.\, \mathsf{conv}\, x\ (\lambda y^{\iota}.\, \mathsf{conv}\, y\ (x\ y))\end{aligned}$$

are not lazy observationally congruent: the context $C[\cdot] = [\cdot]\ (\lambda z.\, 3)\ \Omega$ (where $\Omega$ is any divergent term of type $\iota$) causes the first term to return 3 and the second to diverge under the lazy interpreter.

What is the problem with the translation? The problem lies in the translation of variables. For instance, the variable $x$ in $\overline{M_1}$ can be instantiated with any LPCF-term, including $(\lambda x.\, 3)$ which is not strict in its argument. In other words, $\overline{M_1}$ contains variables that do not range over the target of the translation. The term $\overline{M_2}$, on the other hand, forces $x$ to diverge if its argument $y$ diverges. If there were some uniform way to force a term of functional type to be strict, we could guarantee that the variable $x$ in $\overline{M_1}$ would range over only strict functions.

There are two possible recourses for obtaining full abstraction. On the one hand, one could change the definition of $\sqsubseteq_{lazy}$ so that contexts with strict functions are the only ones allowed. This would probably be enough to guarantee that $M \sqsubseteq_{val} N$ iff $\overline{M} \sqsubseteq_{lazy} \overline{N}$. On the other hand, one could change the translation so that it becomes fully abstract. We shall take the second course, since we want to see if it is possible to obtain a fully abstract translation; the other course is left open, although it has been considered elsewhere for other translations [39].

## 3.4 Full abstraction

Forcing terms of functional type to be strict is the key idea in repairing the translation. Define terms $\delta^{\sigma}$ of type $(\sigma \to \sigma)$ as follows:

$$\begin{aligned}\delta^{\iota} &= \lambda x^{\iota}.\, x\\ \delta^{\tau\to\nu} &= \lambda x^{\tau\to\nu}.\, \mathsf{conv}\, x\ (\lambda y^{\tau}.\, \mathsf{conv}\, y\ (\delta^{\nu}\ (x\ (\delta^{\tau}\ y))))\end{aligned}$$

These $\delta$'s are "strictifying" functions. The function $\delta^{\tau\to\nu}$ makes its first argument $x$ strict by checking the second argument $y$ for convergence, then passing the strict version of $y$ to $x$ and "strictifying" the result.

A first important observation is that "strictifying" twice is the same as "strictifying" once. Abusing notation, we write $\delta^{\sigma}$ for $\mathcal{L}[\![\delta^{\sigma}]\!]$.

**Lemma 3.6** *$\delta^{\sigma}$ is a retraction, i.e., $\delta^{\sigma} \bullet_l (\delta^{\sigma} \bullet_l e) = \delta^{\sigma} \bullet_l e$.*

**Proof:** By induction on types. The basis is easy to verify, since $\delta^{\iota}$ is the identity function. Now consider the induction step, where $\sigma = (\tau \to \nu)$. There are two cases: either $e = \bot$ or $e \neq \bot$. If $e = \bot$, then $\delta \bullet_l e = \bot = \delta \bullet_l (\delta \bullet_l e)$. If $e \neq \bot$, note that neither $(\delta \bullet_l e)$ nor $(\delta \bullet_l (\delta \bullet_l e))$ is $\bot$, since both are the meanings of $\lambda$-abstractions. Thus, both elements are in the lifted part of the domain $\mathcal{L}^{\tau\to\nu}$, or in other words, both $(\delta \bullet_l e)$ and $(\delta \bullet_l (\delta \bullet_l e))$ are lifted *functions*. Thus, to show these elements are equivalent, it is enough to show that they agree when applied to any element in $\mathcal{L}^{\tau}$ via $\bullet_l$ (recall that $d \bullet_l d' = \mathsf{drop}(d)(d')$). So suppose $e' \in \mathcal{L}^{\tau}$. If $e' = \bot$, then

$(\delta \bullet_l e) \bullet_l e' = \bot = (\delta \bullet_l (\delta \bullet_l e)) \bullet_l e'$. If $e' \neq \bot$, then

$$\begin{aligned}(\delta^{\tau\to\nu} \bullet_l (\delta^{\tau\to\nu} \bullet_l e)) \bullet_l e' &= \delta^\nu \bullet_l ((\delta^{\tau\to\nu} \bullet_l e) \bullet_l (\delta^\tau \bullet_l e')) \\ &= \delta^\nu \bullet_l (\delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l (\delta^\tau \bullet_l e')))) \\ &= \delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l e')) \\ &= (\delta^{\tau\to\nu} \bullet_l e) \bullet_l e'\end{aligned}$$

where the first, second, and fourth lines follow from the definition of $\delta^{\tau\to\nu}$, and the third line follows by the induction hypothesis. Thus, since neither $(\delta \bullet_l e)$ nor $(\delta \bullet_l (\delta \bullet_l e))$ is $\bot$, the elements $(\delta \bullet_l e)$ and $(\delta \bullet_l (\delta \bullet_l e))$ are the same lifted function. ■

The functions $\delta^\sigma$ are the essential ingredient to repairing the translation. We modify the translation so that variables are translated via the clause

$$\overline{x^\sigma} = (\delta^\sigma x^\sigma)$$

with all other clauses given as before. From now on, let $\overline{M}$ denote the translation of a term $M$ under the modified translation. This translation is adequate and fully abstract.

**Theorem 3.7** *The new translation satisfies the following properties:*

1. *Adequacy: For any closed PCF-term $M$, $M \Downarrow_v$ iff $\overline{M} \Downarrow_l$. Moreover, if $M$ is of base type, then $M \Downarrow_v n$ iff $\overline{M} \Downarrow_l n$.*
2. *Inequational Full Abstraction: For any $M$ and $N$, $M \sqsubseteq_{val} N$ iff $\overline{M} \sqsubseteq_{lazy} \overline{N}$.*

The proof of adequacy of the new translation follows along the same lines as the proof of Theorem 3.1: the only modification necessary is the variable case of Lemma 3.4, which may easily be seen to follow from

**Lemma 3.8** *If $d\ R^\sigma\ e$, then $d\ R^\sigma\ (\delta^\sigma \bullet_l e)$.*

**Proof:** (Sketch) By induction on types. ■

The ($\Leftarrow$) direction of full abstraction now follows from the adequacy result, using the same argument given in the proof of Corollary 3.5.

In contrast, the proof of the ($\Rightarrow$) direction of full abstraction requires some new ideas, although as before, the proof relies on the models $\mathcal{V}$ and $\mathcal{L}$. By the full abstraction theorems for $\mathcal{L}$ (Theorem 2.4) and for $\mathcal{V}$ (Theorem 2.6), it is sufficient to show that $\mathcal{V}[\![M]\!] \sqsubseteq \mathcal{V}[\![N]\!]$ implies $\mathcal{L}[\![\overline{M}]\!] \sqsubseteq \mathcal{L}[\![\overline{N}]\!]$. Suppose $M$ and $N$ are closed terms and $h_1 = \mathcal{L}[\![\overline{M}]\!]$ and $h_2 = \mathcal{L}[\![\overline{N}]\!]$, but $h_1 \not\sqsubseteq h_2$. Thus, there is some way of applying $h_1$ and $h_2$ (using $\bullet_l$) to other elements $e_i$ to obtain some distinction. Our goal is to show that

**Theorem 3.9** *For any $M$ of type $\sigma$ and $\mathcal{L}$-environment $\rho$, $\mathcal{L}[\![\overline{M}]\!]\rho = (\delta^\sigma \bullet_l \mathcal{L}[\![\overline{M}]\!]\rho)$.*

which will imply that $e_i$ can be assumed to be in the range of $\delta$, and that

**Theorem 3.10** *For any $e \in \mathcal{L}^\sigma$ in the range of $\delta^\sigma$ (i.e., $e = (\delta^\sigma \bullet_l e')$ for some $e'$), there is a $d \in \mathcal{V}^\sigma$ such that $d\ R^\sigma\ e$. That is, the relation $R^\sigma$ is surjective on the range of $\delta^\sigma$.*

Intuitively, then, $h_1$ and $h_2$ can be distinguished by legal representations of call-by-value elements. It will follow that $\mathcal{V}[\![M]\!]$ and $\mathcal{V}[\![N]\!]$ are distinguishable.

We begin by proving Theorem 3.9, for which we need the following lemma.

**Lemma 3.11** *For any PCF term $M$, variable $x$ of type $\sigma$, and $\mathcal{L}$-environment $\rho$,*

$$\mathcal{L}[\![\overline{M}]\!]\rho = \mathcal{L}[\![\overline{M}]\!]\rho[x \mapsto (\delta^\sigma \bullet_l \rho(x))].$$

**Proof:** By induction on the structure of $M$, using the fact that $x$ is translated to $(\delta^\sigma\ x)$. ■

**Proof of Theorem 3.9:** By induction on the structure of the term $M$. In the basis, $M$ is either a variable $x$ or a numeral $k$. If $M = x$, then

$$\mathcal{L}[\![\overline{M}]\!]\rho = (\delta^\sigma \bullet_l \rho(x)) = (\delta^\sigma \bullet_l (\delta^\sigma \bullet_l \rho(x))) = (\delta^\sigma \bullet_l \mathcal{L}[\![\overline{M}]\!]\rho)$$

where the second equality follows from Lemma 3.6. If $M = k$, then

$$\mathcal{L}[\![\overline{M}]\!]\rho = k = (\delta^\iota \bullet_l k) = (\delta^\iota \bullet_l \mathcal{L}[\![\overline{M}]\!]\rho)$$

as desired. There are seven cases in the induction step; we consider three cases here and leave the others to the reader.

1. $M = (P\ Q)$. Then by the induction hypothesis,

   $$\mathcal{L}[\![\overline{M}]\!]\rho = (\mathcal{L}[\![\overline{P}]\!]\rho) \bullet_l (\mathcal{L}[\![\overline{Q}]\!]\rho) = (\delta^{\tau\to\sigma} \bullet_l \mathcal{L}[\![\overline{P}]\!]\rho) \bullet_l (\mathcal{L}[\![\overline{Q}]\!]\rho)$$

   If either $\mathcal{L}[\![\overline{P}]\!]\rho$ or $\mathcal{L}[\![\overline{Q}]\!]\rho$ is $\perp$, then $\mathcal{L}[\![\overline{M}]\!]\rho = \perp = (\delta^\sigma \bullet_l \mathcal{L}[\![\overline{M}]\!]\rho)$. Otherwise,

   $$\begin{aligned}(\delta^{\tau\to\sigma} \bullet_l \mathcal{L}[\![\overline{P}]\!]\rho) \bullet_l (\mathcal{L}[\![\overline{Q}]\!]\rho) &= (\delta^\sigma \bullet_l ((\mathcal{L}[\![\overline{P}]\!]\rho) \bullet_l (\delta^\tau \bullet_l \mathcal{L}[\![\overline{Q}]\!]\rho))) \\ &= (\delta^\sigma \bullet_l ((\mathcal{L}[\![\overline{P}]\!]\rho) \bullet_l (\mathcal{L}[\![\overline{Q}]\!]\rho))) = (\delta^\sigma \bullet_l \mathcal{L}[\![\overline{M}]\!]\rho)\end{aligned}$$

   where the first line holds by the definition of $\delta^{\tau\to\sigma}$ and the last line holds by induction.

2. $M = (\lambda x^\tau.\, P)$, where $\sigma = (\tau \to \nu)$. Let $h_1 = \mathcal{L}[\![\lambda x.\, \mathsf{conv}\, x\, \overline{P}]\!]\rho$ and $h_2 = (\delta^\sigma \bullet_l h_1)$. Since $h_1 \neq \perp$, it follows from the definitions of $\delta$ that $h_2 \neq \perp$. We therefore just need to show that $h_1$ and $h_2$ are equivalent when applied using $\bullet_l$. So suppose $d \in \mathcal{L}^\tau$. If $d = \perp$, then $h_1 \bullet_l d = \perp = h_2 \bullet_l d$. If $d \neq \perp$, then

   $$\begin{aligned}h_2 \bullet_l d &= (\delta^\nu \bullet_l (h_1 \bullet_l (\delta^\tau \bullet_l d))) \\ &= (\delta^\nu \bullet_l \mathcal{L}[\![\overline{P}]\!]\rho[x \mapsto (\delta^\tau \bullet_l d)]) \\ &= (\delta^\nu \bullet_l \mathcal{L}[\![\overline{P}]\!]\rho[x \mapsto d]) \\ &= \mathcal{L}[\![\overline{P}]\!]\rho[x \mapsto d] = h_1 \bullet_l d\end{aligned}$$

   where the first line follows from the definition of $\delta^\sigma$, the third line follows by Lemma 3.11, and the fourth line follows by induction.

3. $M = (\mu x^\sigma.\, P)$. Let $f(d) = \mathcal{L}[\![\overline{P}]\!]\rho[x \mapsto d]$. Note that $\perp = (\delta^\sigma \bullet_l \perp)$. Also, for any $n \geq 1$,

$$\begin{aligned} f^n(\perp) &= \mathcal{L}[\![\overline{P}]\!]\rho[x \mapsto f^{n-1}(\perp)] \\ &= \delta^\sigma \bullet_l \mathcal{L}[\![\overline{P}]\!]\rho[x \mapsto f^{n-1}(\perp)] \\ &= \delta^\sigma \bullet_l f^n(\perp) \end{aligned}$$

where the second line holds by induction. Thus,

$$\mathcal{L}[\![\overline{M}]\!]\rho = \bigsqcup_{n\geq 0} f^n(\perp) = \bigsqcup_{n\geq 0} (\delta^\sigma \bullet_l f^n(\perp)) = (\delta^\sigma \bullet_l (\bigsqcup_{n\geq 0} f^n(\perp))) = (\delta^\sigma \bullet_l \mathcal{L}[\![\overline{M}]\!]\rho).$$

This completes the induction step and hence the proof. ■

The main part of the argument is to prove Theorem 3.10. We follow a method due to Friedman [8] and Plotkin [24], showing that the relations $R^\sigma$ are functional, continuous, *and* surjective on the range of $\delta$. (The additional requirements are just extra hypotheses necessary to prove surjectivity.) Define the auxiliary functions $\alpha^\sigma : \mathcal{V}^\sigma \to \mathcal{L}^\sigma$ and $\beta^\sigma : \mathcal{L}^\sigma \to \mathcal{V}^\sigma$, where

$$\begin{aligned} \alpha^\iota(d) &= d & \qquad \beta^\iota(e) &= e \\ \alpha^{\tau\to\nu}(d) &= \begin{cases} \perp & \text{if } d = \perp \\ f & \text{otherwise} \end{cases} & \qquad \beta^{\tau\to\nu}(e) &= \begin{cases} \perp & \text{if } e = \perp \\ g & \text{otherwise} \end{cases} \end{aligned}$$

where $f \neq \perp$ and $g \neq \perp$ and

$$\begin{aligned} f \bullet_l e' &= \begin{cases} \perp & \text{if } e' = \perp \\ \alpha^\nu(d \bullet_v \beta^\tau(\delta^\tau \bullet_l e')) & \text{otherwise} \end{cases} \\ g \bullet_v d' &= \begin{cases} \perp & \text{if } d' = \perp \\ \beta^\nu(e \bullet_l \alpha^\tau(d')) & \text{otherwise} \end{cases} \end{aligned}$$

It is not at all clear that these functions are well-defined. For instance, the result of $\alpha^{\iota\to\iota}(f)$ may not be in the set $\mathcal{L}^{\iota\to\iota} = [\mathbb{N}_\perp \to_c \mathbb{N}_\perp]_\perp$. The following lemma shows that this cannot happen.

**Lemma 3.12** *1. For any $d \in \mathcal{V}^\sigma$ and $e \in \mathcal{L}^\sigma$, $\alpha^\sigma(d) \in \mathcal{L}^\sigma$ and $\beta^\sigma(e) \in \mathcal{V}^\sigma$; and*

*2. $\alpha^\sigma$ and $\beta^\sigma$ are continuous functions.*

**Proof:** By induction on types. The basis is not difficult, since $\mathcal{V}^\iota = \mathcal{L}^\iota$ and $\alpha^\iota$ and $\beta^\iota$ are the identity functions. Now consider the induction case for the type $\sigma = (\tau \to \nu)$:

1. We will show that $f = \alpha^\sigma(d) \in \mathcal{L}^\sigma$; showing that $\beta^\sigma(e) \in \mathcal{V}^\sigma$ is similar and omitted. If $d = \perp$, then $f = \perp \in \mathcal{L}^{\tau\to\nu}$. Now suppose $d \neq \perp$. We need to show that $f$ is a (lifted) continuous function from $\mathcal{L}^\tau$ to $\mathcal{L}^\nu$. Pick any $e' \in \mathcal{L}^\tau$. If $e' = \perp$, then $f \bullet_l e' = \perp \in \mathcal{L}^\nu$. If $e' \neq \perp$, then

$$f \bullet_l e' = \alpha^\nu(d \bullet_v (\beta^\tau(\delta^\tau \bullet_l e'))) \in \mathcal{L}^\nu$$

by induction. Thus, all we need to show is that $f$ is a lifted, continuous function. So suppose $X \subseteq \mathcal{L}^\tau$ is a directed set. If $\bigsqcup X = \perp$, then all elements of $X$ are $\perp$ and hence

$f \bullet_l (\bigsqcup X) = \bot = \bigsqcup_{x \in X} (f \bullet_l x)$. If $\bigsqcup X \neq \bot$, then some element in $X$ is not $\bot$ and hence

$$\begin{aligned} f \bullet_l (\bigsqcup X) &= \alpha^\nu(d \bullet_v (\beta^\tau(\delta^\tau \bullet_l (\bigsqcup X)))) \\ &= \bigsqcup_{x \in X} (\alpha^\nu(d \bullet_v (\beta^\tau(\delta^\tau \bullet_l x)))) \\ &= \bigsqcup_{x \in X} (f \bullet_l x) \end{aligned}$$

where the second line follows by induction (the continuity of $\alpha^\nu$ and $\beta^\tau$) and the continuity of $\bullet_v$ and $\delta^\tau$. Thus, $f \in \mathcal{L}^{\tau \to \nu}$.

2. Again, we will only show $\alpha^{\tau \to \nu}$ is continuous, since the proof that $\beta^{\tau \to \nu}$ is continuous is similar. Suppose $Y \subseteq \mathcal{V}^{\tau \to \nu}$ is directed; our goal is to show that $u = \alpha^{\tau \to \nu}(\bigsqcup Y)$ is the same lifted continuous function as $v = \bigsqcup_{y \in Y} (\alpha^{\tau \to \nu}(y))$. Suppose, on the one hand, $(\bigsqcup Y) = \bot$; then all elements $y \in Y$ are equal to $\bot$. Thus, $u = \bot = v$. Suppose, on the other hand, $(\bigsqcup Y) \neq \bot$. Then $u \neq \bot$ and $v \neq \bot$. To show $u$ and $v$ are equal as lifted functions, suppose $e' \in \mathcal{L}^\tau$. If $e' = \bot$, then

$$u \bullet_l e' = \bot = (\bigsqcup_{y \in Y} (\alpha^{\tau \to \nu}(y))) \bullet_l e' = v \bullet_l e'$$

If $e' \neq \bot$, then

$$\begin{aligned} u \bullet_l e' &= \alpha^\nu((\bigsqcup Y) \bullet_v (\beta^\tau(\delta^\tau \bullet_l e'))) \\ &= \bigsqcup_{y \in Y} (\alpha^\nu(y \bullet_v (\beta^\tau(\delta^\tau \bullet_l e')))) \\ &= \bigsqcup_{y \in Y} (\alpha^{\tau \to \nu}(y) \bullet_l e') \\ &= (\bigsqcup_{y \in Y} (\alpha^{\tau \to \nu}(y))) \bullet_l e' \\ &= v \bullet_l e' \end{aligned}$$

where the second line follows from the continuity of $\alpha^\nu$ and $\bullet_v$, and the third line follows from the definition of $\alpha^{\tau \to \nu}$.

This completes the induction step and hence the proof. ■

Theorem 3.10 follows directly from Part (1) of the following lemma.

**Lemma 3.13** *For any $d \in \mathcal{V}^\sigma$ and $e \in \mathcal{L}^\sigma$,*

1. *$\beta^\sigma(\delta^\sigma \bullet_l e)\ R^\sigma\ (\delta^\sigma \bullet_l e)$; and*
2. *If $d\ R^\sigma\ (\delta^\sigma \bullet_l e)$, then $\alpha^\sigma(d) = (\delta^\sigma \bullet_l e)$.*

**Proof:** By induction on types. In the basis, Part (1) follows immediately since $R^\iota$ is the identity relation on $\mathcal{V}^\iota = \mathcal{L}^\iota$ and $\beta^\iota(\delta^\iota \bullet_l e) = (\delta^\iota \bullet_l e)$. For Part (2), suppose $d\ R^\iota\ (\delta^\iota \bullet_l e)$. By the definition of the relation, $d = (\delta^\iota \bullet_l e)$. Thus, by the definition of $\alpha^\iota$, $\alpha^\iota(d) = d = (\delta^\iota \bullet_l e)$ as desired.

Now consider the induction case for type $\sigma = (\tau \to \nu)$. There are two parts to verify.

1. The definition of $\beta^{\tau\to\nu}$ implies that $\beta^{\tau\to\nu}(f) = \bot$ iff $f = \bot$. Thus, $\beta^{\tau\to\nu}(\delta^{\tau\to\nu} \bullet_l e) = \bot$ iff $(\delta^{\tau\to\nu} \bullet_l e) = \bot$. Now suppose $(\delta^{\tau\to\nu} \bullet_l e) \neq \bot$ and $d' \; R^\tau \; e'$. If $d' = \bot$, then $e' = \bot$ and so

$$(\beta^{\tau\to\nu}(\delta^{\tau\to\nu} \bullet_l e)) \bullet_v d' = \bot \; R^\nu \; \bot = (\delta^{\tau\to\nu} \bullet_l e) \bullet_l e'$$

Suppose, on the other hand that $d' \neq \bot$; then $e' \neq \bot$. By Lemma 3.8, $d' \; R^\tau \; (\delta^\tau \bullet_l e')$. Therefore,

$$\begin{array}{rll} (\beta^{\tau\to\nu}(\delta^{\tau\to\nu} \bullet_l e)) \bullet_v d' & = & \beta^\nu((\delta^{\tau\to\nu} \bullet_l e) \bullet_l (\alpha^\tau(d'))) \\ & = & \beta^\nu((\delta^{\tau\to\nu} \bullet_l e) \bullet_l (\delta^\tau \bullet_l e')) \\ & = & \beta^\nu(\delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l (\delta^\tau \bullet_l e')))) \\ & = & \beta^\nu(\delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l e'))) \\ R^\nu & \delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l e')) & \\ R^\nu & (\delta^{\tau\to\nu} \bullet_l e) \bullet_l e' & \end{array}$$

where the second and fifth lines hold by induction, and the fourth line holds by Lemma 3.6.

2. Suppose $d \; R^\sigma \; (\delta^\sigma \bullet_l e)$. If $d = \bot$, then $\alpha^\sigma(d) = \bot = (\delta^\sigma \bullet_l e)$. If $d \neq \bot$, then $(\delta^{\tau\to\nu} \bullet_l e) \neq \bot$. To show that $\alpha^{\tau\to\nu}(d) = (\delta^{\tau\to\nu} \bullet_l e)$, we therefore only need to show that they agree when applied using $\bullet_l$. So consider any element $e' \in \mathcal{L}^\tau$. If $e' = \bot$, then

$$\alpha^{\tau\to\nu}(d) \bullet_l e' = \bot = (\delta^{\tau\to\nu} \bullet_l e) \bullet_l e'$$

Now suppose $e' \neq \bot$. By induction, $\beta^\tau(\delta^\tau \bullet_l e') \; R^\tau \; (\delta^\tau \bullet_l e')$ and so

$$d \bullet_v \beta^\tau(\delta^\tau \bullet_l e') \; R^\nu \; (\delta^{\tau\to\nu} \bullet_l e) \bullet_l (\delta^\tau \bullet_l e')$$

Therefore,

$$\begin{array}{rll} \alpha^{\tau\to\nu}(d) \bullet_l e' & = & \alpha^\nu(d \bullet_v \beta^\tau(\delta^\tau \bullet_l e')) \\ & = & (\delta^{\tau\to\nu} \bullet_l e) \bullet_l (\delta^\tau \bullet_l e') \\ & = & \delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l (\delta^\tau \bullet_l e'))) \\ & = & \delta^\nu \bullet_l (e \bullet_l (\delta^\tau \bullet_l e')) \\ & = & (\delta^{\tau\to\nu} \bullet_l e) \bullet_l e' \end{array}$$

where the first line holds by the definition of $\alpha^{\tau\to\nu}$, the second line holds from the fact above and the induction hypothesis, and the fourth line holds from Lemma 3.6.

This completes the induction step and hence the proof. ■

**Proof of Theorem 3.7, Part (2), ($\Rightarrow$):** Suppose $\overline{M} \not\sqsubseteq_{lazy} \overline{N}$. Then by the full abstraction theorem for $\mathcal{L}$,

$$\mathcal{L}[\![\overline{M}]\!]\rho' \not\sqsubseteq \mathcal{L}[\![\overline{N}]\!]\rho'$$

for some environment $\rho'$. Let $h_1 = \mathcal{L}[\![\overline{M}]\!]\rho'$ and $h_2 = \mathcal{L}[\![\overline{N}]\!]\rho'$. By the properties of lazy models, there is some sequence of arguments $e_1, \ldots, e_k$ (possibly the null sequence) such that either

1. $(h_1 \bullet_l e_1 \bullet_l \ldots \bullet_l e_k) \neq \bot$ and $(h_2 \bullet_l e_1 \bullet_l \ldots \bullet_l e_k) = \bot$; or

2. $(h_1 \bullet_l e_1 \bullet_l \ldots \bullet_l e_k) = m$ and $(h_2 \bullet_l e_1 \bullet_l \ldots \bullet_l e_k) = n$, and $m$ and $n$ are different natural numbers.

Let us consider only the first case, since the second case can be proven similarly. By Lemma 3.11, we may assume without loss of generality that for all variables $x^\sigma$, $\rho'(x^\sigma)$ is in the range of $\delta^\sigma$. By Theorem 3.9, $h_1$ and $h_2$ are in the range of $\delta$, and hence we may also assume without loss of generality that $e_i$ are in the range of the $\delta$'s (since $h_i = (\delta^\sigma \bullet_l h_i)$ forces its arguments to be in the range of the $\delta$'s). By Theorem 3.10, there are elements $d_i \in \mathcal{V}$ with $d_i \; R \; e_i$, and moreover, there is a $\mathcal{V}$-environment $\rho$ that is compatible with $\rho'$.

We will use these elements $d_i$ to distinguish $h'_1 = \mathcal{V}[\![M]\!]\rho$ from $h'_2 = \mathcal{V}[\![N]\!]\rho$. By the analog of Lemma 3.4 for the modified translation, $h'_i \; R \; h_i$. By the definition of the relations, it follows that $(h'_1 \bullet_v d_1 \bullet_v \ldots \bullet_v d_k) \neq \bot$ but $(h'_2 \bullet_v d_1 \bullet_v \ldots \bullet_v d_k) = \bot$. Thus, $h'_1 \not\sqsubseteq h'_2$, which by the full abstraction theorem for $\mathcal{V}$ implies that $M \not\sqsubseteq_{val} N$. This completes the proof. ■

# 4 Call-by-name to Call-by-value PCF

We might take the same kind of approach in translating call-by-name PCF to call-by-value PCF, and translate call-by-name $\lambda$-abstractions to call-by-value $\lambda$-abstractions. There are, however, a few technical obstacles to overcome, because evaluation of applications is different in the two languages. Consider, for instance, the PCF-terms $((\lambda x.\, 3)\ (\mu f.\, f))$ and 3. Under call-by-name, both terms reduce to 3; under call-by-value, however, the first diverges.

We therefore need a new idea to translate call-by-name to call-by-value PCF. We use the standard trick of *delaying* the evaluation of a term; under call-by-value, all $\lambda$-abstractions terminate, so delaying may be accomplished by wrapping a term in a dummy $\lambda$-abstraction. This guarantees that all terms—and hence all operands in applications—terminate, so that the call-by-value interpreter never diverges when evaluating an operand. For simplicity, dummy arguments will be of type $\iota$, although one could use dummy arguments of any type. Terms of type $\sigma$ are therefore translated to terms of type $\sigma'$, where

$$\begin{aligned} \iota' &= \iota \to \iota \\ (\tau \to \nu)' &= \iota \to \tau' \to \nu'. \end{aligned}$$

The full translation from call-by-name to call-by-value appears in Table 6. Again, we need retractions $\gamma^\sigma$—which force terms to be constant functions in their first argument—to make the translation fully abstract.

**Theorem 4.1** *The translation $M \mapsto \widehat{M}$ from call-by-name to call-by-value PCF is adequate and inequationally fully abstract. That is,*

1. *Adequacy: For any closed $M$ of type $\iota$, $M \Downarrow_n n$ iff $(\widehat{M}\ 3) \Downarrow_v n$; and*
2. *Inequational Full Abstraction: For any $M$ and $N$, $M \sqsubseteq_{name} N$ iff $\widehat{M} \sqsubseteq_{val} \widehat{N}$.*

The proof of this theorem uses the same methods as those outlined above: we build a logical relation from a fully abstract model of call-by-name PCF to the model $\mathcal{V}$, and show that it is surjective on the range of $\gamma$. The complete proof may be found in [27].

$$
\begin{array}{rcl}
\widehat{x^\sigma} & = & (\gamma^\sigma\ (\lambda z^\iota.\, x^{\sigma'}\ z)) \\
\widehat{k} & = & \lambda z^\iota.\, k \\
\widehat{\mathsf{succ}\, M} & = & \lambda z^\iota.\, \mathsf{succ}\ (\widehat{M}\ 3) \\
\widehat{\mathsf{pred}\, M} & = & \lambda z^\iota.\, \mathsf{pred}\ (\widehat{M}\ 3) \\
\widehat{\lambda x^\sigma.\, M} & = & \lambda z^\iota.\, \lambda x^{\sigma'}.\, \widehat{M} \\
\widehat{(M\ N)} & = & ((\widehat{M}\ 3)\ \widehat{N}) \\
\widehat{\mathsf{cond}\, M\ N\ P} & = & \lambda z^\iota.\, \mathsf{cond}\ (\widehat{M}\ 3)\ (\widehat{N}\ 3)\ (\widehat{P}\ 3) \\
\widehat{\mathsf{pcond}\, M\ N\ P} & = & \lambda z^\iota.\, \mathsf{pcond}\ (\widehat{M}\ 3)\ (\widehat{N}\ 3)\ (\widehat{P}\ 3) \\
\widehat{\mu x^\sigma.\, M} & = & \mu x^{\sigma'}.\, \widehat{M}
\end{array}
$$

$$
\begin{array}{rcl}
\gamma^\iota & = & \lambda x^{\iota'}.\, \lambda z^\iota.\, x\ 3 \\
\gamma^{\tau\to\nu} & = & \lambda x^{(\tau\to\nu)'}.\, \lambda z^\iota.\, \lambda y^{\tau'}.\, (\gamma^\nu\ (\lambda z^\iota.\, x\ 3\ (\gamma^\tau\ y)\ z))
\end{array}
$$

Table 6: Translation of call-by-name to call-by-value PCF. We always assume that $z^\iota$ is a fresh variable not appearing in the term to be translated.

# 5 Lazy to Call-by-value PCF

The same ideas may be adapted to building a translation from lazy to call-by-value PCF. Table 7 gives such a translation. Here, most of the clauses for terms are *identical* to the previous translation; the only exceptions are the definition of the retractions $\chi^\sigma$, the clauses for translating variables and applications, and the additional clause for translating conv. This translation also turns out to be adequate and fully abstract:

**Theorem 5.1** *The translation $M \mapsto \widehat{M}$ from lazy to call-by-value PCF is adequate and inequationally fully abstract. That is,*

1. *Adequacy: For any closed LPCF-term $M$, $M \Downarrow_l k$ iff $(\widehat{M}\ 3) \Downarrow_v k$, and $M \Downarrow_l$ iff $(\widehat{M}\ 3) \Downarrow_v$;*
2. *Inequational Full Abstraction: $M \sqsubseteq_{lazy} N$ iff $\widehat{M} \sqsubseteq_{val} \widehat{N}$.*

Again, the proof uses the same basic technique, constructing a logical relation from the model $\mathcal{L}$ to the model $\mathcal{V}$ that is surjective on the range of $\chi$. The complete proof may be found in [27].

# 6 Corollaries of Full Abstraction

There are a number of complexity-theoretic results, regarding the time required to prove observational approximations, that can be deduced from the full abstraction theorems. For instance, we can deduce a lower bound on the time required to prove call-by-value observational approximations of **pure** terms—those *not* involving numerals, successor, predecessor, recursion, or conditionals. To find this lower bound, first note that call-by-name observational approximations of pure terms coincides with $\beta\eta$-equality (see [6, 27] for the complete argument). Thus, since $\beta\eta$-equality of pure

$$
\begin{array}{rcl}
\widehat{x^\sigma} & = & (\chi^\sigma\ (\lambda z^\iota.\, x^{\sigma'}\ z)) \\
\widehat{k} & = & \lambda z^\iota.\, k \\
\widehat{\mathsf{succ}\, M} & = & \lambda z^\iota.\, \mathsf{succ}\ (\widehat{M}\ 3) \\
\widehat{\mathsf{pred}\, M} & = & \lambda z^\iota.\, \mathsf{pred}\ (\widehat{M}\ 3) \\
\widehat{\lambda x^\sigma.\, M} & = & \lambda z^\iota.\, \lambda x^{\sigma'}.\, \widehat{M} \\
\widehat{(M\ N)} & = & \lambda z^\iota.\, ((\widehat{M}\ 3)\ \widehat{N})\ z \\
\widehat{\mathsf{cond}\, M\ N\ P} & = & \lambda z^\iota.\, \mathsf{cond}\ (\widehat{M}\ 3)\ (\widehat{N}\ 3)\ (\widehat{P}\ 3) \\
\widehat{\mathsf{pcond}\, M\ N\ P} & = & \lambda z^\iota.\, \mathsf{pcond}\ (\widehat{M}\ 3)\ (\widehat{N}\ 3)\ (\widehat{P}\ 3) \\
\widehat{\mu x^\sigma.\, M} & = & \mu x^{\sigma'}.\, \widehat{M} \\
\widehat{\mathsf{conv}\, M\ N} & = & \lambda z^\iota.\, (\lambda w.\, \widehat{N})\ (\widehat{M}\ 3)
\end{array}
$$

$$
\begin{array}{rcl}
\chi^\iota & = & \lambda x^{\iota'}.\, \lambda z^\iota.\, x\ 3 \\
\chi^{\tau\to\nu} & = & \lambda x^{(\tau\to\nu)'}.\, \lambda z^\iota.\, (\lambda w.\, \lambda y^{\tau'}.\, (\chi^\nu\ (\lambda z^\iota.\, x\ 3\ (\chi^\tau\ y)\ z)))\ (x\ 3)
\end{array}
$$

Table 7: Translation of lazy PCF to call-by-value PCF. As before, $z^\iota$ is a fresh variable not appearing in the term to be translated.

terms cannot be solved in elementary recursive time [36], testing to see whether $M \sqsubseteq_{name} N$ for pure $M$ and $N$ cannot be solved in elementary recursive time either.[5] Since the translation from call-by-name to call-by-value PCF works in linear time,

**Corollary 6.1** *The following question cannot be decided in elementary recursive time: given two pure PCF-terms $P$ and $Q$, is it the case that $P \sqsubseteq_{val} Q$?*

**Proof:** Suppose $P \sqsubseteq_{val} Q$ can be decided in elementary recursive time. Then one may decide whether $M \sqsubseteq_{name} N$ for pure terms: first translate and check whether $\widehat{M} \sqsubseteq_{val} \widehat{N}$. The result of this procedure is correct, since $\widehat{M} \sqsubseteq_{val} \widehat{N}$ iff $M \sqsubseteq_{name} N$. This would give a procedure that runs in elementary recursive time for determining whether $M \sqsubseteq_{name} N$, which is a contradiction. Thus, $P \sqsubseteq_{val} Q$ cannot be decided in elementary recursive time. ■

This corollary implies that deciding $M \sqsubseteq_{val} N$ requires time beyond that expressed by any fixed, finite stack of 2's.

Along similar lines, one can show that the problem of deciding $M \sqsubseteq_{lazy} N$ for pure conv-terms (those containing only the construct conv) cannot be decided in elementary recursive time. In fact, the decision problems $M \sqsubseteq_{lazy} N$ for pure conv-terms, and $M \sqsubseteq_{val} N$ for pure terms, are equivalent under polynomial-time reducibility: this follows immediately from the fact that there

[5] Non-elementary recursive time implies that a problem cannot be decided in time

$$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$$

for any bounded height of exponents [29].

are linear time reductions—via the translations—between these two problems. We conjecture the following upper bound:

**Conjecture 6.2** *The decision problem $M \sqsubseteq_{val} N$ for pure $M$ and $N$ can be solved in iterated exponential time (i.e., within time determined by some stack of 2's, where the height is determined by the size of the term). Thus, the problem of deciding $M \sqsubseteq_{lazy} N$ for $M$ and $N$ pure* conv*-terms can also be solved in iterated exponential time.*

It is already known that the problem of $M \sqsubseteq_{name} N$ for pure $M$ and $N$ can be decided in iterated exponential time [29, 36].

# 7 Functional Translations

In the introduction, we argued that fully abstract translations could provide the basis of an expressiveness theory. Nevertheless, there are trivial solutions to the problem of finding fully abstract translations between languages. This section considers such a trivial translation based on gödelnumbering, and then attempts to build an expressiveness theory by placing conditions on translations.

## 7.1 Gödelnumbering translations

It is easy to design a fully abstract translation between any two programming languages. For instance, if the target language contains numerals and all numerals are observationally distinct, one could simply translate all terms in an observational congruence class to a unique numeral in the target language. This translation preserves observational congruences and non-congruences. Nevertheless, we would not consider it a reasonable translation, since it is not effective. But even the condition of effectiveness is not sufficiently strong to rule out unreasonable translations. Consider the case of translating lazy PCF into call-by-name PCF.

**Theorem 7.1** *There exists an effective translation $M \mapsto \widetilde{M}$ of lazy to call-by-name PCF that is equationally fully abstract, i.e., $M \equiv_{lazy} N \iff \widetilde{M} \equiv_{name} \widetilde{N}$.*

**Proof:** (Sketch) We translate an LPCF-term $M$ to $(I\ \#M)$, for some gödelnumbering $\#$ of LPCF-terms. The closed term $I : \iota \to \iota \to \iota$ represents a "two-argument interpreter" for lazy PCF written in call-by-name PCF, where the first argument is the term to interpret and the second argument is a gödelnumbered tuple of arguments to $M$ (possibly an empty tuple). It is not hard to design such an interpreter meeting the following requirements:

1. $(I\ \#M\ \langle n_1, \ldots, n_m \rangle) \Uparrow_n$ if any of $n_1, \ldots, n_m$ is not the gödelnumber of a closed term;
2. $(I\ \#M\ \langle \#N_1, \ldots, \#N_m \rangle) \Uparrow_n$ if the lazy term $(M\ N_1 \ldots N_m)$ is not well-typed;
3. $(I\ \#M\ \langle \#N_1, \ldots, \#N_m \rangle) \Downarrow_n$ iff $(M\ N_1 \ldots N_m) \Downarrow_l$; and
4. $(I\ \#M\ \langle \#N_1, \ldots, \#N_m \rangle) \Downarrow_n k$ iff $(M\ N_1 \ldots N_m) \Downarrow_l k$.

To verify that the translation preserves observational congruences, suppose $M \not\equiv_{lazy} N$ with $M$ and $N$ having type $(\sigma_1 \to \ldots \to \sigma_n \to \iota)$. By the proof of the full abstraction theorem for lazy PCF (Theorem 2.4), there are terms $P_1, \ldots, P_m$ such that either

1. $(M\ P_1 \ldots P_m) \Downarrow_l$ and $(N\ P_1 \ldots P_m) \Uparrow_l$; or
2. $(M\ P_1 \ldots P_m) \Downarrow_l k$ and $(N\ P_1 \ldots P_m) \Downarrow_l k'$, where $k \neq k'$ and $m = n$.

By the properties of $I$, $(\widetilde{M}\ \langle \#P_1, \ldots, \#P_m \rangle)$ has different behavior than $(\widetilde{N}\ \langle \#P_1, \ldots, \#P_m \rangle)$. Thus, $\widetilde{M} \not\equiv_{name} \widetilde{N}$. The converse follows similarly and is omitted. ■

Similar translations based on gödelnumbers can be found between almost all universal programming languages, *i.e.*, those languages that can represent all partial recursive functions. An expressiveness theory based on only full abstraction must therefore identify most languages.

## 7.2 Definition of functional translations

In order to build an *interesting* expressiveness theory, we must place more stringent conditions on translations. There have been other attempts to find suitable conditions on translations. In [13, 15], for example, Mitchell examines translations that are compositional and preserve observable behavior, and is able to prove that there are no compositional translations between certain languages. Others, including Felleisen [7] and Shapiro [31] have developed similar definitions based on compositionality.

Unfortunately, not all of the translations in this paper fit the definitions of Mitchell, Felleisen, and Shapiro. In particular, two of the translations—the translations from lazy and call-by-name to call-by-value PCF—produce terms that do not have the same observable behavior as source terms: one must first apply a "dummy" numeral argument to obtain an observable result. Other reasonable translations, *e.g.*, continuation-passing style (cps), also require applications at the end of translation in order to produce results [21]. Of course, we might extend these definitions so that a translation may place a term—generated from a source term in some compositional manner—into some uniform context. This would cover the case of translating from call-by-name to call-by-value. But this definition would also allow gödelnumbering translations, since one could explicitly compute the gödelnumber of a term in the target language (which can be defined compositionally) and then apply the interpreter function $I$ to the result.

The search for suitably restrictive syntactic conditions seems unclear and complicated. We therefore leave the search for *syntactic* conditions open, and instead look for *semantic* conditions. Since the proofs of full abstraction for all three translations above are similar semantically, we use the common structure in seeking suitable conditions on translations. For simplicity, we consider translations between a restricted class of functional languages:

**Definition 7.2** A **simply-typed functional language** $L$ is a set of terms and observations $\mathcal{O}$ in which every term is assigned a type in the grammar

$$\sigma ::= \iota \mid (\sigma \to \sigma)$$

and where the set of terms is closed under application, *i.e.*, $(M\ N)$ is a term of type $\nu$ whenever $M$ and $N$ are terms of types $(\tau \to \nu)$ and $\tau$ respectively. Also, for any terms $M : (\sigma \to \tau)$ and $N : (\tau \to \nu)$, there must exist an $L$-term $(N \circ M) : (\sigma \to \nu)$ such that for any $L$-term $P$ of type $\sigma$, $((N \circ M)\ P) \equiv^{\mathcal{O}}_L (N\ (M\ P))$. Finally, $L$ must be **operationally extensional** (*cf.* [3, 4]) with respect to its observational congruence relation, *i.e.*, $M \equiv^{\mathcal{O}}_L N$ iff for all terms $P_1, \ldots, P_k$, $(M\ P_1 \ldots\ P_k)$ yields the same observations as $(N\ P_1 \ldots\ P_k)$.

When we take the set of terms to be the *closed* terms, call-by-name, call-by-value, and lazy PCF are simply-typed functional languages. In order to obtain operational extensionality for call-by-value and lazy PCF, we need to observe both numerals *and* termination; nevertheless, observing both numerals and termination does not change the observational congruence relations for call-by-value and lazy PCF.

It is instructive to first consider the translation from call-by-value to lazy PCF. Under this translation, lazy versions are "functionally equivalent" to the original call-by-value terms, in the sense that translations of terms of type $\iota$ have the same values as the original terms, and translations of functionally-typed terms, when provided with strict arguments, return strict results. This tight correspondence between the source and target terms is captured by a logical relation. Logical relations will thus play a key role in the definition below.

Under the other two translations, the connection between source and target terms is not as clear: a translated term has a different type than its source term. Nevertheless, using a definable projection function $\phi$, we may recover some of the behavior of the source term. At ground type, $\phi^{\iota} : \iota' \to \iota$ is the function that applies a term of type $\iota'$ to a dummy argument (3 in our version of the translation) to obtain a numeric result. In fact, this projection function is **generic**, *viz.*, it does not matter which numeral we pick to apply to terms. Similarly, one may define call-by-value functions

$$\phi^{\tau\to\nu} : (\tau \to \nu)' \to (\tau' \to \nu')$$

that apply their argument to a dummy argument to obtain a function. Indeed, suitably-defined projection functions are a key feature of each of the translations: the projections for the translation from call-by-value to lazy are simply the *identity* functions.

Putting these ideas together, we arrive at the following definition, slightly modified from the definition appearing in [26]. To simplify the definition, we use the notation $L^{\sigma}$ to denote $L$-terms of type $\sigma$, and the notation $M \rightleftharpoons_{\mathcal{O}} N$ (read "$M$ mutually simulates $N$") to signify that $M$ and $N$ yield the same observations in $\mathcal{O}$ when evaluated ($M$ and $N$ may be in different languages).

**Definition 7.3** Let $L_1$ and $L_2$ be simply-typed functional languages with observations $\mathcal{O}$. Let $M \mapsto \widetilde{M}$ be a translation of $L_1^{\sigma}$ to $L_2^{\sigma'}$ (note that this means the translation must work uniformly on types). Then the translation is **functional** if there are $L_2$-definable projections

$$\phi^{\iota} : \iota' \to \iota$$
$$\phi^{\tau\to\nu} : (\tau \to \nu)' \to (\tau' \to \nu')$$

and relations $R^{\sigma} \subseteq L_1^{\sigma} \times L_2^{\sigma'}$ such that

F1 $(M\ R\ \widetilde{M})$.

F2 $R$ is a logical relation:

1. $M\ R^{\iota}\ N$ implies $M \rightleftharpoons_{\mathcal{O}} (\phi^{\iota}\ N)$; and
2. $M\ R^{\tau\to\nu}\ N$ implies $M \rightleftharpoons_{\mathcal{O}} (\phi^{\tau\to\nu}\ N)$, and $P\ R^{\tau}\ Q$ implies that $(M\ P)\ R^{\nu}\ (N \bullet Q)$, where $N \bullet Q = ((\phi^{\tau\to\nu}\ N)\ Q)$.

F3 Applications are translated uniformly: $(\widetilde{M\ N}) \equiv^{\mathcal{O}}_{L_2} ((\phi\ \widetilde{M})\ \widetilde{N})$.

F4 Projections $\phi$ are generic: For any $L_2$-term $N$ in the range of $R$ and any $L_2$-terms $Q_i$ of the appropriate type, $(\phi\ N) \equiv^{\mathcal{O}}_{L_2} (N\ Q_1 \ldots Q_n)$.

F5 Translated functions convert arguments to the range of $R$: For any $M$ in the range of $R^{\tau\to\nu}$ and $P$ of type $\tau'$, there exists a term $P'$ in the range of $R^{\tau}$ such that $(M \bullet P) \equiv^{\mathcal{O}}_{L_2} (M \bullet P')$.

F6 The target sublanguage is operationally extensional: Suppose $M$ and $N$ are in the range of $R^{\sigma}$, and for all $P_i$ in the range of $R$,

$$(\phi\,(M \bullet P_1 \bullet \ldots \bullet P_k)) \rightleftharpoons_{\mathcal{O}_1} (\phi\,(N \bullet P_1 \bullet \ldots \bullet P_k)).$$

Then $M \equiv^{\mathcal{O}}_{L_2} N$.

This definition should be compared to the definition of the relations $R$ given in Section 3.2. The final clause is necessary to achieve full abstraction: intuitively, it says that if two terms in the target of the translation are distinguishable operationally, there is a way of distinguishing them by terms in the target of the translation.

We begin by proving that all functional translations are fully abstract.

**Lemma 7.4** *Suppose $M \mapsto \widetilde{M}$ is a functional translation from $L_1^{\sigma}$ to $L_2^{\sigma'}$ with projections $\phi^{\sigma}$ and relations $R^{\sigma}$. Suppose further that $M\ R^{\sigma}\ P$ and $N\ R^{\sigma}\ Q$. Then $M \equiv^{\mathcal{O}}_{L_1} N$ iff $P \equiv^{\mathcal{O}}_{L_2} Q$.*

**Proof:** ($\Leftarrow$) Suppose $M \not\equiv^{\mathcal{O}}_{L_1} N$. Then by the operational extensionality of $L_1$, there exist terms $P_i$ with $(M\ P_1 \ldots P_k) \not\Rightarrow_{\mathcal{O}} (N\ P_1 \ldots P_k)$. By Clauses F1 and F2,

$$(\phi\,(P \bullet \widetilde{P_1} \bullet \ldots \bullet \widetilde{P_k})) \not\Rightarrow_{\mathcal{O}} (\phi\,(Q \bullet \widetilde{P_1} \bullet \ldots \bullet \widetilde{P_k})).$$

Thus, $P \not\equiv^{\mathcal{O}}_{L_2} Q$.

($\Rightarrow$) Suppose $P \not\equiv^{\mathcal{O}}_{L_2} Q$. Then by Clause F6, there exist $P_i$ in the range of $R$ such that $(\phi\,(P \bullet P_1 \bullet \ldots \bullet P_k)) \not\Rightarrow_{\mathcal{O}} (\phi\,(Q \bullet P_1 \bullet \ldots \bullet P_k))$. Now pick $P_i'$ such that $P_i'\ R\ P_i$ (these must exist). By Clause F2, $(M\ P_1' \ldots P_k') \not\Rightarrow_{\mathcal{O}} (N\ P_1' \ldots P_k')$. Thus, $M \not\equiv^{\mathcal{O}}_{L_1} N$. ■

**Theorem 7.5** *Let $L_1$ and $L_2$ be simply-typed functional languages. Suppose $M \mapsto \widetilde{M}$ is a functional translation from $L_1$ to $L_2$ with relations $R$. Then $M \mapsto \widetilde{M}$ is equationally fully abstract.*

**Proof:** Follows easily from Lemma 7.4 and the fact that $M\ R\ \widetilde{M}$. ■

In order to be a suitable basis for an expressiveness theory, functional translations should be closed under composition. This has an intuitive justification: if language A is no more expressive than B (*i.e.*, there is a functional translation from A to B), and B is no more expressive than C, then A should be no more expressive than C.

**Theorem 7.6** *Suppose there are functional translations $M \mapsto \overline{M}$ from $L_1^{\sigma}$ to $L_2^{\sigma'}$ and $M \mapsto \widetilde{M}$ from $L_2^{\sigma}$ to $L_3^{\sigma''}$, and $\mathcal{O}$ is the set of observations for each of the three languages. Then there is a functional translation from $L_1^{\sigma}$ to $L_3^{(\sigma')''}$.*

**Proof:** Let $R_i$ and $\phi_i^{\sigma}$ be the parameters of the two functional translations. Define

$$\begin{aligned}
R_3^{\sigma} &= R_2^{\sigma'} \circ R_1^{\sigma} \subseteq L_1^{\sigma} \times L_3^{(\sigma')''} \\
\phi_3^{\iota} &= \phi_2^{\iota} \circ (\phi_2^{\iota'\to\iota}\ \widetilde{\phi_1^{\iota}}) : (\iota')'' \to \iota \\
\phi_3^{\tau\to\nu} &= \phi_2^{\tau'\to\nu'} \circ (\phi_2^{(\tau\to\nu)'\to(\tau'\to\nu')}\ \widetilde{\phi_1^{\tau\to\nu}}) : ((\tau\to\nu)')'' \to (\tau')'' \to (\nu')''
\end{aligned}$$

The reader may check that these relations and terms have the advertised type. Let $\widehat{M} = \widetilde{\overline{M}}$; we must verify the requirements F1–F6 hold for this composite translation:

1. $M\ R_3\ \widehat{M}$: This is obvious, since $M\ R_1\ \overline{M}\ R_2\ \widetilde{\overline{M}}$.

2. $R_3$ is a logical relation: There are two requirements to verify—$R_3$-related terms produce the same observable behavior, and applying related terms to related arguments produces related results. For the first part, suppose that $M\ R_3^\sigma\ P$, *i.e.*, there exists an $N$ such that $M\ R_1^\sigma\ N\ R_2^{\sigma'}\ P$. By Clause F2, $M \rightleftharpoons_{\mathcal{O}} (\phi_1\ N)$ and $(\phi_1\ N) \rightleftharpoons_{\mathcal{O}} (\phi_2\ (\widetilde{\phi_1} \bullet_2 P)) \equiv^{\mathcal{O}}_{L_3} (\phi_3\ P)$, where $L \bullet_2 Q = ((\phi_2\ L)\ Q)$. Thus, $M \rightleftharpoons_{\mathcal{O}} (\phi_3\ P)$ as desired.

   Now suppose $\sigma = (\tau \to \nu)$, and there exists an $N_0$ such that $M_0\ R_1^\sigma\ N_0\ R_2^{\sigma'}\ P_0$. Suppose further that $M_1\ R_1^\tau\ N_1\ R_2^{\tau'}\ P_1$. By Clause F2,

$$(M_0\ M_1)\ R_1\ ((\phi_1\ N_0)\ N_1)\ R_2\ ((\widetilde{\phi_1} \bullet_2 P_0) \bullet_2 P_1).$$

   However, by the definition of $\phi_3$, $((\widetilde{\phi_1} \bullet_2 P_0) \bullet_2 P_1) \equiv^{\mathcal{O}}_{L_3} ((\phi_3\ P_0)\ P_1)$, so by Clause F2 we may conclude

$$(M_0\ M_1)\ R_1\ ((\phi_1\ N_0)\ N_1)\ R_2\ ((\phi_3\ P_0)\ P_1).$$

   Thus, $(M_0\ M_1)\ R_3\ (P_0 \bullet_3 P_1)$, where $(P_0 \bullet_3 P_1) = ((\phi_3\ P_0)\ P_1)$, as desired.

3. $(\widehat{M\ N}) \equiv^{\mathcal{O}}_{L_3} ((\phi_3\ \widehat{M})\ \widehat{N})$: To make the notation a bit easier to read, define $\mathcal{F}(M) = \widetilde{M}$. Then

$$\begin{aligned}
((\phi_3\ \widehat{M})\ \widehat{N}) &\equiv^{\mathcal{O}}_{L_3} ((\widetilde{\phi_1} \bullet_2 \widehat{M}) \bullet_2 \widehat{N}) \\
&\equiv^{\mathcal{O}}_{L_3} ((\widetilde{\phi_1} \bullet_2 \mathcal{F}(\overline{M})) \bullet_2 \widehat{N}) \\
&\equiv^{\mathcal{O}}_{L_3} ((\mathcal{F}(\phi_1\ \overline{M})) \bullet_2 \widehat{N}) \\
&\equiv^{\mathcal{O}}_{L_3} ((\mathcal{F}(\phi_1\ \overline{M})) \bullet_2 \mathcal{F}(\overline{N})) \\
&\equiv^{\mathcal{O}}_{L_3} \mathcal{F}((\phi_1\ \overline{M})\ \overline{N}) \\
&\equiv^{\mathcal{O}}_{L_3} \mathcal{F}(\overline{M\ N}) \equiv^{\mathcal{O}}_{L_3} (\widehat{M\ N})
\end{aligned}$$

   where the first line follows from the definition of $\phi_3$, and the third, fifth, and sixth lines follow from Clause F3 of the definition of functional translation.

4. $\phi_3$ is generic: Suppose $P$ is in the range of $R_3$. Then there are terms $M$ and $N$ with $(M\ R_1\ N\ R_2\ P)$. By Clause F1, we know that $N\ R_2\ \widetilde{N}$. By Lemma 7.4, $P \equiv^{\mathcal{O}}_{L_3} \widetilde{N}$. Thus, for any $L_3$-terms $P_i$ and $Q_i$ and $L_2$-terms $S_j$ of the appropriate types,

$$\begin{aligned}
(\phi_3\ P) &\equiv^{\mathcal{O}}_{L_3} (\phi_3\ \widetilde{N}) \\
&\equiv^{\mathcal{O}}_{L_3} (\phi_2\ (\widetilde{\phi_1} \bullet_2 \widetilde{N})) \\
&\equiv^{\mathcal{O}}_{L_3} (\phi_2\ \mathcal{F}(\phi_1\ N)) \\
&\equiv^{\mathcal{O}}_{L_3} (\mathcal{F}(\phi_1\ N)\ Q_1 \ldots Q_k) \\
&\equiv^{\mathcal{O}}_{L_3} (\mathcal{F}(N\ S_1 \ldots S_l)\ Q_1 \ldots Q_k) \\
&\equiv^{\mathcal{O}}_{L_3} ((\widetilde{N} \bullet_2 \widetilde{S_1} \bullet_2 \ldots \bullet_2 \widetilde{S_l})\ Q_1 \ldots Q_k) \\
&\equiv^{\mathcal{O}}_{L_3} (\widetilde{N}\ \vec{P_1}\ \widetilde{S_1} \ldots \vec{P_l}\ \widetilde{S_l}\ Q_1 \ldots Q_k)
\end{aligned}$$

   where the second line follows from the definition of $\phi_3$, the third and sixth lines follow from Clause F3, and the fourth, fifth, and seventh lines follow from Clause F4. This is now almost

in the form we want—except that some of the arguments (namely $\widetilde{S_i}$) are in the range of one of the translations. So consider any $L_3$-terms $S_i'$. By Clause F5, there exists an $S_i''$ in the range of $R_2$ such that $(\widetilde{N} \bullet_2 S_1' \bullet_2 \ldots \bullet_2 S_l') \equiv^{\mathcal{O}}_{L_3} (\widetilde{N} \bullet_2 S_1'' \bullet_2 \ldots \bullet_2 S_l'')$. Since $S_i''$ is in the range of $R_2$, there exists $S_i'''$ $R_2$ $S_i''$. Note by Clause F1 and Lemma 7.4, $S_i'' \equiv^{\mathcal{O}}_{L_3} \widetilde{S_i'''}$. Thus, we may assume $S_i''$—and hence $S_i'$—are in the range of the translation $\widetilde{(\cdot)}$. Therefore, it is enough to consider only those arguments in the range of the translation, so it follows that $(\phi_3\ P) \equiv^{\mathcal{O}}_{L_3} (P\ P_1\ \ldots\ P_m)$ for any terms $P_j$ of the appropriate type.

5. Translated functions convert arguments to be in the range of $R_3$: Suppose $P$ is in the range of $R_3$, *i.e.*, $(M\ R_1\ N\ R_2\ P)$. Note that by Lemma 7.4, $P \equiv^{\mathcal{O}}_{L_3} \widetilde{\overline{M}}$. Pick any term $T$ such that $(P \bullet_3 T)$ is well-typed. By the definition of $\phi_3$,

$$(P \bullet_3 T) \equiv^{\mathcal{O}}_{L_3} ((\widetilde{\phi_1} \bullet_2 P) \bullet_2 T) \tag{1}$$

Since $(\widetilde{\phi_1} \bullet_2 P)$ is in the range of $R_2$, by Clause F5 there exists a $T_0$ in the range of $R_2$ such that

$$((\widetilde{\phi_1} \bullet_2 P) \bullet_2 T) \equiv^{\mathcal{O}}_{L_3} ((\widetilde{\phi_1} \bullet_2 P) \bullet_2 T_0) \tag{2}$$

Pick any $S\ R_2\ T_0$ (we know such an $S$ exists since $T_0$ is in the range of $R_2$). Since $S\ R_2\ \widetilde{S}$, by Lemma 7.4, $\widetilde{S} \equiv^{\mathcal{O}}_{L_3} T_0$. Therefore,

$$\begin{aligned}
(\widetilde{\phi_1} \bullet_2 P \bullet_2 T_0) &\equiv^{\mathcal{O}}_{L_3} (\widetilde{\phi_1} \bullet_2 \widetilde{\overline{M}} \bullet_2 T_0) && (3)\\
&\equiv^{\mathcal{O}}_{L_3} (\widetilde{\phi_1} \bullet_2 \widetilde{\overline{M}} \bullet_2 \widetilde{S}) && (4)\\
&\equiv^{\mathcal{O}}_{L_3} \mathcal{F}((\phi_1\ \overline{M})\ S) && (5)
\end{aligned}$$

where the last line follows from Clause F3. Now by Clause F5, there is an $S_0$ in the range of $R_1$ such that $((\phi_1\ \overline{M})\ S) \equiv^{\mathcal{O}}_{L_2} ((\phi_1\ \overline{M})\ S_0)$. Pick $Q$ such that $Q\ R_1\ S_0$; then by Lemma 7.4, $\overline{Q} \equiv^{\mathcal{O}}_{L_2} S_0$. Thus,

$$((\phi_1\ \overline{M})\ S_0) \equiv^{\mathcal{O}}_{L_2} ((\phi_1\ \overline{M})\ \overline{Q}) \equiv^{\mathcal{O}}_{L_2} (\overline{M\ Q})$$

where the last observational congruence follows from Clause F3. Thus, since $\widetilde{(\cdot)}$ is fully abstract by Theorem 7.5,

$$\mathcal{F}((\phi_1\ \overline{M})\ S) \equiv^{\mathcal{O}}_{L_3} \mathcal{F}((\phi_1\ \overline{M})\ S_0) \equiv^{\mathcal{O}}_{L_3} \mathcal{F}((\phi_1\ \overline{M})\ \overline{Q}) \equiv^{\mathcal{O}}_{L_3} (\widetilde{\phi_1} \bullet_2 \widetilde{\overline{M}} \bullet_2 \widetilde{\overline{Q}}) \tag{6}$$

Putting together Equations 1–6, we arrive at the fact that

$$(P \bullet_3 T) \equiv^{\mathcal{O}}_{L_3} (\widetilde{\phi_1} \bullet_2 \widetilde{\overline{M}} \bullet_2 \widetilde{\overline{Q}}) \equiv^{\mathcal{O}}_{L_3} (P \bullet_3 \widetilde{\overline{Q}}).$$

Since $\widetilde{\overline{Q}}$ is in the range of $R_3$, we are done.

6. Operational extensionality: Suppose $M_i\ R_1^{\sigma}\ N_i\ R_2^{\sigma'}\ P_i$ and $P_0 \not\equiv^{\mathcal{O}}_{L_3} P_1$. By Lemma 7.4, $N_0 \not\equiv^{\mathcal{O}}_{L_2} N_1$ and hence $M_0 \not\equiv^{\mathcal{O}}_{L_1} M_1$. Since $L_1$ is operationally extensional, there exist $Q_i$ with $(M_0\ Q_1 \ldots\ Q_l) \not\equiv_{\mathcal{O}} (M_1\ Q_1 \ldots\ Q_l)$. Thus,

$$(\phi_1\ (N_0 \bullet_1 \overline{Q_1} \bullet_1 \ldots \bullet_1 \overline{Q_l})) \not\equiv_{\mathcal{O}} (\phi_1\ (N_1 \bullet_1 \overline{Q_1} \bullet_1 \ldots \bullet_1 \overline{Q_l}))$$

where $S \bullet_1 S' = ((\phi_1\ S)\ S')$. Note that by Clause F2,

$$(\phi_1\ (N_i \bullet_1 \overline{Q_1})) \rightleftharpoons_{\mathcal{O}} (\phi_2\ (\widetilde{\phi_1} \bullet_2 ((\widetilde{\phi_1} \bullet_2 P_i) \bullet_2 \widehat{Q_1}))) \equiv^{\mathcal{O}}_{L_3} (\phi_3\ (P_i \bullet_3 \widehat{Q_1}))$$

In general,

$$(\phi_1\ (N_i \bullet_1 \overline{Q_1} \bullet_1 \ldots \bullet_1 \overline{Q_l})) \rightleftharpoons_{\mathcal{O}} (\phi_3\ (P_i \bullet_3 \widehat{Q_1} \bullet_3 \ldots \bullet_3 \widehat{Q_l}))$$

Thus,

$$(\phi_3\ (P_0 \bullet_3 \widehat{Q_1} \bullet_3 \ldots \bullet_3 \widehat{Q_l})) \not\rightleftharpoons_{\mathcal{O}} (\phi_3\ (P_1 \bullet_3 \widehat{Q_1} \bullet_3 \ldots \bullet_3 \widehat{Q_l}))$$

and Clause F6 now follows from the fact that $\widehat{Q_i}$ are in the range of $R_3$.

This completes the verification of each part and hence the proof. ■

## 7.3 Distinctions made by functional translations

The translations of Sections 3 and 5 demonstrate that call-by-value and lazy PCF are "equivalent" under the notion of functional translation: each can indeed be seen to be functional, when the observations of the two languages are chosen to be numerals *and* termination. Call-by-name PCF can also be functionally translated into call-by-value—and by the Theorem 7.6, into lazy PCF as well—as long as specify what "termination" means in call-by-name PCF. Here, the correct choice is to say that *all* terms of higher-type terminate under the call-by-name semantics; choosing this as our meaning of termination does not change the observational approximation relation $\sqsubseteq_{name}$, even though the call-by-name interpreter given above does not really terminate on all terms of higher-type.

Nevertheless, call-by-name PCF is strictly *less* expressive (under the notion of functional translations) than either call-by-value or lazy PCF. For definiteness, we prove that call-by-name cannot be translated to call-by-value.

**Theorem 7.7** *There is no functional translation from call-by-value to call-by-name PCF.*

**Proof:** Suppose $M \mapsto \widetilde{M}$ is a functional translation with projections $\phi^\sigma$ and relations $R^\sigma$. Let $\Omega_1$ and $\Omega_2$ be divergent call-by-value PCF terms of types $(\iota \to \iota)$ and $\iota$ respectively. Note that $\Omega_1 \not\equiv_{val} \lambda x.\, \Omega_2$. Thus, by Theorem 7.5,

$$\widetilde{\Omega_1} \not\equiv_{name} \lambda\widetilde{x.}\, \widetilde{\Omega_2}.$$

However, by the definition of functional translation, $(\phi\ ((\phi^{\iota\to\iota}\ \lambda\widetilde{x.}\, \widetilde{\Omega_2})\ N))$ for any closed $N$ diverges. Similarly, $(\phi\ ((\phi^{\iota\to\iota}\ \widetilde{\Omega_1})\ N))$ diverges. By Clause F4 of the definition of functional translation,

$$(\phi\ ((\phi^{\iota\to\iota}\ \lambda\widetilde{x.}\, \widetilde{\Omega_2})\ N)) \equiv_{name} ((\lambda\widetilde{x.}\, \widetilde{\Omega_2})\ \vec{P}\ N\ \vec{Q})$$

for any terms $P_i$ and $Q_i$. Similarly,

$$(\phi\ ((\phi^{\iota\to\iota}\ \widetilde{\Omega_1})\ N)) \equiv_{name} (\widetilde{\Omega_1}\ \vec{P}\ N\ \vec{Q}).$$

Therefore, since both $\lambda\widetilde{x.}\, \widetilde{\Omega_2}$ and $\widetilde{\Omega_1}$ diverge when applied to any arguments, both are call-by-name observationally congruent to $\Omega$. Thus,

$$\widetilde{\Omega_1} \equiv_{name} \lambda\widetilde{x.}\, \widetilde{\Omega_2}$$

This is a contradiction, so there can be no functional translation from call-by-value to call-by-name PCF. ■

# 8 Conclusion

Letting $L_1 \leq L_2$ denote the proposition that there is a functional translation from $L_1$ to $L_2$, and $L_1 \sim L_2$ denote $L_1 \leq L_2$ and $L_2 \leq L_1$, the main results of the paper may be summarized in symbols as follows:

$$\text{Call-by-name PCF} < \text{Call-by-value PCF} \sim \text{Lazy PCF}$$

It seems quite likely that other fully abstract translations exist between other functional languages. Indeed, although we have not proven it here, there is a well-structured translation from the untyped call-by-value $\lambda$-calculus to the untyped lazy $\lambda$-calculus. This translation uses a fairly natural modification of the retractions in the call-by-value to lazy case. The proof relies on two models: the fully abstract model for the untyped lazy $\lambda$-calculus [1, 19, 20], and the fully abstract model for the untyped call-by-value $\lambda$-calculus composed of lifted, strict continuous functions (Felleisen and Sitaram, personal communication). Instead of logical relations, we use inclusive predicates. This example should provide clues for adding general recursive types, since untyped languages are essentially languages with one recursive type; it should also provide clues for extending the language with sums and products.

All three of the languages considered here incorporate parallel conditional. Of course, we would like sequential fully abstract translations as well, *e.g.*, from sequential call-by-value PCF to sequential lazy PCF. We believe our methods will carry over to this problem, albeit carried out directly on the language instead of through the use of models. Extending the languages with richer type structures or other features, such as those captured by monads [17, 18], would also be interesting.

We have only briefly discussed how the notion of functional translations leads to a definition of expressiveness. Proving other algebraic properties beyond composition for functional translations would be a good start. Also, the definition of functional translation may, on further insight, be too restrictive. In particular, Clause F4, which posits that the projections functions behave generically, seems very restrictive. It may well be that a less restrictive definition would still rule out gödelnumbering translations. We leave this question open as well.

# Acknowledgments

I especially thank Albert Meyer for the suggestion of this problem and many productive conversations. I also thank Samson Abramsky, Val Breazu-Tannen, Stavros Cosmadakis, Matthias Felleisen, Carl Gunter, Eugenio Moggi, and Gordon Plotkin for helpful discussions, and Michael Ernst, Lalita Jategaonkar, Trevor Jim, Arthur Lent, Ramesh Subrahmanyam, and David Wald for comments on drafts of this paper.

# References

[1] Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison-Wesley, 1990.

[2] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic*. North-Holland, 1981. Revised Edition, 1984.

[3] Bard Bloom. Can LCF be topped? In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 282–295. IEEE, 1988.

[4] Bard Bloom. Can LCF be topped? Flat lattice models of typed $\lambda$-calculus. *Information and Computation*, 87:264–301, 1990.

[5] Bard Bloom and Jon G. Riecke. LCF should be lifted. In Teodor Rus, editor, *Proc. Conf. Algebraic Methodology and Software Technology*, pages 133–136. Department of Computer Science, University of Iowa, 1989.

[6] Stavros S. Cosmadakis, Albert R. Meyer, and Jon G. Riecke. Completeness for typed lazy inequalities (preliminary report). In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 312–320, 1990.

[7] Matthias Felleisen. On the expressive power of programming languages. In *Proc. of European Symp. on Programming*, volume 432 of *Lect. Notes in Computer Sci.*, pages 134–151. Springer-Verlag, 1990. Expanded version to appear in *Science of Computer Programming.*

[8] Harvey Friedman. Equality between functionals. In Rohit Parikh, editor, *Logic Colloquium '73*, volume 453 of *Lect. Notes in Math.*, pages 22–37. Springer-Verlag, 1975.

[9] Carl A. Gunter. Structures and techniques for the semantics of programming languages. Unpublished manuscript, University of Pennsylvania, January 1991.

[10] Carl A. Gunter and Dana S. Scott. Semantic domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 633–674. Elsevier, 1990.

[11] Albert R. Meyer. Semantical paradigms: Notes for an invited lecture, with two appendices by Stavros S. Cosmadakis. In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 236–255. IEEE, 1988.

[12] Robin Milner. Fully abstract models of the typed lambda calculus. *Theoretical Computer Sci.*, 4:1–22, 1977.

[13] John C. Mitchell. Lisp is not universal (summary). Unpublished manuscript, AT&T Bell Laboratories, August 1986.

[14] John C. Mitchell. Type systems for programming languages. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 365–458. Elsevier, 1990.

[15] John C. Mitchell. On abstraction and the expressive power of programming languages. In *Theoretical Aspects of Computer Software, Lect. Notes in Computer Sci.*, 1991. To appear.

[16] Eugenio Moggi. *The Partial Lambda Calculus.* PhD thesis, University of Edinburgh, 1988.

[17] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 14–23. IEEE, 1989.

[18] Eugenio Moggi. Notions of computation and monads. *Information and Control*, 93:55–92, 1991.

[19] Chih-Hao Luke Ong. Fully abstract models of the lazy lambda calculus. In *29th Annual Symposium on Foundations of Computer Science*, pages 368–376. IEEE, 1988.

[20] Chih-Hao Luke Ong. *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming.* PhD thesis, Imperial College, University of London, 1988.

[21] Gordon D. Plotkin. Call-by-name, call-by-value and the $\lambda$-calculus. *Theoretical Computer Sci.*, 1:125–159, 1975.

[22] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Sci.*, 5:223–257, 1977.

[23] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus Univ., Computer Science Dept., Denmark, 1981.

[24] Gordon D. Plotkin. Notes on completeness of the full continuous type hierarchy. Unpublished manuscript, Massachusetts Institute of Technology, November 1982.

[25] Jon G. Riecke. A complete and decidable proof system for call-by-value equalities (preliminary report). In M. S. Paterson, editor, *Automata, Languages and Programming: 17th International Colloquium*, volume 443 of *Lect. Notes in Computer Sci.*, pages 20–31. Springer-Verlag, 1990.

[26] Jon G. Riecke. Fully abstract translations between functional languages (preliminary report). In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 245–254. ACM, 1991.

[27] Jon G. Riecke. *The Logic and Expressibility of Simply-Typed Call-by-Value and Lazy Languages.* PhD thesis, Massachusetts Institute of Technology, 1991.

[28] V.Yu. Sazonov. Expressibility of functions in D. Scott's LCF language. *Algebra i Logika*, 15:308–330, 1976. Russian.

[29] Helmut Schwichtenberg. Complexity of normalization in the pure typed lambda-calculus. In A.S. Troelstra and D. van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, pages 453–457. North Holland, 1982.

[30] Dana Scott. A type theoretical alternative to CUCH, ISWIM, OWHY. Unpublished manuscript, Oxford University, 1969.

[31] Ehud Shapiro. Separating concurrent languages with categories of language embeddings. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, 1991.

[32] Kurt Sieber. Message to types@theory.lcs.mit.edu electronic mail forum, June 1989.

[33] Kurt Sieber. Relating full abstraction results for different programming languages. In *Foundations of Software Technology and Theoretical Computer Science, Bangalore, India*, December 1990.

[34] Dorai Sitaram and Matthias Felleisen. Reasoning with continuations II: Full abstraction for models of control. In *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 161–175. ACM, 1990.

[35] Dorai Sitaram and Matthias Felleisen. Modeling continuations without continuations. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 185–196. ACM, 1991.

[36] Richard Statman. The typed $\lambda$-calculus is not elementary recursive. *Theoretical Computer Sci.*, 9:73–81, 1979.

[37] Richard Statman. Logical relations in the typed $\lambda$-calculus. *Information and Control*, 65:86–97, 1985.

[38] Alan Stoughton. *Fully Abstract Models of Progamming Languages.* Research Notes in Theoretical Computer Science. Pitman/Wiley, 1988. Revision of Ph.D thesis, Dept. of Computer Science, Univ. Edinburgh, Report No. CST-40-86, 1986.

[39] Bent Thomsen. A calculus of higher order communicating systems. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*, pages 143–154. ACM, 1989.