J. Functional Programming **10** (2): 167–190, March 2000. Printed in the United Kingdom
© 2000 Cambridge University Press

Program equivalence in a linear functional language

G. M. BIERMAN
Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK

Abstract

Researchers have recently proposed that for certain applications it is advantageous to use functional languages whose type systems are based upon linear logic: so-called *linear* functional languages. In this paper we develop reasoning techniques for programs in a linear functional language, linPCF, based on their *operational* behaviour. The principal theorem of this paper is to show that contextual equivalence of linPCF programs can be characterised coinductively. This characterisation provides a tractable method for reasoning about contextual equivalence, and is used in three ways:

- A number of useful contextual equivalences between linPCF programs is given.
- A notion of type isomorphism with respect to contextual equivalence, called operational isomorphism, is given. In particular the types $!\phi \otimes !\psi$ and $!(\phi \& \psi)$ are proved to be operationally isomorphic.
- A translation of non-strict PCF into linPCF is shown to be adequate, but not fully abstract, with respect to contextual equivalence.

Capsule Review

This paper fills in many of the gaps in the technology for reasoning operationally about a simple functional language (linPCF) having a type system based on linear logic. In particular, the paper provides a methodology based on bisimulation for showing that two linPCF terms are contextually equivalent. Numerous examples of contextual equivalences are given. The methodology is also useful for reasoning about the relationship between types, and for reasoning about translations between such a language and a language having a type system based on intuitionistic logic.

Recently, there has been increasing interest in exploiting type information throughout a compiler. The resource-conscious nature of linear logic could make it a useful basis for a type system to be used in such a compiler. In this setting, a methodology for reasoning about contextual equivalence is essential for proving the correctness of compiler optimizations.

1 Introduction

Since its inception, Girard's linear logic (Girard, 1987) has promised to give a refined view of computation due to its resource-conscious nature. One possibility is to consider a functional language whose type system is based upon (intuitionistic) linear logic – a linear functional language. Work by Holmström (1988; 1989), Mackie (1994), Wadler (1990; 1991), Wakeling (1990; 1991) and, more recently, Barendsen and Smetsers (1996) have suggested that there are good pragmatic reasons for functional programmers to move to a *linear* type system. However programmers need reasoning principles for their linear functional programs. Surprisingly there is comparatively little work in this area – most attention having been paid to reasoning via a denotational model (Bierman, 1995; Braüner, 1997). The approach taken in this paper is to use techniques based upon the *operational* behaviour of programs. One advantage of such operationally based techniques is that they require relatively little mathematical overhead. (Further arguments in favour of operationally based techniques can be found in the literature (e.g. Gordon and Pitts, 1998).

There is another potential use of a linear type system: inside a compiler. It is now becoming accepted that intermediate languages in functional compilers should be typed, preferably with a rich type system. A number of recent compilers have taken this approach, e.g. TIL (Morrisett, 1995) and MLj (Benton *et al*., 1998). It is known that a linear type system allows for elegant translations of both strict and non-strict programming languages into a linear language (Maraist *et al*., 1995). Thus, there is good reason to study a language which could serve as a *linear* intermediate language. To this end, we shall consider how a non-strict functional language can be translated into such a linear intermediate language and use the reasoning principles developed in this paper to study the translation.

This paper is organised as follows. In section 2 we give the syntax and operational semantics for a prototypical linear functional language: linPCF. One complication of the linear type system is the definition of a context – a program with possibly many holes in it. In section 3.1 we show how standard definitions are inadequate before offering a solution. Given this definition, we define in section 3.2 a Morris-style notion of contextual equivalence and an alternative notion of program equivalence known as applicative (bi)similarity. Employing by now fairly standard methods (the details are given in the appendix), we show that applicative bisimilarity coincides with contextual equivalence. Thus, to show that two programs are contextually equivalent, it is sufficient to show that they are bisimilar. We shall use this fact in three ways. First, in section 4.1 this fact is used to give a number of examples of contextually equivalent programs. Secondly, in section 4.2 we develop the notion of an "operational isomorphism" between linPCF types, proving that the types $!\phi\otimes!\psi$ and $!(\phi\&\psi)$ are operationally isomorphic. Thirdly, in section 6, we give a translation of terms from non-strict PCF (defined in section 5) to linPCF, and show that the translation is adequate but *not* fully abstract with respect to contextual equivalence. We conclude, in section 7, with an indication of future work.

2 linPCF

Plotkin's PCF, the prototypical functional language,[1] (Plotkin, 1977) is an extension of the typed λ-calculus with constants, a conditional operator and recursion. Analogously, linPCF is the typed *linear* λ-calculus (Benton *et al*., 1993) extended with

[1] "The mother of all toy programming languages" (Pitts, 1997).

booleans, a conditional operator and recursion. Some details concerning linPCF are given below, but the reader to referred to other work for a fuller discussion (Braüner, 1994; Chirimar *et al.*, 1996; Bierman, 1997).

Types are given by the grammar

$$\phi ::= \mathsf{bool} \mid \phi\otimes\phi \mid \phi\multimap\phi \mid \phi\&\phi \mid !\phi,$$

and terms are given by the grammar

M, N, P	$::=$	$\mathtt{true}, \mathtt{false}$	Booleans
	$\mid$	x	Variable
	$\mid$	$\lambda x\!:\!\phi.M$	Abstraction
	$\mid$	MN	Application
	$\mid$	$M\otimes N$	Multiplicative Pair
	$\mid$	$\mathsf{let}\ M\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ N$	Split
	$\mid$	$\langle M, N\rangle$	Additive Pair
	$\mid$	$\mathsf{fst}(M) \mid \mathsf{snd}(M)$	Projections
	$\mid$	$\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P$	Conditional
	$\mid$	$\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N$	Promote
	$\mid$	$\mathsf{derelict}(M)$	Derelict
	$\mid$	$\mathsf{discard}\ M\ \mathsf{in}\ N$	Discarding
	$\mid$	$\mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N$	Duplication
	$\mid$	$\mathsf{rec}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y\!:\!\phi.N$	Recursion;

where x, y are taken from some countable set of variables and ϕ is a well-formed type.

A typing judgement is written $\Gamma \triangleright M\!:\!\phi$ where Γ is a set of (variable,type)-pairs. The convention is that the variable names are distinct, thus $\Gamma, x\!:\!\phi$ denotes the disjoint union of the sets Γ and $\{(x, \phi)\}$. Accordingly Γ, Δ denotes the disjoint union of the sets Γ and Δ. The rules for forming a valid typing judgement are given in figure 1. In this paper, all linPCF programs are assumed to be well-typed.

It is worth emphasizing some of the features of this language. The computational significance of the linearity is that variables are used exactly once – this is reflected in the typing relation by the fact that if a term M has a typing judgement $\Gamma \triangleright M\!:\!\phi$, then its free variables are *exactly* those contained in the set Γ. Linearity also means that we have two, distinct forms of pairs: $M\otimes N$ is a “multiplicative” pair where both components are used, and so the free variables of M and N are disjoint; $\langle M, N\rangle$ is an “additive” pair where only one component is used, and so the free variables of M and N are required to be identical.

Of course a language which can only use things exactly once would be very weak, computationally speaking. The linear type system allows terms to be used non-linearly, that is discarded or duplicated explicitly, but only if they are of a nonlinear “!” type – see the rules Weakening and Contraction. For example, here is the linPCF equivalent of the **K** combinator, along with its type.

$$\lambda x\!:\!\phi.\lambda y\!:\!!\psi.\mathsf{discard}\ y\ \mathsf{in}\ x : \phi\multimap !\psi\multimap\phi$$

Terms of the non-linear “!” type are constructed using the Promotion rule and are of the following form (in some cases promote is abbreviated to prom):

$$\mathsf{promote}\ M_1, \ldots, M_k\ \mathsf{for}\ x_1, \ldots, x_k\ \mathsf{in}\ N$$

$$\frac{}{x\colon\phi \rhd x\colon\phi} \qquad \frac{}{\emptyset \rhd b\colon\mathsf{bool}}$$

$$\frac{\Gamma \rhd M\colon\phi \qquad \Delta \rhd N\colon\psi}{\Gamma,\Delta \rhd M\otimes N\colon\phi\otimes\psi}\otimes_{\mathscr{I}} \qquad \frac{\Gamma \rhd M\colon\phi\otimes\psi \qquad \Delta, x\colon\phi, y\colon\psi \rhd N\colon\varphi}{\Gamma,\Delta \rhd \mathsf{let}\ M\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ N\colon\varphi}\otimes_{\mathscr{E}}$$

$$\frac{\Gamma, x\colon\phi \rhd M\colon\psi}{\Gamma \rhd \lambda x\colon\phi.M\colon\phi\multimap\psi}\multimap_{\mathscr{I}} \qquad \frac{\Gamma \rhd M\colon\phi\multimap\psi \qquad \Delta \rhd N\colon\phi}{\Gamma,\Delta \rhd MN\colon\psi}\multimap_{\mathscr{E}}$$

$$\frac{\Gamma \rhd M\colon\phi \qquad \Gamma \rhd N\colon\psi}{\Gamma \rhd \langle M,N\rangle\colon\phi\&\psi}\&_{\mathscr{I}} \quad \frac{\Gamma \rhd M\colon\phi\&\psi}{\Gamma \rhd \mathsf{fst}(M)\colon\phi}\&_{\mathscr{E}-1} \quad \frac{\Gamma \rhd M\colon\phi\&\psi}{\Gamma \rhd \mathsf{snd}(M)\colon\psi}\&_{\mathscr{E}-2}$$

$$\frac{\Gamma_1 \rhd M_1\colon !\phi_1 \cdots \Gamma_n \rhd M_n\colon !\phi_n \qquad x_1\colon !\phi_1,\ldots,x_n\colon !\phi_n \rhd N\colon\psi}{\Gamma_1,\ldots,\Gamma_n \rhd \mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N\colon !\psi}\text{Promotion}$$

$$\frac{\Gamma \rhd M\colon !\phi}{\Gamma \rhd \mathsf{derelict}(M)\colon\phi}\text{Dereliction}$$

$$\frac{\Gamma \rhd M\colon !\phi \qquad \Delta \rhd N\colon\psi}{\Gamma,\Delta \rhd \mathsf{discard}\ M\ \mathsf{in}\ N\colon\psi}\text{Weakening} \quad \frac{\Gamma \rhd M\colon !\phi \qquad \Delta, x\colon !\phi, y\colon !\phi \rhd N\colon\psi}{\Gamma,\Delta \rhd \mathsf{copy}\ M\ \mathsf{as}\ x,y\ \mathsf{in}\ N\colon\psi}\text{Contraction}$$

$$\frac{\Gamma \rhd M\colon\mathsf{bool} \qquad \Delta \rhd N\colon\phi \qquad \Delta \rhd P\colon\phi}{\Gamma,\Delta \rhd \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P\colon\phi}\text{Conditional}$$

$$\frac{\Gamma_1 \rhd M_1\colon !\phi_1 \cdots \Gamma_n \rhd M_n\colon !\phi_n \qquad x_1\colon !\phi_1,\ldots,x_n\colon !\phi_n, y\colon !\psi \rhd N\colon\psi}{\Gamma_1,\ldots,\Gamma_n \rhd \mathsf{rec}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y\colon !\psi.N\colon\psi}\text{Recursion}$$

Fig. 1. Typing rules for linPCF.

Looking at the Promotion rule, the reader will see that $x_1,\ldots,x_k$ are exactly the free variables of N, and are all of a non-linear type. This is quite intuitive – if we wish a term N to be discarded or duplicated, all its free variables must also be able to be discarded or duplicated. The bindings of terms for these free variables is to ensure that the language is well-behaved in the following sense (again, this is well documented in the literature).

Proposition 1 (*Closure under substitution*)
If $\Gamma \rhd M\colon\phi$ and $\Delta, x\colon\phi \rhd N\colon\psi$ then $\Gamma,\Delta \rhd N[x := M]\colon\psi$.

Although the syntax for Promotion looks cumbersome, it is often the case that a term has no free variables, in which case the term

$$\mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ M$$

will be abbreviated to the more palatable

$$\mathit{promote}(M).$$

The Recursion rule forces bindings of free variables for exactly the same reasons.[2] Further details are given by Braüner (1994).

A linPCF term M containing no free variables is said to be *closed* (otherwise it is said to be *open*), in which case it is called a *program*. The set of linPCF-terms which can be assigned the type ϕ given Γ shall be written $\mathrm{Exp}_\Gamma(\phi)$. If the set Γ is empty, this shall be abbreviated to $\mathrm{Exp}(\phi)$.

The process of executing a linPCF program can be given using an evaluation relation between terms and terms in canonical form, or *values*. In his influential paper, Abramsky (1993) proposed that the refined connectives of linear logic actually encode a natural evaluation strategy: in particular, linear function application is strict, multiplicative pairs are strict, and additive pairs are non-strict. In this paper, we adopt Abramsky's proposal and extend his rules to linPCF. First, linPCF canonicals, or *values*, are defined inductively as follows:

$$v ::= \mathtt{true} \mid \mathtt{false} \mid \lambda x\!:\!\phi.M \mid v \otimes v \mid \langle M, M\rangle \mid \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ M.$$

The evaluation relation, written $M \Downarrow v$, is given in figure 2.

Proposition 2
Evaluation is deterministic and preserves typing, i.e.

1. (Determinacy) If $M \Downarrow v$ and $M \Downarrow v'$ then $v = v'$.
2. (Subject Reduction) If $\emptyset \rhd M\!:\!\phi$ and $M \Downarrow v$ then $\emptyset \rhd v\!:\!\phi$.

In what follows, we will use the following definitions:

$$\begin{aligned} M \Downarrow &\overset{\text{def}}{=} \exists v.M \Downarrow v \\ M \Uparrow &\overset{\text{def}}{=} \neg(\exists v.M \Downarrow v) \\ \Omega^\phi &\overset{\text{def}}{=} \mathit{rec}(y\!:\!!\phi.\mathsf{derelict}(y)) \end{aligned}$$

3 Program Equivalence

Morris-style contextual equivalence is commonly accepted as the natural notion of equivalence for functional languages. Essentially, two programs are considered contextually equivalent if interchanging one for the other in any larger program does not affect the result. Another way of thinking of this is that two programs are considered contextually equivalent if one cannot observe any difference between them. In this section, we define a notion of contextual equivalence for linPCF, and then give an alternative coinductive characterisation.

3.1 Contexts

Before we can formalise the definition of contextual equivalence, we first need to define a *context*. A context is simply a term with designated place-holders, or *holes*, into which other terms may be placed. An important feature is that this placement

[2] The term rec $-$ for $-$ in $y.M$ will similarly be abbreviated to $\mathit{rec}(y.M)$.

$$\frac{}{b \Downarrow b}\,(\Downarrow \text{Bool}) \qquad \frac{}{\lambda x\!:\!\phi.M \Downarrow \lambda x\!:\!\phi.M}\,(\Downarrow \multimap_{\mathcal{I}})$$

$$\frac{M \Downarrow \lambda x\!:\!\phi.P \qquad N \Downarrow v \qquad P[x := v] \Downarrow v'}{MN \Downarrow v'}\,(\Downarrow \multimap_{\mathcal{E}})$$

$$\frac{M \Downarrow v \qquad N \Downarrow v'}{M \otimes N \Downarrow v \otimes v'}\,(\Downarrow \otimes_{\mathcal{I}}) \qquad \frac{M \Downarrow v \otimes v' \qquad N[x := v, y := v'] \Downarrow v''}{\mathsf{let}\ M\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ N \Downarrow v''}\,(\Downarrow \otimes_{\mathcal{E}})$$

$$\frac{M \Downarrow \texttt{true} \qquad N \Downarrow v}{\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P \Downarrow v}\,(\Downarrow \text{Cond}) \qquad \frac{M \Downarrow \texttt{false} \qquad P \Downarrow v}{\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P \Downarrow v}\,(\Downarrow \text{Cond})$$

$$\frac{}{\langle M, N\rangle \Downarrow \langle M, N\rangle}\,(\Downarrow \&)$$

$$\frac{M \Downarrow \langle N, N'\rangle \qquad N \Downarrow v}{\mathsf{fst}(M) \Downarrow v}\,(\Downarrow \&_{\mathcal{E}}) \qquad \frac{M \Downarrow \langle N, N'\rangle \qquad N' \Downarrow v'}{\mathsf{snd}(M) \Downarrow v'}\,(\Downarrow \&_{\mathcal{E}})$$

$$\frac{M_i \Downarrow v_i \qquad 0 \leqslant i \leqslant |\vec{M}|}{\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N}\,(\Downarrow \text{Promotion})$$

$$\frac{M \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N \qquad N[\vec{x} := \vec{v}] \Downarrow v'}{\mathsf{derelict}(M) \Downarrow v'}\,(\Downarrow \text{Dereliction})$$

$$\frac{M \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{z}\ \mathsf{in}\ P \qquad N[x, y := \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{z}\ \mathsf{in}\ P] \Downarrow v'}{\mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N \Downarrow v'}\,(\Downarrow \text{Contraction})$$

$$\frac{M \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ P \qquad N \Downarrow v'}{\mathsf{discard}\ M\ \mathsf{in}\ N \Downarrow v'}\,(\Downarrow \text{Weakening})$$

$$\frac{M_i \Downarrow v_i \quad N[\vec{x} := \vec{v}, y := \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}'\ \mathsf{in}\ \mathsf{rec}\ \vec{x}'\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y.N] \Downarrow v' \quad 0 \leqslant i \leqslant |\vec{M}|}{\mathsf{rec}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y.N \Downarrow v'}\,(\Downarrow \text{Recursion})$$

Fig. 2. Evaluation relation for linPCF.

of terms for holes is permitted to capture free variables, in contrast to the familiar substitution of terms for variables.

For languages such as PCF, a traditional treatment of contexts (e.g. Pitts, 1997) is first to extend the syntactic class of terms to allow holes, and then add a new typing rule. For linPCF a hole is represented by the symbol ξ, which will be indexed if there are several distinct holes in a term. The obvious typing rule is then

$$\frac{}{\xi\!:\!\phi \triangleright \xi\!:\!\phi.}$$

However, there are soon problems: one cannot even well-type the linear context

$$\lambda x\!:\!\phi.\xi$$

(The linearity constraint means that to abstract the variable x, it must be a free variable of the body, ξ, which it clearly is not).

The solution (as is familiar with the linear setting) is to be more explicit. As explained earlier, holes are place-holders into which *open* terms may be placed, whose free variables may be captured, or bound. The important information here is the free variables. Consequently, holes should really be parameterised with these free variables. The typing rule for holes becomes

$$\frac{}{\xi(\Gamma):\phi;\Gamma \rhd \xi(\Gamma):\phi.}$$

Thus $\xi(\Gamma):\phi$ represents a hole of type ϕ, which can be filled with a term whose set of free variables is equal to the variables contained in the set Γ. In the typing judgement above, the holes and variables are separated in the antecedent with a semi-colon but this is just for clarity. The typing rules for contexts are straightforward. The earlier problematic example can now be well-typed as follows:

$$\dfrac{\dfrac{}{\xi(x:\phi):\phi;x:\phi \rhd \xi(x:\phi):\phi}}{\xi(x:\phi):\phi \rhd \lambda x:\phi.\xi(x:\phi):\phi \multimap \phi}\multimap_{\mathscr{I}}$$

The action of placing a term (or, more generally, a context) for a hole is given by the rule

$$\dfrac{\mathscr{H};\Gamma \rhd \mathscr{C}:\phi \qquad \mathscr{H}',\xi(\Gamma):\phi;\Delta \rhd \mathscr{D}:\psi}{\mathscr{H},\mathscr{H}';\Delta \rhd \mathscr{D}[\mathscr{C}]:\psi}\text{ Placement}$$

Thus, one can only place a context, $\mathscr{C}$, for the hole $\xi(\Gamma)$ if its set of free variables is Γ. The result of this placement is then defined by induction on the structure of $\mathscr{D}$. The result of placing a context $\mathscr{C}$ for the hole is written $\mathscr{D}[\mathscr{C}]$. In the case where there is only one hole, the symbol $\bullet$ will be used rather than ξ.

It should be noted that there is nothing inherently linear about this very general treatment of contexts, indeed it can be applied to any language. A number of other people have (independently) suggested similar extensions to the traditional notion of context; for example, Pitts (1994) and Hashimoto and Ohori (1996).

3.2 Contextual equivalence

Having formalised the notion of a linPCF context, it is now possible to give a definition of contextual equivalence.

Definition 1
Given $\Gamma \rhd M:\phi$ and $\Gamma \rhd N:\phi$, M is said to *contextually refine* N, written $\Gamma \rhd M \sqsubseteq N:\phi$, iff all closing contexts, $\bullet(\Gamma):\phi \rhd \mathscr{C}:\psi$, if $\mathscr{C}[M] \Downarrow$ then $\mathscr{C}[N] \Downarrow$.

Contextual equivalence, written $\Gamma \rhd M \approx N:\phi$, holds iff $\Gamma \rhd M \sqsubseteq N:\phi$ and $\Gamma \rhd N \sqsubseteq M:\phi$.

Of course, having given a precise definition of contextual equivalence, it remains to explore its theory, i.e. which terms are, or are not, contextually equivalent. It is quite easy to show when two terms are *not* equivalent: one simply finds a context which distinguishes them. Unfortunately, demonstrating that two terms are contextually equivalent is much harder: the problem is essentially the quantification over all contexts. What is needed is an alternative characterisation which is more suitable for proofs. One approach is to move to a more abstract, mathematical setting – typically, some form of domain theory. The approach taken here is rather to characterise contextual equivalence as a form of bisimilarity (this approach was first suggested by Abramsky (1990)).

This alternative equivalence, *applicative similarity*, is defined as the greatest fixed point of a certain monotone operation on relations. This operation is given in two stages.

Definition 2
Given a family of (type-indexed) relations $R = (R_\phi \subseteq \mathrm{Exp}(\phi) \times \mathrm{Exp}(\phi))$ between closed linPCF terms, one can define a family of relations $\langle R\rangle_\phi$ between closed values as follows:

- $b\ \langle R\rangle_{\mathsf{bool}}\ b'$ if $b \equiv b'$,
- $v_1 \otimes v_2\ \langle R\rangle_{\phi\otimes\psi}\ v_1' \otimes v_2'$ if $v_1 R_\phi v_1'$ and $v_2 R_\psi v_2'$,
- $\lambda x{:}\phi.M\ \langle R\rangle_{\phi\multimap\psi}\ \lambda x{:}\phi.M'$ if $\forall v{:}\phi.M[x := v]\ R_\psi\ M'[x := v]$,
- $\langle M, N\rangle\ \langle R\rangle_{\phi\&\psi}\ \langle M', N'\rangle$ if $M\ R_\phi\ M'$ and $N\ R_\psi\ N'$, and
- promote $\vec{v}$ for $\vec{x}$ in $M\ \langle R\rangle_{!\phi}$ promote $\vec{v'}$ for $\vec{x'}$ in M' if $M[\vec{x} := \vec{v}]\ R_\phi\ M'[\vec{x'} := \vec{v'}]$.

This definition is extended to closed linPCF terms as follows:

$$M[R]_\phi N \iff \forall v.\text{if } M \Downarrow v \text{ then } \exists v'.N \Downarrow v' \text{ and } v\ \langle R\rangle_\phi\ v'$$

A family of relations, R, satisfying $R \subseteq [R]$, is called a (linPCF) *simulation*. As the function $R \mapsto [R]$ is monotone and the families indexed by their types form a complete lattice then it has a *greatest* fixed point, which is written $\leqslant$, and referred to as (linPCF) *applicative similarity*. This relation can be extended to open linPCF terms as follows:

$$\vec{x}{:}\Gamma \rhd M \leqslant^\circ N{:}\psi \iff \forall \vec{v}.\ M[\vec{x} := \vec{v}] \leqslant N[\vec{x} := \vec{v}]{:}\psi$$

where the v_i are values. It is easy to show that the relation $\leqslant$ is a partial order. Applicative bisimilarity, written $\approx_{app}$, is defined as the symmetrisation of $\leqslant$, i.e. $\Gamma \rhd M \approx^\circ_{app} N{:}\phi$ iff $\Gamma \rhd M \leqslant^\circ N{:}\phi$ and $\Gamma \rhd N \leqslant^\circ M{:}\phi$.

The fact that (bi)similarity is defined as the greatest fixed point yields a powerful and useful proof technique.

Proposition 3 (*Coinduction Principle*)
Given $M, N \in \mathrm{Exp}(\phi)$, to prove that $M \leqslant N{:}\phi$ ($M \approx_{app} N{:}\phi$) it suffices to find a simulation (bisimulation) $\mathscr{S}$ such that $M \mathscr{S}_\phi N$.

The principal theorem in this paper is that applicative bisimilarity characterises completely contextual equivalence.

Theorem 1

- $\Gamma \rhd M \sqsubseteq N{:}\phi$ iff $\Gamma \rhd M \leqslant^\circ N{:}\phi$.
- $\Gamma \rhd M \approx N{:}\phi$ iff $\Gamma \rhd M \approx^\circ_{app} N{:}\phi$.

Proof
Details of the proof can be found in the appendix. □

The desirable consequence of this theorem is that the following coinduction principle holds for contextual equivalence:

To prove that two closed terms are contextually equivalent, it suffices to find a bisimulation between them.

4 Examples

Now one has a method for verifying contextual equivalences between linPCF programs. In section 4.1 we list a number of linPCF programs which can be shown to be contextually equivalent. In section 4.2 we define and study the notion of an "operational isomorphism" between linPCF types.

4.1 Program equivalences

As explained earlier, to demonstrate that two linPCF programs are contextually equivalent, one need only find a bisimulation that relates them. Often, one can make use of an even easier relation, called *Kleene equivalence*.

Definition 3
Given $\emptyset \triangleright M : \phi$ and $\emptyset \triangleright N : \phi$, then

$$\begin{array}{lcl} M \leqslant_{kl} N : \phi & \Longleftrightarrow & \forall v. \text{If } M \Downarrow v \text{ then } N \Downarrow v, \\ M \approx_{kl} N : \phi & \Longleftrightarrow & M \leqslant_{kl} N : \phi \text{ and } N \leqslant_{kl} M : \phi. \end{array}$$

We will show that if two terms are Kleene equivalent, then they are contextually equivalent. The converse is, of course, not necessarily true. Consider, for example, the terms

$$\lambda x.(\lambda y.\Omega)\texttt{true} \quad \text{and} \quad \lambda x.\Omega$$

They both converge, but to different values. They are, however, contextually equivalent.

Proposition 4
If $M \leqslant_{kl} N : \phi$ then $M \sqsubseteq N : \phi$.

Proof
From Theorem 1, one need only show that if $M \leqslant_{kl} N : \phi$ then $M \leqslant N : \phi$. To demonstrate this, form the type-indexed family

$$\mathscr{S} = \mathscr{S}_\phi \stackrel{\text{def}}{=} \{(M, N) | M \leqslant_{kl} N : \phi\}$$

and show that $\mathscr{S} \subseteq [\mathscr{S}]$. □

Using this fact, it is quite easy to show that the β-rules of linPCF (Bierman, 1997)

β-rules:

$$
\begin{array}{rcll}
(\lambda x\!:\!\phi.M)v & \approx & M[x := v] & (1)\\
\mathsf{let}\ v{\otimes}w\ \mathsf{be}\ x{\otimes}y\ \mathsf{in}\ P & \approx & P[x := v, y := w] & (2)\\
\mathsf{fst}(\langle M,N\rangle) & \approx & M & (3)\\
\mathsf{snd}(\langle M,N\rangle) & \approx & N & (4)\\
\mathsf{derelict}(\mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N) & \approx & N[\vec{x} := \vec{v}] & (5)\\
\mathsf{discard}\ (\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N)\ \mathsf{in}\ P & \approx & \mathsf{discard}\ \vec{M}\ \mathsf{in}\ P & (6)\\
\mathsf{copy}\ (\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N)\ \mathsf{as}\ y,z\ \mathsf{in}\ P & \approx & \mathsf{copy}\ \vec{M}\ \mathsf{as}\ \vec{x'},\vec{x''}\ \mathsf{in} & \\
 & & P\ [y := \mathsf{promote}\ \vec{x'}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N, & \\
 & & \quad z := \mathsf{promote}\ \vec{x''}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N] & (7)\\
\mathsf{if}\ \mathtt{true}\ \mathsf{then}\ M\ \mathsf{else}\ N & \approx & M & (8)\\
\mathsf{if}\ \mathtt{false}\ \mathsf{then}\ M\ \mathsf{else}\ N & \approx & N & (9)\\
\mathsf{rec}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y.N & \approx & \mathsf{copy}\ \vec{M}\ \mathsf{as}\ \vec{x'},\vec{x''}\ \mathsf{in} & (10)\\
 & & N\ [\vec{x} := \vec{x'}, & \\
 & & \quad y := \mathsf{prom}\ \vec{x''}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ \mathsf{rec}\ \vec{x}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y.N] & (11)
\end{array}
$$

Commuting conversions:

$$
\begin{array}{rcll}
\mathsf{let}\ (\mathsf{let}\ M\ \mathsf{be}\ x{\otimes}x'\ \mathsf{in}\ N)\ \mathsf{be}\ y{\otimes}y'\ \mathsf{in}\ P & \approx & \mathsf{let}\ M\ \mathsf{be}\ x{\otimes}x'\ \mathsf{in}\ \mathsf{let}\ N\ \mathsf{be}\ y{\otimes}y'\ \mathsf{in}\ P & (12)\\
\mathsf{copy}\ (\mathsf{let}\ M\ \mathsf{be}\ x{\otimes}x'\ \mathsf{in}\ N)\ \mathsf{as}\ y,z\ \mathsf{in}\ P & \approx & \mathsf{let}\ M\ \mathsf{be}\ x{\otimes}x'\ \mathsf{in}\ \mathsf{copy}\ N\ \mathsf{as}\ y,z\ \mathsf{in}\ P & (13)\\
\mathsf{let}\ (\mathsf{copy}\ M\ \mathsf{as}\ x,y\ \mathsf{in}\ N)\ \mathsf{be}\ z{\otimes}z'\ \mathsf{in}\ P & \approx & \mathsf{copy}\ M\ \mathsf{as}\ x,y\ \mathsf{in}\ \mathsf{let}\ N\ \mathsf{be}\ z{\otimes}z'\ \mathsf{in}\ P & (14)
\end{array}
$$

Comonoid Equations:

$$
\begin{array}{rcll}
\mathsf{discard}\ v\ \mathsf{in}\ M & \approx & M & (15)\\
\mathsf{copy}\ v\ \mathsf{as}\ x,y\ \mathsf{in}\ \mathsf{discard}\ x\ \mathsf{in}\ M & \approx & M[y := v] & (16)
\end{array}
$$

Comonad Equations:

$$
\begin{array}{rcll}
\mathsf{prom}\ M,(\mathsf{prom}\ N\ \mathsf{for}\ x\ \mathsf{in}\ P)\ \mathsf{for}\ y,z\ \mathsf{in}\ Q & \approx & \mathsf{prom}\ M,N\ \mathsf{for}\ y,x' & \\
 & & \mathsf{in}\ Q[z := \mathsf{prom}\ x'\ \mathsf{for}\ x\ \mathsf{in}\ P] & (17)\\
\mathsf{promote}\ M\ \mathsf{for}\ x\ \mathsf{in}\ \mathsf{derelict}(x) & \approx & M & (18)
\end{array}
$$

Additional Equivalences:

$$
\begin{array}{rcll}
\Omega & \sqsubseteq & M & (19)\\
\lambda x.vx & \approx & v & (20)\\
\mathsf{let}\ v\ \mathsf{be}\ x{\otimes}y\ \mathsf{in}\ x{\otimes}y & \approx & v & (21)\\
\langle \mathsf{fst}(v),\mathsf{snd}(v)\rangle & \approx & v & (22)\\
\mathsf{copy}\ v\ \mathsf{as}\ x,y\ \mathsf{in}\ M & \approx & M[x,y := v] & (23)\\
\mathsf{promote}\ v\ \mathsf{for}\ x\ \mathsf{in}\ M & \approx & \mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ M[x := v] & (24)
\end{array}
$$

Fig. 3. Example linPCF program equivalences.

are contextual equivalences. These, along with a number of other useful equivalences, are listed in figure 3.

An important property of an applicative bisimulation relation is that it is a *precongruence* (this is proved in Proposition 12 in the appendix). This means that it is both compatible (it respects the term formation rules) and transitive. This is an important aid to proving that two programs are applicatively bisimilar, as it allows equational reasoning. For example, one of the rules for compatibility of a relation $\mathscr{R}$ is

$$\frac{\Gamma \triangleright M \mathscr{R} N : \phi \multimap \psi \qquad \Delta \triangleright M' \mathscr{R} N' : \phi}{\Gamma, \Delta \triangleright (MM') \mathscr{R} (NN') : \psi.}$$

The fact that applicative bisimilarity is a precongruence will be used extensively in the following section.

4.2 Operational isomorphisms

It is a simple exercise in proof theory to see that a number of formulae are equiprovable in intuitionistic linear logic. Similarly, there are a number of isomorphisms between (denotations of) types in models of linPCF (Bierman, 1995). An interesting question is whether certain types are *operationally isomorphic*, i.e. whether there are functions between the types that are mutually inverse up to contextual equivalence.

Apart from theoretical interest, there is a practical application to such a question. Rather than searching for a function in a library via a simple textual match of identifiers, Rittri (1991) suggested that they be searched via the function's *type*. However, given two different libraries, the same function may be defined in a different way so that the type signatures differ. Clearly, syntactic equality of types is an inappropriate search method. Di Cosmo (1993; 1995) suggested that searches be performed modulo a notion of type isomorphism. His notion of isomorphism is defined with respect to a denotational model, and he considers only a fragment of PCF.

In contrast, the following notion of type isomorphism is based on contextual equivalence. It could be used to assist library searches for a linear functional language.

Definition 4
The linPCF types ϕ and ψ are said to be *operationally isomorphic* if there exists linPCF terms I and J, such that $x : \phi \triangleright I : \psi$ and $y : \psi \triangleright J : \phi$, which also satisfy

$$x : \phi \triangleright (\lambda y : \psi . J) I \approx x : \phi, \text{and}$$

$$y : \psi \triangleright (\lambda x : \phi . I) J \approx y : \psi.$$

The rest of this section contains details of a proof that the types $!\phi \otimes !\psi$ and $!(\phi \& \psi)$ are operationally isomorphic.[3] The candidates for the mutually inverse terms are:

$$x : !\phi \otimes !\psi \triangleright \mathsf{let}\ x\ \mathsf{be}\ y \otimes z\ \mathsf{in}$$
$$\mathsf{prom}\ y, z\ \mathsf{for}\ s, t\ \mathsf{in}\ \langle \mathsf{discard}\ t\ \mathsf{in}\ \mathsf{derelict}(s), \mathsf{discard}\ s\ \mathsf{in}\ \mathsf{derelict}(t) \rangle : !(\phi \& \psi)$$

and

$$y : !(\phi \& \psi) \triangleright \mathsf{copy}\ y\ \mathsf{as}\ i, j\ \mathsf{in}$$
$$(\mathsf{prom}\ i\ \mathsf{for}\ k\ \mathsf{in}\ \mathsf{fst}(\mathsf{derelict}(k))) \otimes (\mathsf{prom}\ j\ \mathsf{for}\ l\ \mathsf{in}\ \mathsf{snd}(\mathsf{derelict}(l))) : !\phi \otimes !\psi.$$

We shall consider only one direction of the isomorphism to save space. The reader

[3] This isomorphism is important in categorical models of linear logic (Seely, 1989; Bierman, 1995).

is invited to prove the other direction. Thus, we aim to prove

$$(\lambda x.\mathsf{let}\ x\ \mathsf{be}\ y{\otimes}z\ \mathsf{in}\ \mathsf{prom}\ y,z\ \mathsf{for}\ s,t\ \mathsf{in}\ \langle D,E\rangle)(\mathsf{copy}\ v\ \mathsf{as}\ i,j\ \mathsf{in}\ B{\otimes}C) \approx_{app} v : !(\phi \& \psi)$$

where

$$\begin{array}{lcl} B & \overset{\text{def}}{=} & \mathsf{prom}\ i\ \mathsf{for}\ k\ \mathsf{in}\ \mathsf{fst}(\mathsf{derelict}(k)), \\ C & \overset{\text{def}}{=} & \mathsf{prom}\ j\ \mathsf{for}\ l\ \mathsf{in}\ \mathsf{snd}(\mathsf{derelict}(l)), \\ D & \overset{\text{def}}{=} & \mathsf{discard}\ t\ \mathsf{in}\ \mathsf{derelict}(s), \text{and} \\ E & \overset{\text{def}}{=} & \mathsf{discard}\ s\ \mathsf{in}\ \mathsf{derelict}(t). \end{array}$$

This can be demonstrated using the following (equational) reasoning (each line is labelled with the number of the equivalence used, where appropriate).:

$$\begin{array}{lll}
 & (\lambda x.\mathsf{let}\ x\ \mathsf{be}\ y{\otimes}z\ \mathsf{in}\ \mathsf{prom}\ y,z\ \mathsf{for}\ s,t\ \mathsf{in}\ \langle D,E\rangle)(\mathsf{copy}\ v\ \mathsf{as}\ i,j\ \mathsf{in}\ B{\otimes}C) & \\
\approx_{app} & (\lambda x.\mathsf{let}\ x\ \mathsf{be}\ y{\otimes}z\ \mathsf{in}\ \mathsf{prom}\ y,z\ \mathsf{for}\ s,t\ \mathsf{in}\ \langle D,E\rangle)(B[i:=v]{\otimes}C[j:=v]) & (23) \\
\equiv & (\lambda x.\mathsf{let}\ x\ \mathsf{be}\ y{\otimes}z\ \mathsf{in}\ \mathsf{prom}\ y,z\ \mathsf{for}\ s,t\ \mathsf{in}\ \langle D,E\rangle)(w_1{\otimes}w_2) & \\
\approx_{app} & \mathsf{let}\ w_1{\otimes}w_2\ \mathsf{be}\ y{\otimes}z\ \mathsf{in}\ \mathsf{promote}\ y,z\ \mathsf{for}\ s,t\ \mathsf{in}\ \langle D,E\rangle & (1) \\
\approx_{app} & \mathsf{promote}\ w_1,w_2\ \mathsf{for}\ s,t\ \mathsf{in}\ \langle D,E\rangle & (2) \\
\approx_{app} & \mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ \langle D[s:=w_1,t:=w_2],E[s:=w_1,t:=w_2]\rangle & (24) \\
\equiv & \mathit{promote}(\langle \mathsf{discard}\ w_2\ \mathsf{in}\ \mathsf{derelict}(w_1), \mathsf{discard}\ w_1\ \mathsf{in}\ \mathsf{derelict}(w_2)\rangle) & \\
\approx_{app} & \mathit{promote}(\langle \mathsf{derelict}(w_1), \mathsf{derelict}(w_2)\rangle) & (15) \\
\approx_{app} & \mathit{promote}(\langle \mathsf{fst}(\mathsf{derelict}(v)), \mathsf{snd}(\mathsf{derelict}(v))\rangle) & (5) \\
\approx_{app} & \mathit{promote}(\mathsf{derelict}(v)) & (22) \\
\approx_{app} & \mathsf{promote}\ v\ \mathsf{for}\ x\ \mathsf{in}\ \mathsf{derelict}(x) & (24) \\
\approx_{app} & v & (18).
\end{array}$$

Proposition 5
The types $!\phi \otimes !\psi$ and $!(\phi \& \psi)$ are operationally isomorphic.

5 PCF

In this section we shall recall some material on PCF, the prototypical non-strict functional language. The expert reader may wish to skip to the following section, where a translation from PCF to linPCF is studied. More information on PCF can be found in the literature (e.g. Winksel, 1993).

PCF consists of the typed λ-calculus extended with pairs, booleans, a conditional and recursion. The rules for forming typing judgements are as follows:

$$\frac{}{\Gamma, x : \sigma \rhd x : \sigma} \qquad \frac{}{\Gamma \rhd b : \mathsf{bool}}$$

$$\frac{\Gamma \rhd e : \sigma \qquad \Gamma \rhd f : \tau}{\Gamma \rhd \langle e, f\rangle : \sigma \times \tau}\times_{\mathscr{I}} \quad \frac{\Gamma \rhd e : \sigma \times \tau}{\Gamma \rhd \mathsf{fst}(e) : \sigma}\times_{\mathscr{E}-1} \quad \frac{\Gamma \rhd e : \sigma \times \tau}{\Gamma \rhd \mathsf{snd}(e) : \tau}\times_{\mathscr{E}-2}$$

$$\frac{\Gamma, x : \sigma \rhd e : \tau}{\Gamma \rhd \lambda x : \sigma . e : \sigma \to \tau}\to_{\mathscr{I}} \qquad \frac{\Gamma \rhd e : \sigma \to \tau \qquad \Gamma \rhd f : \sigma}{\Gamma \rhd e f : \tau}\to_{\mathscr{E}}$$

$$\frac{\Gamma \rhd e : \mathsf{bool} \qquad \Gamma \rhd f : \sigma \qquad \Gamma \rhd g : \sigma}{\Gamma \rhd \mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g : \sigma}\text{Conditional}$$

$$\frac{\Gamma, x : \sigma \rhd e : \sigma}{\Gamma \rhd \mathsf{rec}\ x : \sigma . e : \sigma}\text{Recursion}$$

The defining feature of non-strict program execution is that arguments are passed in unevaluated. Another feature is that pairs are considered to be values (the elements of a pair are not evaluated). Values are given by the inductive definition

$$c ::= \texttt{true} \mid \texttt{false} \mid \lambda x{:}\sigma.e \mid \langle e, e\rangle.$$

The evaluation relation is as follows:

$$b \Downarrow b \qquad \lambda x{:}\sigma.e \Downarrow \lambda x{:}\sigma.e \qquad \frac{e \Downarrow \lambda x{:}\sigma.g \qquad g[x := f] \Downarrow c}{ef \Downarrow c}$$

$$\langle e, f\rangle \Downarrow \langle e, f\rangle \qquad \frac{e \Downarrow \langle f, g\rangle \qquad f \Downarrow c}{\mathsf{fst}(e) \Downarrow c} \qquad \frac{e \Downarrow \langle f, g\rangle \qquad g \Downarrow c}{\mathsf{snd}(e) \Downarrow c}$$

$$\frac{e \Downarrow \texttt{true} \qquad f \Downarrow c}{\mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g \Downarrow c} \quad \frac{e \Downarrow \texttt{false} \qquad g \Downarrow c}{\mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g \Downarrow c} \qquad \frac{e[x := \mathsf{rec}\ x.e] \Downarrow c}{\mathsf{rec}\ x.e \Downarrow c}$$

Now to contextual equivalence and applicative bisimilarity for PCF: both Gordon (1995) and Pitts (1997) have offered definitions – here we shall follow those given by Pitts.

Definition 5
Given $\Gamma \rhd e{:}\sigma$ and $\Gamma \rhd f{:}\sigma$, e is said to *contextually refine* f, written $\Gamma \rhd e \sqsubseteq_{gnd} f{:}\sigma$, iff for all closing boolean contexts, $\bullet(\Gamma){:}\sigma \rhd \mathscr{C}{:}\mathsf{bool}$, if $\mathscr{C}[e] \Downarrow \texttt{true}$ then $\mathscr{C}[f] \Downarrow \texttt{true}$.

Ground contextual equivalence, written $\Gamma \rhd e \approx_{gnd} f{:}\sigma$, holds iff $\Gamma \rhd e \sqsubseteq_{gnd} f{:}\sigma$ and $\Gamma \rhd f \sqsubseteq_{gnd} e{:}\sigma$.

Definition 6
Given a family of (type-indexed) relations $R = (R_\sigma \subseteq \mathrm{Exp}(\sigma) \times \mathrm{Exp}(\sigma))$ between closed PCF terms, one can define a family of relations $[R]_\sigma$ as follows:

- $e[R]_{\mathsf{bool}}f$ iff $\forall b.$ if $e \Downarrow b$ then $f \Downarrow b$,
- $e[R]_{\sigma\times\tau}f$ iff $\mathsf{fst}(e)\ R_\sigma\ \mathsf{fst}(f)$ and $\mathsf{snd}(e)\ R_\tau\ \mathsf{snd}(f)$,
- $e[R]_{\sigma\to\tau}f$ iff $\forall g{:}\sigma.eg\ R_\tau\ fg$.

A family of relations, R, satisfying $R \subseteq [R]$, is called a (PCF) simulation. As the function $R \mapsto [R]$ is monotone and the families indexed by their types form a complete lattice then the function has a *greatest* fixed point, which is written $\leqslant$, and referred to as (PCF) *applicative similarity*. This relation is extended to open PCF terms as follows:

$$\vec{x}{:}\Gamma \rhd e \leqslant^\circ f{:}\tau \iff \forall \vec{g}.\ e[\vec{x} := \vec{g}] \leqslant f[\vec{x} := \vec{g}]{:}\tau$$

where the g_i are (closed) PCF terms. Applicative bisimilarity, written $\approx_{app}$, is defined as the symmetrisation of $\leqslant$, i.e. $\Gamma \rhd e \approx^\circ_{app} f{:}\sigma$ iff $\Gamma \rhd e \leqslant^\circ f{:}\sigma$ and $\Gamma \rhd f \leqslant^\circ e{:}\sigma$. It is then possible to show that these two notions of program equivalence coincide.

Theorem 2
$\Gamma \rhd e \approx_{gnd} f{:}\sigma$ iff $\Gamma \rhd e \approx^\circ_{app} f{:}\sigma$.

Proof
An analogous proof is given in detail by Pitts (1997). □

6 Translation of PCF to linPCF

In this section we will show how to translate PCF programs into linPCF programs, and study the properties of this translation with respect to contextual equivalence.

In his seminal paper, Girard presented a translation of formulae from intuitionistic logic (**IL**) to intuitionistic linear logic (**ILL**) which he denoted by $(-)^\circ$. It has become folklore that this corresponds to a call-by-name translation. The translation is as follows.[4]

$$\begin{array}{rcl}
\mathsf{bool}^\circ & \overset{\text{def}}{=} & \mathsf{bool} \\
(\sigma \to \tau)^\circ & \overset{\text{def}}{=} & !\sigma^\circ \multimap \tau^\circ \\
(\sigma \times \tau)^\circ & \overset{\text{def}}{=} & !\sigma^\circ \otimes !\tau^\circ
\end{array}$$

The reader will recall from section 2 that objects of type $!\phi$ are left unevaluated. Thus, the translation of a function type $\sigma \to \tau$ to $!\sigma^\circ \multimap \tau^\circ$ indicates that arguments are passed in unevaluated, i.e. a call-by-name strategy. The translation can be given at the level of typing derivations as follows:

$$\begin{array}{rcl}
|\vec{x}:\Gamma \rhd b:\mathsf{bool}|^\circ & \overset{\text{def}}{=} & \vec{x}: !\Gamma^\circ \rhd \mathsf{discard}\ \vec{x}\ \mathsf{in}\ b:\mathsf{bool} \\
|\vec{x}:\Gamma, y:\sigma \rhd y:\sigma|^\circ & \overset{\text{def}}{=} & \vec{x}: !\Gamma^\circ, y: !\sigma^\circ \rhd \mathsf{discard}\ \vec{x}\ \mathsf{in}\ \mathsf{derelict}(y):\sigma^\circ \\
|\Gamma \rhd \lambda y:\sigma.e:\sigma \to \tau|^\circ & \overset{\text{def}}{=} & !\Gamma^\circ \rhd \lambda y: !\sigma^\circ.|e|^\circ : !\sigma^\circ \multimap \tau^\circ \\
|\vec{x}:\Gamma \rhd ef:\tau|^\circ & \overset{\text{def}}{=} & \vec{x}: !\Gamma^\circ \rhd \mathsf{copy}\ \vec{x}\ \mathsf{as}\ \vec{x}', \vec{x}'' \\
& & \mathsf{in}\ ((|e[\vec{x} := \vec{x}']|^\circ)(\mathsf{promote}\ \vec{x}''\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ |f|^\circ)):\tau^\circ \\
|\vec{x}:\Gamma \rhd \langle e, f\rangle :\sigma \times \tau|^\circ & \overset{\text{def}}{=} & \vec{x}: !\Gamma^\circ \rhd \mathsf{copy}\ \vec{x}\ \mathsf{as}\ \vec{x}', \vec{x}'' \\
& & \mathsf{in}\ (\mathsf{promote}\ \vec{x}'\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ |e|^\circ) \\
& & \quad \otimes(\mathsf{promote}\ \vec{x}''\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ |f|^\circ): !\sigma^\circ \otimes !\tau^\circ \\
|\Gamma \rhd \mathsf{fst}(e):\sigma|^\circ & \overset{\text{def}}{=} & !\Gamma^\circ \rhd \mathsf{let}\ |e|^\circ\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ (\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x)):\sigma^\circ \\
|\Gamma \rhd \mathsf{snd}(e):\tau|^\circ & \overset{\text{def}}{=} & !\Gamma^\circ \rhd \mathsf{let}\ |e|^\circ\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ (\mathsf{discard}\ x\ \mathsf{in}\ \mathsf{derelict}(y)):\tau^\circ \\
|\vec{x}:\Gamma \rhd \mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g:\sigma|^\circ & \overset{\text{def}}{=} & \vec{x}: !\Gamma^\circ \rhd \mathsf{copy}\ \vec{x}\ \mathsf{as}\ \vec{x}', \vec{x}'' \\
& & \mathsf{in}\ (\mathsf{if}\ |e[\vec{x} := \vec{x}']|^\circ\ \mathsf{then}\ |f[\vec{x} := \vec{x}'']|^\circ\ \mathsf{else}\ |g[\vec{x} := \vec{x}'']|^\circ):\sigma^\circ \\
|\vec{x}:\Gamma \rhd \mathsf{rec}\ y:\sigma.e:\sigma|^\circ & \overset{\text{def}}{=} & \vec{x}: !\Gamma^\circ \rhd \mathsf{rec}\ \vec{x}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y: !\sigma^\circ.|e|^\circ:\sigma^\circ
\end{array}$$

The way this translation interacts with substitution means that the $|-|^\circ$ translation does *not* preserve evaluation, i.e. if $e \Downarrow c$ then it is not necessarily the case that $|e|^\circ \Downarrow |c|^\circ$. A counterexample is the term $(\lambda x.\lambda y.x)\mathtt{true}$, where

$$(\lambda x.\lambda y.x)\mathtt{true} \Downarrow \lambda y.\mathtt{true}.$$

Clearly $|\lambda y.\mathtt{true}|^\circ = \lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathtt{true}$, but

$$|(\lambda x.\lambda y.x)\mathtt{true}|^\circ = (\lambda x.\lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x))\mathit{promote}(\mathtt{true})$$

and

$$(\lambda x.\lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x))\mathit{promote}(\mathtt{true}) \Downarrow \lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(\mathit{promote}(\mathtt{true})).$$

However, if attention is restricted only to programs which evaluate to a boolean then a useful result does hold:

[4] In fact Girard translates products into additive products – this variant is considered at the end of this section.

Proposition 6
If $e \Downarrow b$ then $|e|^\circ \Downarrow b$.

Proof
See Theorem 10.3.1 of Braüner (1996). □

There is also a trivial translation on formulae in the other direction: from **ILL** to **IL**, which replaces the linear connectives with their intuitionistic counterparts and deletes any occurrences of the exponential. This is written $(-)^s$, and is given by

$$\begin{array}{rclcrcl} \mathsf{bool}^s & \stackrel{\text{def}}{=} & \mathsf{bool} & \quad & (\phi \multimap \psi)^s & \stackrel{\text{def}}{=} & \phi^s \to \psi^s \\ (\phi \otimes \psi)^s & \stackrel{\text{def}}{=} & \phi^s \times \psi^s & & (!\phi)^s & \stackrel{\text{def}}{=} & \phi^s. \end{array}$$

This translation can be extended to typing judgements in an obvious way. Thus, we now have maps between PCF and linPCF in both directions. The maps are related in the following sense:

Proposition 7
For all PCF terms e, $||\Gamma \rhd e : \sigma|^\circ|^s \equiv \Gamma \rhd e : \sigma$.

However, there is little interesting to say about the composition of the maps in the other direction. The $|-|^s$ translation erases all the information concerning the exponential, which is then re-introduced in an entirely uniform way by the $|-|^\circ$ translation. Indeed, the composition need not even preserve the type of a term, for example

$$||\emptyset \rhd \lambda x : \mathsf{bool}.x : \mathsf{bool} \multimap \mathsf{bool}|^s|^\circ \stackrel{\text{def}}{=} \emptyset \rhd \lambda x : !\mathsf{bool}.\mathsf{derelict}(x) : !\mathsf{bool} \multimap \mathsf{bool}.$$

In addition, one might wonder whether the $|-|^s$ translation preserves evaluation, i.e.

$$\text{If } M \Downarrow v \text{ then } |M|^s \Downarrow |v|^s,$$

but a moment's thought shows that this is not true; the $|-|^s$ translation does not even preserve values. (A counterexample is the term (and value) *promote*(Ω).) However, one can prove the converse to Proposition 6.

Proposition 8
If $|e|^\circ \Downarrow b$ then $e \Downarrow b$.

Proof
See Theorem 10.3.1 of (Braüner, 1996). □

We come now to the main result of this section. If the call-by-name translation reflects contextual equivalence it is termed *adequate*. Should it, in addition, preserve contextual equivalence it is termed *fully abstract*. Fortunately, the translation is adequate, the essence of which is given in the following proposition.

Proposition 9
If $|e|^\circ \approx_{gnd} |f|^\circ : \sigma^\circ$ then $e \approx_{gnd} f : \sigma$.

Proof
Form the type-indexed family $\mathscr{S} = \mathscr{S}_\sigma \overset{\text{def}}{=} \{(e,f) \mid |e|^\circ \approx_{gnd} |f|^\circ : \sigma^\circ\}$, and show that $\mathscr{S} \subseteq [\mathscr{S}]$. $\square$

Corollary 1
The call-by-name translation is adequate.

Unfortunately, the translation is *not* fully abstract.

Theorem 3
The call-by-name translation is *not* fully abstract, i.e. there are PCF programs e and f such that $e \approx_{gnd} f : \sigma$ and $|e|^\circ \not\approx_{gnd} |f|^\circ : \sigma^\circ$.

Proof
First, observe that

$$|\Omega^\sigma|^\circ \overset{\text{def}}{=} |\mathsf{rec}\ x\!:\!\sigma.x|^\circ \overset{\text{def}}{=} \mathit{rec}(x\!:\,!\sigma^\circ.\mathsf{derelict}(x)) \overset{\text{def}}{=} \Omega^{\sigma^\circ}.$$

In call-by-name PCF we have that $e \approx_{gnd} \langle \mathsf{fst}(e), \mathsf{snd}(e)\rangle : \sigma \times \tau$, for all e (Equation 2.25 of (Pitts, 1997)). Consider the case when $e \equiv \mathsf{rec}\ x.x \equiv \Omega^{\mathsf{bool}\times\mathsf{bool}}$, thus

$$\begin{array}{rcl} |\Omega|^\circ & \overset{\text{def}}{=} & \Omega,\ \text{and} \\ |\langle \mathsf{fst}(\Omega), \mathsf{snd}(\Omega)\rangle|^\circ & \overset{\text{def}}{=} & \mathit{promote}(\mathsf{let}\ \Omega\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x)) \\ & & \otimes\ \mathit{promote}(\mathsf{let}\ \Omega\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathsf{discard}\ x\ \mathsf{in}\ \mathsf{derelict}(y)). \end{array}$$

These two terms can be distinguished by the boolean context

$$\mathscr{C} \overset{\text{def}}{=} \mathsf{let}\ \bullet\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathsf{discard}\ x\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathtt{true}.$$

as $\mathscr{C}[|\Omega|^\circ] \Uparrow$ but $\mathscr{C}[|\langle \mathsf{fst}(\Omega), \mathsf{snd}(\Omega)\rangle|^\circ] \Downarrow \mathtt{true}$. $\square$

Rather than translate PCF pairs into (linear) multiplicative pairs, we could use additive pairs and change the translation to

$$\begin{array}{rcl} (\sigma \times \tau)^\circ & \overset{\text{def}}{=} & \sigma^\circ \& \tau^\circ,\ \text{and} \\ |\Gamma \triangleright \langle e, f\rangle : \sigma \times \tau|^\circ & \overset{\text{def}}{=} & !\Gamma^\circ \triangleright \langle |e|^\circ, |f|^\circ\rangle : \sigma^\circ \& \tau^\circ \\ |\Gamma \triangleright \mathsf{fst}(e) : \sigma|^\circ & \overset{\text{def}}{=} & !\Gamma^\circ \triangleright \mathsf{fst}(|e|^\circ) : \sigma^\circ \\ |\Gamma \triangleright \mathsf{snd}(e) : \tau|^\circ & \overset{\text{def}}{=} & !\Gamma^\circ \triangleright \mathsf{snd}(|e|^\circ) : \tau^\circ. \end{array}$$

In fact, this was the original translation given by Girard. The counter-example to full abstraction given above would now be translated as

$$\begin{array}{rcl} |\Omega|^\circ & \overset{\text{def}}{=} & \Omega^{\mathsf{bool}\&\mathsf{bool}},\ \text{and} \\ |\langle \mathsf{fst}(\Omega), \mathsf{snd}(\Omega)\rangle|^\circ & \overset{\text{def}}{=} & \langle \mathsf{fst}(\Omega), \mathsf{snd}(\Omega)\rangle. \end{array}$$

These two (translated) terms are easily seen to be applicatively similar. This translation can be proven to be adequate, but it is still an open question whether it is fully abstract.

7 Conclusions

If linear functional programming languages are to be used by programmers, then these programmers need means to reason about their programs. In this paper, we have given some reasoning principles based upon the operational behaviour of programs. Although these techniques are not new, their application to the linear setting is novel, and some surprises have arisen because of the inherent linearity – primarily the attention required to define linPCF contexts. All in all, the techniques of Howe (1996), Pitts (1997) and others, seem applicable to linear functional languages.

Crole (1996) has (independently) applied Howe's method to a small fragment of linPCF– in fact, just the fragment including multiplicative pairs and linear implication. However, an important difference is Crole's choice of a non-strict evaluation strategy for his linear language. It is unclear whether all of the proof techniques detailed in the appendix can be easily applied to the non-strict setting (see the comment in the proof of Proposition 14). Clearly, this is future work.

Benton and Wadler (1996) have shown that both Girard translations are related to Moggi's translations of the λ-calculus into the computational λ-calculus. We have been unable to find analogous work considering full abstraction and adequacy for Moggi's calculus. One would hope that Benton and Wadler's work could be used to derive these results from the work in this paper. Maraist *et al.* (1995) have also considered the Girard translations, but only with respect to term reduction and for a different language (one without recursion, for example).

An important piece of future work is to investigate the notion of *ground* contextual equivalence, that is where we only make observations at the bool type.

Definition 7

Given $\Gamma \triangleright M : \phi$ and $\Gamma \triangleright N : \phi$, M is said to *ground contextually refine* N, written $\Gamma \triangleright M \sqsubseteq_{gnd} N : \phi$, iff for all closing boolean contexts, $\bullet(\Gamma) : \phi \triangleright \mathscr{C} : \mathsf{bool}$, if $\mathscr{C}[M] \Downarrow \mathtt{true}$ then $\mathscr{C}[N] \Downarrow \mathtt{true}$.

Ground contextual equivalence, written $\Gamma \triangleright M \approx_{gnd} N : \phi$, holds iff $\Gamma \triangleright M \sqsubseteq_{gnd} N : \phi$ and $\Gamma \triangleright N \sqsubseteq_{gnd} M : \phi$.

The question is whether this coincides with contextual equivalence (Definition 1). We have been unable to verify this, and moreover we conjecture that it does not. Consider the terms

$$\lambda x : !\mathsf{bool}.\mathsf{discard}\ x\ \mathsf{in}\ \Omega^{\mathsf{bool}} \qquad \text{and} \qquad \Omega^{!\mathsf{bool} \multimap \mathsf{bool}}$$

Clearly, these terms are not contextually equivalent: the empty context distinguishes them. However can a context of type bool distinguish them? Such a context either entirely discards the term (in which case they are observably the same), or uses the term, i.e. it is applied to an argument (in which case they both fail to terminate and are observably the same). Any other alternative seems excluded by the linear type system. A coinductive characterisation of ground contextual equivalence, which will of course settle this point, is work in progress.

To gauge the usefulness of linear functional languages requires much more practical experience. Mackie's lilac system (Mackie, 1994) is clearly an important step. The possibility of using a linear intermediate language (extended with impredicative polymorphism) inside a compiler is the subject of on-going research.

Acknowledgements

This work was carried out whilst I was at the University of Cambridge Computer Laboratory and supported by EPSRC Grant GR/M04716 and Gonville and Caius College, Cambridge. I am grateful to Nick Benton, Andrew Gordon, Martin Hyland, Søren Lassen and Andrew Pitts for many useful discussions. In particular, I should like to thank Søren Lassen for many insightful comments on an earlier draft. Finally, I should like to thank the anonymous referees who made a number of useful suggestions to improve this paper.

A Proof of Theorem 1

The proof that contextual refinement coincides with applicative similarity essentially splits in two parts. First to show that applicative similarity is a precongruence, i.e. compatible (a relation which respects the term formation rules) and transitive; and secondly that contextual refinement is a simulation. To prove the former we shall adopt an ingenious method due to Howe (1996). One defines an auxiliary relation, which is trivially compatible and, rather less trivially, is an applicative simulation.

Definition 8
The Howe relation, $\leqslant^{\star}$, between two well-typed expressions is defined in figure A 1.

An important property of this relation is the following:

Lemma 1
If $\Gamma \triangleright M \leqslant^{\star} N : \phi$ and $\Gamma \triangleright N \leqslant^{\circ} P : \phi$ then $\Gamma \triangleright M \leqslant^{\star} P : \phi$.

Proof
By induction over $\Gamma \triangleright M \leqslant^{\star} N : \phi$ and transitivity of $\leqslant^{\circ}$. □

One direction of the equivalence between applicative similarity and the Howe relation is now immediate.

Proposition 10
If $\Gamma \triangleright M \leqslant^{\circ} N : \phi$ then $\Gamma \triangleright M \leqslant^{\star} N : \phi$.

Proof
It is clear that $\Gamma \triangleright M \leqslant^{\star} M : \phi$ (reflexivity) and, by assumption, that $\Gamma \triangleright M \leqslant^{\circ} N : \phi$. From Lemma 1 we conclude $\Gamma \triangleright M \leqslant^{\star} N : \phi$. □

It is also relatively straightforward to prove the following:

Lemma 2
If $\emptyset \triangleright v \leqslant^{\star} v' : \phi$ and $\Gamma, x : \phi \triangleright M \leqslant^{\star} M' : \psi$ then $\Gamma \triangleright M[x := v] \leqslant^{\star} M'[x := v'] : \psi$.

Proof
By induction on the judgement $\Gamma, x : \phi \triangleright M \leqslant^{\star} M' : \psi$. □

To prove an equivalence between $\leqslant^{\star}$ and applicative similarity one needs to prove the following vital lemma.

$$\begin{array}{rcl}
x\colon\phi \rhd x \leqslant^\star N\colon\phi & \text{iff} & x\colon\phi \rhd x \leqslant^\circ N\colon\phi \\
\emptyset \rhd b \leqslant^\star N\colon\mathsf{bool} & \text{iff} & b \leqslant N\colon\mathsf{bool} \\
\Gamma \rhd \lambda x\colon\phi.M \leqslant^\star N\colon\phi\multimap\psi & \text{iff} & \exists M'.\Gamma, x\colon\phi \rhd M \leqslant^\star M'\colon\psi \text{ and} \\
 & & \Gamma \rhd \lambda x\colon\phi.M' \leqslant^\circ N\colon\phi\multimap\psi \\
\Gamma,\Delta \rhd (M_1 M_2) \leqslant^\star N\colon\psi & \text{iff} & \exists M_1', M_2'.\Gamma \rhd M_1 \leqslant^\star M_1'\colon\phi\multimap\psi, \\
 & & \Delta \rhd M_2 \leqslant^\star M_2'\colon\phi \text{ and} \\
 & & \Gamma,\Delta \rhd (M_1' M_2') \leqslant^\circ N\colon\psi \\
\Gamma,\Delta \rhd M_1 \otimes M_2 \leqslant^\star N\colon\phi\otimes\psi & \text{iff} & \exists M_1', M_2'.\Gamma \rhd M_1 \leqslant^\star M_1'\colon\phi, \\
 & & \Delta \rhd M_2 \leqslant^\star M_2'\colon\psi \text{ and} \\
 & & \Gamma,\Delta \rhd M_1'\otimes M_2' \leqslant^\circ N\colon\phi\otimes\psi \\
\Gamma,\Delta \rhd \mathsf{let}\ M_1\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ M_2 \leqslant^\star N\colon\varphi & \text{iff} & \exists M_1', M_2'.\Gamma \rhd M_1 \leqslant^\star M_1'\colon\phi\otimes\psi, \\
 & & \Delta, x\colon\phi, y\colon\psi \rhd M_2 \leqslant^\star M_2'\colon\varphi \text{ and} \\
 & & \Gamma,\Delta \rhd \mathsf{let}\ M_1'\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ M_2' \leqslant^\circ N\colon\varphi \\
\Gamma \rhd \langle M_1, M_2\rangle \leqslant^\star N\colon\phi\&\psi & \text{iff} & \exists M_1', M_2'.\Gamma \rhd M_1 \leqslant^\star M_1'\colon\phi, \\
 & & \Gamma \rhd M_2 \leqslant^\star M_2'\colon\psi \text{ and} \\
 & & \Gamma \rhd \langle M_1', M_2'\rangle \leqslant^\circ N\colon\phi\&\psi \\
\Gamma \rhd \mathsf{fst}(M) \leqslant^\star N\colon\phi & \text{iff} & \exists M'.\Gamma \rhd M \leqslant^\star M'\colon\phi\&\psi \text{ and} \\
 & & \Gamma \rhd \mathsf{fst}(M') \leqslant^\circ N\colon\phi \\
\Gamma \rhd \mathsf{snd}(M) \leqslant^\star N\colon\psi & \text{iff} & \exists M'.\Gamma \rhd M \leqslant^\star M'\colon\phi\&\psi \text{ and} \\
 & & \Gamma \rhd \mathsf{snd}(M') \leqslant^\circ N\colon\psi \\
\Gamma,\Delta \rhd \mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3 \leqslant^\star N\colon\phi & \text{iff} & \exists M_1', M_2', M_3'.\Gamma \rhd M_1 \leqslant^\star M_1'\colon\mathsf{bool}, \\
 & & \Delta \rhd M_2 \leqslant^\star M_2'\colon\phi, \\
 & & \Delta \rhd M_3 \leqslant^\star M_3'\colon\phi \text{ and} \\
 & & \Gamma,\Delta \rhd \mathsf{if}\ M_1'\ \mathsf{then}\ M_2'\ \mathsf{else}\ M_3' \leqslant^\circ N\colon\phi \\
\Gamma \rhd \mathsf{derelict}(M) \leqslant^\star N\colon\phi & \text{iff} & \exists M'.\Gamma \rhd M \leqslant^\star M'\colon !\phi \text{ and} \\
 & & \Gamma \rhd \mathsf{derelict}(M') \leqslant^\circ N\colon\phi \\
\Gamma_1,\ldots,\Gamma_n \rhd \mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N \leqslant^\star P\colon !\psi & \text{iff} & \exists \vec{M'}, N'.\Gamma_i \rhd M_i \leqslant^\star M_i'\colon !\phi_i, \\
 & & x_i\colon !\phi_1,\ldots,x_n\colon !\phi_n \rhd N \leqslant^\star N'\colon\psi \text{ and} \\
 & & \Gamma_1,\ldots,\Gamma_n \rhd \mathsf{promote}\ \vec{M'}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N' \leqslant^\circ P\colon !\psi \\
\Gamma,\Delta \rhd \mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N \leqslant^\star P\colon\psi & \text{iff} & \exists M'.N'.\Gamma \rhd M \leqslant^\star M'\colon !\phi, \\
 & & \Delta, x\colon !\phi, y\colon !\phi \rhd N \leqslant^\star N'\colon\psi \text{ and} \\
 & & \Gamma,\Delta \rhd \mathsf{copy}\ M'\ \mathsf{as}\ x, y\ \mathsf{in}\ N' \leqslant^\circ P\colon\psi \\
\Gamma,\Delta \rhd \mathsf{discard}\ M\ \mathsf{in}\ N \leqslant^\star P\colon\psi & \text{iff} & \exists M', N'.\Gamma \rhd M \leqslant^\star M'\colon !\phi, \\
 & & \Delta \rhd N \leqslant^\star N'\colon\psi \text{ and} \\
 & & \Gamma,\Delta \rhd \mathsf{discard}\ M'\ \mathsf{in}\ N' \leqslant^\circ P\colon\psi \\
\Gamma_1,\ldots,\Gamma_n \rhd \mathsf{rec}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y.N \leqslant^\star P\colon\psi & \text{iff} & \exists \vec{M'}, N'.\Gamma_i \rhd M_i \leqslant^\star M_i'\colon !\phi_i, \\
 & & x_i\colon !\phi_1,\ldots,x_n\colon !\phi_n, y\colon !\psi \rhd N \leqslant^\star N'\colon\psi \text{ and} \\
 & & \Gamma_1,\ldots,\Gamma_n \rhd \mathsf{rec}\ \vec{M'}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ y.N' \leqslant^\circ P\colon\psi
\end{array}$$

Fig. A 1. Howe relation for linPCF.

Lemma 3
If $\emptyset \rhd M \leqslant^\star N\colon\phi$ and $M \Downarrow v$ then $\exists v'$ such that $N \Downarrow v'$ and $\emptyset \rhd v \leqslant^\star v'\colon\phi$.

Proof
By induction on the derivation of $M \Downarrow v$. Two example cases are the following:

1. $\mathsf{promote}\ M\ \mathsf{for}\ x\ \mathsf{in}\ Q \Downarrow \mathsf{promote}\ v\ \mathsf{for}\ x\ \mathsf{in}\ Q$: By assumption $\exists M', Q'.\emptyset \rhd M \leqslant^\star M'\colon !\psi$ and $x\colon !\psi \rhd Q \leqslant^\star Q'\colon\phi$ and $\mathsf{promote}\ M'\ \mathsf{for}\ x\ \mathsf{in}\ Q' \leqslant N\colon !\phi$. By induction we have $\exists v'.M' \Downarrow v'$ and $\emptyset \rhd v \leqslant^\star v'\colon\phi$. We can deduce that $\mathsf{promote}\ M'\ \mathsf{for}\ x\ \mathsf{in}\ Q' \Downarrow \mathsf{promote}\ v'\ \mathsf{for}\ x\ \mathsf{in}\ Q'$ and hence it follows that $N \Downarrow w$ and $\mathsf{promote}\ v'\ \mathsf{for}\ x$

in $Q' \leqslant w\!:!\phi$. From Proposition 10 we can conclude $\emptyset \triangleright \mathsf{promote}\ v\ \mathsf{for}\ x\ \mathsf{in}\ Q \leqslant^\star w\!:!\phi$ and we are done.

2. $\mathsf{derelict}(M) \Downarrow v$: By assumption $\exists M'.\emptyset \triangleright M \leqslant^\star M'\!:!\phi$ and $\mathsf{derelict}(M') \leqslant N\!:\phi$. By induction we have $\exists v''.M' \Downarrow v''$ and $\emptyset \triangleright \mathsf{promote}\ v'\ \mathsf{for}\ x\ \mathsf{in}\ P \leqslant^\star v''\!:!\phi$. By definition $\exists w, P'.\emptyset \triangleright v' \leqslant^\star w\!:!\psi$ and $x\!:!\psi \triangleright P \leqslant^\star P'\!:\phi$ and $\mathsf{promote}\ w\ \mathsf{for}\ x\ \mathsf{in}\ P' \leqslant v''\!:!\phi$. By Lemma 2 we have that $\emptyset \triangleright P[x := v'] \leqslant^\star P'[x := w]\!:\phi$. By determinacy of evaluation, we have that $v'' \equiv \mathsf{promote}\ w''\ \mathsf{for}\ y\ \mathsf{in}\ Q$ and then as $\mathsf{promote}\ v'\ \mathsf{for}\ x\ \mathsf{in}\ P' \leqslant \mathsf{promote}\ w''\ \mathsf{for}\ y\ \mathsf{in}\ Q\!:!\phi$ we can conclude that $w \leqslant w''\!:!\psi$ and $P'[x := w] \leqslant Q[y := w'']\!:\phi$. From Lemma 1 we have that $\emptyset \triangleright P[x := v'] \leqslant^\star Q[y := w']\!:\phi$ and then by induction $\exists a.Q[y := w'] \Downarrow a$ and $\emptyset \triangleright v \leqslant^\star a\!:\phi$. We can now conclude that $\mathsf{derelict}(M') \Downarrow a$, and hence that $N \Downarrow c$ and $a \leqslant c\!:\phi$. From $\emptyset \triangleright v \leqslant^\star a\!:\phi$ and $a \leqslant c\!:\phi$ we can conclude that $\emptyset \triangleright v \leqslant^\star c\!:\phi$ and we are done.

$\square$

One can now prove the other direction of the equivalence between applicative similarity and the Howe relation.

Proposition 11
If $\Gamma \triangleright M \leqslant^\star N\!:\phi$ then $\Gamma \triangleright M \leqslant^\circ N\!:\phi$.

Proof
Form the type-indexed family $\mathscr{S} = \mathscr{S}_\phi \stackrel{\text{def}}{=} \{(M, N) \mid \emptyset \triangleright M \leqslant^\star N\!:\phi\}$, and show that $\mathscr{S} \subseteq [\mathscr{S}]$, which holds essentially by Lemma 3. Thus we have that $\emptyset \triangleright M \leqslant^\star N\!:\phi$ implies $M \leqslant N\!:\phi$. We have that $\forall v.\emptyset \triangleright v \leqslant^\star v\!:\psi$, hence given any open terms $\vec{x}\!:\Gamma \triangleright M \leqslant^\star N\!:\phi$ we have by Lemma 1 that $\emptyset \triangleright M[\vec{x} := \vec{v}] \leqslant^\star N[\vec{x} := \vec{v}]\!:\phi$ and then we can invoke the above reasoning for the resulting closed terms. $\square$

It is almost immediate by its definition that $\leqslant^\star$ is compatible, which given the equivalence contained in Lemma 1 and Proposition 11, and the fact that $\leqslant$ is transitive, immediately yields the following proposition:

Proposition 12
$\leqslant^\circ$ is a precongruence.

Thus, the first half of our strategy is complete. It is now possible to show that applicative similarity implies contextual refinement.

Proposition 13
If $\Gamma \triangleright M \leqslant^\circ N\!:\phi$ then $\Gamma \triangleright M \sqsubseteq N\!:\phi$.

Proof
Suppose that $\Gamma \triangleright M \leqslant^\circ N\!:\phi$. As $\leqslant^\circ$ is a precongruence, we have that for any closing linear context $\bullet(\Gamma)\!:\phi \triangleright \mathscr{C}\!:\psi$ that $\mathscr{C}[M] \leqslant \mathscr{C}[N]\!:\psi$. This, by definition, that if $\mathscr{C}[M] \Downarrow v$ then $\exists v'$ such that $\mathscr{C}[N] \Downarrow v'$, and so we are done. $\square$

Corollary 2
If $\Gamma \triangleright M \leqslant^\circ N\!:\phi$ then $\Gamma \triangleright M \sqsubseteq_{gnd} N\!:\phi$.

Two important properties of contextual refinement are the following:

Proposition 14
If $M \sqsubseteq N : \phi$ then $M \leqslant N : \phi$.

Proof
Form the type-indexed family $\mathscr{S} = \mathscr{S}_\phi \stackrel{\text{def}}{=} \{(M,N) \mid M \sqsubseteq N : \phi\}$, and show that $\mathscr{S} \subseteq [\mathscr{S}]$. Thus take M and N such that $M \sqsubseteq N : \phi$ and $M \Downarrow v$. Using the definition of contextual refinement, one can consider the identity context, and by definition there exists a v' such that $N \Downarrow v'$. We show that $v \langle \mathscr{S} \rangle_\phi \ v'$ by case analysis on the type ϕ.[5]

1. ($\phi \equiv \mathsf{bool}$) Build the context $\mathscr{C}$, which is defined as
$$\bullet(\emptyset) : \mathsf{bool} \rhd \mathsf{if} \ \bullet \ \mathsf{then}\ \mathtt{true}\ \mathsf{else}\ \Omega : \mathsf{bool}.$$
Thus, $M \Downarrow \mathtt{true}$ iff $\mathscr{C}[M] \Downarrow$ iff $\mathscr{C}[N] \Downarrow$ iff $N \Downarrow \mathtt{true}$, and we are done. The case for $M \Downarrow \mathtt{false}$ is similar.
2. ($\phi \equiv \phi \otimes \psi$) We have that $M \Downarrow v \otimes w$ and $N \Downarrow v' \otimes w'$. Take any context $\bullet(\emptyset) : \phi \rhd \mathscr{C} : \varphi$
$$\begin{aligned}
\mathscr{C}[v] \Downarrow \quad & \Longleftrightarrow \quad \mathscr{C}[v] \otimes w \Downarrow \\
& \Longleftrightarrow \quad \mathsf{let}\ v \otimes w\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathscr{C}[x] \otimes y \Downarrow \\
& \Longleftrightarrow \quad \mathsf{let}\ M\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathscr{C}[x] \otimes y \Downarrow \\
& \Longleftrightarrow \quad \mathsf{let}\ N\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathscr{C}[x] \otimes y \Downarrow \\
& \Longleftrightarrow \quad \mathsf{let}\ v' \otimes w'\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathscr{C}[x] \otimes y \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}[v'] \otimes w' \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}[v'] \Downarrow .
\end{aligned}$$
Similar reasoning satisfies the case for w.[6]
3. ($\phi \equiv \phi \multimap \psi$) We have that $M \Downarrow \lambda x.P$ and $N \Downarrow \lambda x.Q$. Take any context $\bullet(\emptyset) : \psi \rhd \mathscr{C} : \varphi$ and call $\mathscr{C}'$ the context which results from replacing the occurrence of the hole $\bullet$ with the context $\bullet(\emptyset) : \phi \multimap \psi \rhd (\bullet v) : \psi$. Thus,
$$\begin{aligned}
\mathscr{C}[P[x := v]] \Downarrow \quad & \Longleftrightarrow \quad \mathscr{C}[(\lambda x.P)v] \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}'[\lambda x.P] \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}'[M] \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}'[N] \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}'[\lambda x.Q] \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}[(\lambda x.Q)v] \Downarrow \\
& \Longleftrightarrow \quad \mathscr{C}[Q[x := v]] \Downarrow .
\end{aligned}$$

[5] In Bierman (1997) it was incorrectly claimed that this proof was by induction over the structure of the type ϕ.
[6] This proof relies on the strict nature of the multiplicative pair. Had it been defined to be non-strict (as in Crole, 1996), then it is unclear how to complete this proof.

4. ($\phi \equiv \phi \& \psi$) We have that $M \Downarrow \langle P_1, P_2 \rangle$ and $N \Downarrow \langle P_1', P_2' \rangle$. Take any context $\bullet(\emptyset) : \phi \triangleright \mathscr{C} : \varphi$ and call $\mathscr{C}'$ the context which results from replacing the occurrence of the hole $\bullet$ with the context $\bullet(\emptyset) : \phi \& \psi \triangleright \mathsf{fst}(\bullet) : \phi$. Thus

$$\begin{array}{rcl} \mathscr{C}[P_1] \Downarrow & \Longleftrightarrow & \mathscr{C}[\mathsf{fst}(\langle P_1, P_2 \rangle)] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[\langle P_1, P_2 \rangle] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[M] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[N] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[\langle P_1', P_2' \rangle] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}[\mathsf{fst}(\langle P_1', P_2' \rangle)] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}[P_1'] \Downarrow . \end{array}$$

Similar reasoning satisfies the case for P_2.

5. ($\phi \equiv !\phi$) Thus we have that $M \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ P$ and $N \Downarrow \mathsf{promote}\ \vec{v}'\ \mathsf{for}\ \vec{x}'$ in Q. Take any context $\bullet(\emptyset) : \phi \triangleright \mathscr{C} : \psi$ and call $\mathscr{C}'$ the context which results from replacing the hole $\bullet$ with the context $\bullet(\emptyset) : !\phi \triangleright \mathsf{derelict}(\bullet) : \phi$. Thus

$$\begin{array}{rcl} \mathscr{C}[P[\vec{x} := \vec{v}]] \Downarrow & \Longleftrightarrow & \mathscr{C}[\mathsf{derelict}(\mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ P)] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[\mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ P] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[M] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[N] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[\mathsf{promote}\ \vec{v}'\ \mathsf{for}\ \vec{x}'\ \mathsf{in}\ Q] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}[\mathsf{derelict}(\mathsf{promote}\ \vec{v}'\ \mathsf{for}\ \vec{x}'\ \mathsf{in}\ Q)] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}[Q[\vec{x}' := \vec{v}']] \Downarrow . \end{array}$$

□

Proposition 15
If $\Gamma, x : \phi \triangleright M \sqsubseteq N : \psi$ then $\Gamma \triangleright M[x := v] \sqsubseteq N[x := v] : \psi$ for any $\emptyset \triangleright v : \phi$.

Proof
Assume that $\Gamma, x : \phi \triangleright M \sqsubseteq N : \psi$. Then for a given context $\bullet(\Gamma) : \psi \triangleright \mathscr{C} : \varphi$, call $\mathscr{C}'$ the context which results from replacing the hole $\bullet$ with the context $\bullet(\Gamma, x : \phi) : \psi \triangleright \mathscr{C}' : \varphi)$.

$$\begin{array}{rcl} \mathscr{C}[M[x := v]] \Downarrow & \Longleftrightarrow & \mathscr{C}[(\lambda x.M)v] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[M] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}'[N] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}[(\lambda x.N)v] \Downarrow \\ & \Longleftrightarrow & \mathscr{C}[N[x := v]] \Downarrow \end{array}$$

□

These enable us to prove the following implication:

Proposition 16
If $\Gamma \triangleright M \sqsubseteq N : \phi$ then $\Gamma \triangleright M \lesssim^{\circ} N : \phi$.

Proof
By definition $x_1 : \psi_1, \ldots, x_n : \psi_n \triangleright M \leqslant^\circ N : \phi$ iff $M[\vec{x} := \vec{v}] \leqslant N[\vec{x} := \vec{v}] : \phi$ for values v_i of type ψ_i. From Lemma 15 we have that $M[\vec{x} := \vec{v}] \sqsubseteq N[\vec{x} := \vec{v}] : \phi$, and then we can apply Proposition 14 to get $M[\vec{x} := \vec{v}] \leqslant N[\vec{x} := \vec{v}] : \phi$ and we are done. □

Thus, contextual equivalence and applicative bisimilarity coincide.

Corollary 3
$\Gamma \triangleright M \approx N : \phi$ iff $\Gamma \triangleright M \approx^\circ_{app} N : \phi$.

References

Abramsky, S. (1990) The lazy lambda calculus. In: Turner, D. A. (ed), *Research Topics in Functional Programming*, pp. 65–116. Addison-Wesley.

Abramsky, S. (1993) Computational interpretations of linear logic. *Theor. Comput. Sci.* **111**(1–2), 3–57.

Barendsen, E. and Smetsers, S. (1996) Uniqueness typing for functional languages with graph rewriting semantics. *Math. Struct. in Comput. Sci.* **6**(6), 579–612.

Benton, P. N. and Wadler, P. (1996) Linear logic, monads and the lambda calculus. *Proc. Symposium on Logic in Computer Science*, pp. 420–431.

Benton, P. N., Bierman, G. M., de Paiva, V. C. V. and Hyland, J. M. E. (1993) A term calculus for intuitionistic linear logic. In: Bezem, M. and Groote, J. F. (eds.), *Proc. 1st Int. Conf. on Typed λ-calculi and Applic.*, pp. 75–90. *Lecture Notes in Computer Science*. **664**. Springer-Verlag.

Benton, P. N., Kennedy, A. J. and Russell, G. (1998) Compiling Standard ML to Java bytecodes. *Proc. Int. Conf. on Functional Programming*. (*ACM SIGPLAN Notices*, **34**(1), 129–140, January 1999.)

Bierman, G. M. (1995) What is a categorical model of intuitionistic linear logic? *Proc. 2nd Int. Conf. on Typed λ-calculi and Applic.*, pp. 78–93. *Lecture Notes in Computer Science*, **902**. Springer-Verlag.

Bierman, G. M. (1997) *Observations on a linear PCF.* Technical Report 412, Computer Laboratory, University of Cambridge.

Braüner, T. (1994) The Girard translation extended with recursion. *Proc. Conf. on Computer Science Logic*, pp. 31–45. *Lecture Notes in Computer Science*, **933**. Springer-Verlag.

Braüner, T. (1996) *An axiomatic approach to adequacy*. PhD thesis, Department of Computer Science, University of Århus, Denmark. (Available as BRICS Technical Report DS–96–4.)

Braüner, T. (1997) A general adequacy result for a linear functional language. *Theor. Comput. Sci.*, **177**(1), 27–58.

Chirimar, J., Gunter, C. A. and Riecke, J. G. (1996) Reference counting as a computational interpretation of linear logic. *J. Functional Programming*, **6**(2), 195–244.

Crole, R.L. (1996) How linear is Howe? In: McCusker, G., Edalat, A. and Jourdan, S. (eds.), *Advances in Theory and Formal Methods*, pp. 60–72. Imperial College Press.

Di Cosmo, R. (1993) Deciding type isomorphisms in a type-assignment framework. *J. Functional Programming*, **3**(4), 485–525.

Di Cosmo, R. (1995) *Isomorphisms of Types: From λ-calculus to information retrieval and language design*. Progress in Theoretical Computer Science. Birkhauser.

Girard, J.-Y. (1987) Linear logic. *Theor. Comput. Sci.*, **50**, 1–101.

Gordon, A.D. (1995) Bisimilarity as a theory of functional programming. *11th Ann. Conf. on Mathematical Foundations of Programming Semantics*. Electronic Notes in Theoretical Computer Science. Elsevier.

Gordon, A. D. and Pitts, A. M. (eds.) (1998) *Higher Order Operational Techniques in Semantics*. Publications of the Newton Institute. Cambridge University Press.

Hashimoto, M. and Ohori, A. (1996) *A typed context calculus*. Technical Report RIMS–1098, Research Institute for Mathematical Sciences, Kyoto University.

Holmström, S. (1988) A linear functional language. *Proc. Workshop on Implementation of Lazy Functional Languages*, pp. 13–32. Aspenäs, Sweden. Programming Methodology Group, University of Göteborg and Chalmers University of Technology. PMG Tech Report 53.

Holmström, S. (1989) *Quicksort in a linear functional language*. Technical Report 65, Programming Methodology Group, University of Göteborg and Chalmers University of Technology.

Howe, D. J. (1996) Proving congruence of bisimulation in functional programming languages. *Infor. & Control*, **124**(2), 103–112.

Mackie, I. (1994) Lilac: A functional programming language based on linear logic. *J. Functional Programming*, **4**(4), 1–39.

Maraist, J., Odersky, M., Turner, D. N. and Wadler, P. (1995) Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Proc. Conf. on Mathematical Foundations of Programming Semantics*. Electronic Notes in Theoretical Computer Science. Elsevier.

Morrisett, G. (1995) *Compiling with types*. PhD thesis, School of Computer Science, Carnegie Mellon University. (Available as Technical Report CMU–CS–95–226.)

Pitts, A. M. (1994) *Some notes on inductive and co-inductive techniques in the semantics of functional languages*. Technical Report BRICS-NS-94-5, BRICS, Department of Computer Science, University of Århus.

Pitts, A. M. (1997) Operationally-based theories of program equivalence. In: Dybjer, P. and Pitts, A. M. (eds.), *Semantics and Logics of Computation*, pp. 241–298. Publications of the Newton Institute. Cambridge University Press.

Plotkin, G. D. (1977) LCF considered as a programming language. *Theor. Comput. Sci.*, **5**, 223–255.

Rittri, M. (1991) Using types as search keys in function libraries. *J. Functional Programming*, **1**(1), 71–89.

Seely, R. A. G. (1989) Linear logic, *-autonomous categories and cofree algebras. In: *Conference on Categories in Computer Science and Logic*, pp. 371–382.

Wadler, P. (1990) Linear types can change the world! In: Broy, M. and Jones, C. (eds.), *Programming Concepts and Methods*. North Holland.

Wadler, P. (1991) Is there a use for linear logic? *Proc. Symposium on Partial Evaluation and Semantics based Program Manipulation*. (In *ACM SIGPLAN Notices*, **26**(9), 255–273.)

Wakeling, D. (1990) *Linearity and laziness*. PhD thesis, University of York. (Available as Technical Report *YCST 90/07*.)

Wakeling, D. and Runciman, C. (1991) Linearity and laziness. *Proc. Conference on Functional Programming Languages and Computer Architecture*, pp. 215–240. *Lecture Notes in Computer Science*, **523**. Springer-Verlag.

Winskel, G. (1993) *The Formal Semantics of Programming Languages: An introduction*. MIT Press.