Technical Report

UCAM-CL-TR-412
ISSN 1476-2986

Number 412

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Observations on a linear PCF (preliminary report)

G.M. Bierman

January 1997

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Observations on a Linear PCF (Preliminary Report)

G.M. BIERMAN

Gonville and Caius College,
Cambridge. England.

ABSTRACT

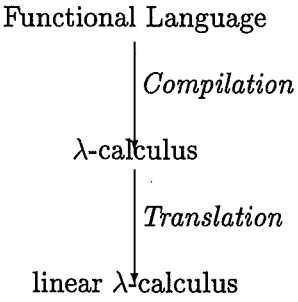
This paper considers some theoretical and practical issues concerning the use of linear logic as a logical foundation of functional programming languages such as Haskell and SML. First I give an operational theory for a linear PCF: the (typed) linear λ-calculus extended with booleans, conditional and non-termination. An operational semantics is given which corresponds in a precise way to the process of β-reduction which originates from proof theory. Using this operational semantics I define notions of observational equivalence (sometimes called contextual equivalence). Surprisingly, the linearity of the language forces a reworking of the traditional notion of a context (the details are given in an appendix). A co-inductively defined notion, applicative bisimularity, is developed and compared with observational equivalence using a variant of Howe's method. Interestingly the equivalence of these two notions is greatly complicated by the linearity of the language. These equivalences are used to study a call-by-name translation of PCF into linear PCF. It is shown that this translation is adequate but not fully abstract. Finally I show how Landin's SECD machine can be adapted to execute linear PCF programs.

1 INTRODUCTION

Since its inception, Girard's linear logic [11] has promised to give a refined view of computation due to its resource-conscious nature. The intuitionistic fragment yields, via the Curry-Howard correspondence, a (typed) linear λ-calculus [3, 5]; where linearity means that variables occur exactly once and, consequently, there are explicit operations to discard and duplicate terms. An important result is that there are several ways of translating the (typed) λ-calculus (the foundation of functional programming languages) into the linear λ-calculus. Semantically this has proved a very useful viewpoint—rather than devising a model of the λ-calculus one can, in its stead, produce a model of the linear λ-calculus [6]. This approach has been utilised, for example, by Plotkin in studying recursion and parametricity [21] and by Abramsky *et al.* [2] to produce fully abstract models of PCF.

A more operational perspective is to consider the linear λ-calculus as some sort of intermediate language to which the λ-calculus is compiled.[1] This has an obvious practical advantage in that the linear calculus is explicit about its manipulation of data and so possible optimisations should be expressible as simple term rewrites. The picture in mind is

[1] Maybe a more fashionable description is to say that the linear λ-calculus can be thought of as a computational *metalanguage*.



where the first step traditionally occurs in functional language compilers [15]. At the level of the λ-calculus we normally say that an optimisation is the replacement of a subterm M with another, N, which we permit only if they are *observationally equivalent*, *viz.* if no matter where they are placed in a program we can not tell them apart. Thus if we are including an extra stage of translation to the linear λ-calculus we need not only to develop a theory of observational equivalence for the linear calculus, but also to consider to what extent the translation process preserves this equivalence. This paper represents a first step in that direction.

This paper is organised as follows. In §2 I give the syntax and operational semantics for a linear PCF. I consider the question of a formal definition of a context for the linear calculus and show that traditional treatments for non-linear calculi are inadequate. I give a more precise definition which is suitable for the linear calculus. Using this definition I give two versions of observational equivalence: one where observations are made only at boolean type (ground) and one where observations are made at all types (lazy). I then give a co-inductive definition of program equivalence known as applicative (bi)simularity. Employing a variant of Howe's method I show that applicative bisimularity coincides with lazy observational equivalence. In §3.1 I give the syntax of PCF and also a translation of terms from linear PCF to PCF. In §3.2 I give a call-by-name operational semantics for PCF and also recall standard definitions of observational equivalence and applicative bisimulation. In §4 I give a translation of terms from (call-by-name) PCF to linear PCF and show that the translation is adequate but *not* fully abstract. In §5 I show how Landin's SECD machine can be adapted to execute linear PCF-terms. I conclude, in §6, with an indication of future work.

2 A Linear PCF

The core corresponds via the Curry-Howard correspondence to the natural deduction formulation of the $(\otimes, \multimap, !)$-fragment of Intuitionistic Linear Logic (**ILL**). For the purposes of this paper the core calculus has been extended with booleans, a conditional operator and non-terminating constants (at all types) to yield a simple linear functional programming language, which we refer to as linear PCF.

Types are given by the grammar

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

and raw terms are then given by the grammar

$$
\begin{array}{rcll}
M & ::= & \mathtt{true}, \mathtt{false} & \text{Booleans} \\
& | & x & \text{Variable} \\
& | & \lambda x{:}\,\phi.M & \text{Abstraction} \\
& | & MM & \text{Application} \\
& | & M \otimes M & \text{Multiplicative Pair} \\
& | & \mathsf{let}\ M\ \mathsf{be}\ x \otimes x\ \mathsf{in}\ M & \text{Split} \\
& | & \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M & \text{Conditional} \\
& | & \mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ M & \text{Promote} \\
& | & \mathsf{derelict}(M) & \text{Derelict} \\
& | & \mathsf{discard}\ M\ \mathsf{in}\ M & \text{Discarding} \\
& | & \mathsf{copy}\ M\ \mathsf{as}\ x, x\ \mathsf{in}\ M & \text{Duplication} \\
& | & \Omega^{\phi} & \text{Non-termination;}
\end{array}
$$

where x is 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 multiset of (variable,type)-pairs. In this paper we shall only consider well-typed terms, i.e. those for which there is a valid typing judgement. The rules for forming typing judgements are given in Figure 1. A term M containing no free variables (i.e. $\emptyset \triangleright M{:}\,\phi$) is said to be *closed* (otherwise it is said to be *open*). The set of linear PCF-terms which can be assigned the type ϕ given Γ shall be written $Exp_{\Gamma}(\phi)$. If the multiset Γ is empty this shall be abbreviated to $Exp(\phi)$.

The one-step 'β-rules' arise for the core calculus by analysing the process of normalisation via the Curry-Howard correspondence (those for the conditional are intuitive). They are as follows.

$$
\begin{array}{rcl}
(\lambda x{:}\,\phi.M)N & \leadsto_{\beta} & M[x := N] \\
\mathsf{let}\ M \otimes N\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ P & \leadsto_{\beta} & P[x := M, y := N] \\
\mathsf{derelict}(\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N) & \leadsto_{\beta} & N[\vec{x} := \vec{M}] \\
\mathsf{discard}\ (\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N)\ \mathsf{in}\ P & \leadsto_{\beta} & \mathsf{discard}\ \vec{M}\ \mathsf{in}\ P \\
\mathsf{copy}\ (\mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N)\ \mathsf{as}\ y, z\ \mathsf{in}\ P & \leadsto_{\beta} & \mathsf{copy}\ \vec{M}\ \mathsf{as}\ \vec{x'}, \vec{x''}\ \mathsf{in} \\
 & & P\,[y := \mathsf{promote}\ \vec{x'}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N, \\
 & & \quad z := \mathsf{promote}\ \vec{x''}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N] \\
\mathsf{if}\ \mathtt{true}\ \mathsf{then}\ M\ \mathsf{else}\ N & \leadsto_{\beta} & M \\
\mathsf{if}\ \mathtt{false}\ \mathsf{then}\ M\ \mathsf{else}\ N & \leadsto_{\beta} & N
\end{array}
$$

Again further details can be found in earlier papers [3, 5]. Two important properties are the following, which are known as closure under substitution and the subject reduction property, respectively.

Proposition 1.

1. If $\Gamma \triangleright M{:}\,\phi$ and $\Delta, x{:}\,\phi \triangleright N{:}\,\psi$ then $\Gamma, \Delta \triangleright N[x := M]{:}\,\psi$.
2. If $\Gamma \triangleright M{:}\,\phi$ and $M \leadsto_{\beta} N$ then $\Gamma \triangleright N{:}\,\phi$.

To use linear PCF in the operational rôle mentioned in the introduction, we need to consider how a program (a closed term) is reduced, or evaluates, to a value. To do this we need to consider where and when to apply the β-rules from above. This can be done by defining a big-step operational semantics, *viz.* an inductively defined relation, written

$$x\colon\phi \triangleright x\colon\phi \qquad \emptyset \triangleright b\colon \mathsf{bool} \qquad \emptyset \triangleright \Omega^{\phi}\colon\phi$$

$$\frac{\Gamma, x\colon\phi \triangleright M\colon\psi}{\Gamma \triangleright \lambda x\colon\phi.M\colon\phi\multimap\psi}\,(\multimap_{\mathcal{I}}) \qquad \frac{\Gamma \triangleright M\colon\phi\multimap\psi \qquad \Delta \triangleright N\colon\phi}{\Gamma, \Delta \triangleright MN\colon\psi}\,(\multimap_{\mathcal{E}})$$

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

$$\frac{\Gamma \triangleright M\colon !\phi}{\Gamma \triangleright \mathsf{derelict}(M)\colon\phi}\ \textit{Dereliction}$$

$$\frac{\Gamma \triangleright M\colon !\phi \qquad \Delta \triangleright N\colon\psi}{\Gamma, \Delta \triangleright \mathsf{discard}\ M\ \mathsf{in}\ N\colon\psi}\ \textit{Weakening}$$

$$\frac{\Gamma \triangleright M\colon !\phi \qquad \Delta, x\colon !\phi, y\colon !\phi \triangleright N\colon\psi}{\Gamma, \Delta \triangleright \mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N\colon\psi}\ \textit{Contraction}$$

$$\frac{\Gamma \triangleright M\colon\phi \qquad \Delta \triangleright N\colon\psi}{\Gamma, \Delta \triangleright M \otimes N\colon\phi\otimes\psi}\,(\otimes_{\mathcal{I}})$$

$$\frac{\Gamma \triangleright M\colon\phi\otimes\psi \qquad \Delta, x\colon\phi, y\colon\psi \triangleright N\colon\varphi}{\Gamma, \Delta \triangleright \mathsf{let}\ M\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ N\colon\varphi}\,(\otimes_{\mathcal{E}})$$

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

Figure 1: Type Assignment for Linear PCF.

$$M \Downarrow v$$

where M and v are closed terms and v is a value (defined later).

Normally when defining this relation there are a number of choices corresponding to the different calling mechanisms. In his influential paper, Abramsky [1] demonstrated that the refined connectives of linear logic effectively suggest their own evaluation strategy. However, Abramsky's semantics was for a slightly different calculus (one which fails to have the properties given in Proposition 1), although the main difference is with the rules involving the exponential.

Firstly values are defined inductively as

$$\begin{array}{rcl} v & ::= & \texttt{true}, \texttt{false} \\ & | & \lambda x{:}\,\phi.M \\ & | & v \otimes v \\ & | & \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ M. \end{array}$$

The operational semantics is given in Figure 2.

Two remarks are worth making here. Firstly it is tempting to suggest that as attention is restricted to just closed terms, all occurrences of the *Promotion* rule yield terms of the form

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

and so one need only consider the syntactic form $\mathsf{promote}(M)$ (c.f. [17, p. 403]). Unfortunately, as there are constants, this simply is not true; for example the closed term

$$\mathsf{promote}\ (\mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ \texttt{true})\ \mathsf{for}\ x\ \mathsf{in}\ x.$$

Secondly, there are alternative definitions of the rules for *Weakening* and *Contraction*, *viz.*

$$\frac{M \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ P \qquad \mathsf{discard}\ \vec{v}\ \mathsf{in}\ N \Downarrow v'}{\mathsf{discard}\ M\ \mathsf{in}\ N \Downarrow v'}$$

$$\frac{M \Downarrow \mathsf{promote}\ \vec{v}\ \mathsf{for}\ \vec{z}\ \mathsf{in}\ P \qquad \mathsf{copy}\ \vec{v}\ \mathsf{as}\ \vec{z'}, \vec{z''}\ \mathsf{in}\ N \left[\begin{array}{l} x := \mathsf{promote}\ \vec{z'}\ \mathsf{for}\ \vec{z}\ \mathsf{in}\ P \\ y := \mathsf{promote}\ \vec{z''}\ \mathsf{for}\ \vec{z}\ \mathsf{in}\ P \end{array}\right] \Downarrow v'}{\mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N \Downarrow v'}$$

These are given directly by the β-rules and were included in a first draft of this paper. However given the definition of values above, they are equivalent (and much less efficient) to those in Figure 2. If we had constants of exponential type, it might be the case that these rules were preferable. Another reason for not using them here is practical. In the alternative rule for *Contraction* given above, the upper two substitutions involve *open* terms, which is quite unusual and would require a complicated implementation involving pointers (the rules of Figure 2 can be implemented quite simply, see §5).

An alternative method of presenting program evaluation is to define a transition relation, written

$$M \Rightarrow M'$$

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

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

$$\frac{M \Downarrow v \qquad N \Downarrow v'}{M \otimes N \Downarrow v \otimes v'}\ (\Downarrow \otimes_{\mathcal{I}})$$

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

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

$$\frac{M_i \Downarrow v_i \qquad 0 \leq i \leq k}{\mathsf{promote}\ M_i\ \mathsf{for}\ x_i\ \mathsf{in}\ N \Downarrow \mathsf{promote}\ v_i\ \mathsf{for}\ x_i\ \mathsf{in}\ N}\ (\Downarrow Promotion)$$

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

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

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

Figure 2: Operational Semantics for Linear PCF.

$$(\lambda x{:}\,\phi.M)v \Rightarrow M[x := v]$$

$$\text{let } v \otimes v' \text{ be } x \otimes y \text{ in } P \Rightarrow P[x := v, y := v']$$

$$\text{derelict}(\text{promote } \vec{v} \text{ for } \vec{x} \text{ in } N) \Rightarrow N[\vec{x} := \vec{v}]$$

$$\text{discard } (\text{promote } \vec{v} \text{ for } \vec{x} \text{ in } N) \text{ in } P \Rightarrow \text{discard } \vec{v} \text{ in } P$$

$$\text{copy } (\text{promote } \vec{v} \text{ for } \vec{x} \text{ in } N) \text{ as } y, z \text{ in } P \Rightarrow \text{copy } \vec{v} \text{ as } \vec{x'}, \vec{x''} \text{ in } P \begin{bmatrix} y := \text{promote } \vec{x'} \text{ for } \vec{x} \text{ in } N \\ z := \text{promote } \vec{x''} \text{ for } \vec{x} \text{ in } N \end{bmatrix}$$

$$\text{if } \texttt{true} \text{ then } M \text{ else } N \Rightarrow M$$

$$\text{if } \texttt{false} \text{ then } M \text{ else } N \Rightarrow N$$

$$\frac{M \Rightarrow M'}{MN \Rightarrow M'N} \qquad \frac{N \Rightarrow N'}{vN \Rightarrow vN'}$$

$$\frac{M \Rightarrow M'}{\text{if } M \text{ then } N \text{ else } P \Rightarrow \text{if } M' \text{ then } N \text{ else } P}$$

$$\frac{M \Rightarrow M'}{M \otimes N \Rightarrow M' \otimes N} \qquad \frac{N \Rightarrow N'}{v \otimes N \Rightarrow v \otimes N'}$$

$$\frac{M \Rightarrow M'}{\text{let } M \text{ be } x \otimes y \text{ in } N \Rightarrow \text{let } M' \text{ be } x \otimes y \text{ in } N}$$

$$\frac{M_i \Rightarrow M_i'}{\begin{array}{l}\text{promote } M_1, \ldots, M_i, \ldots, M_k \text{ for } x_1, \ldots, x_i, \ldots, x_k \text{ in } N \\ \Rightarrow \text{promote } M_1, \ldots, M_i', \ldots, M_k \text{ for } x_1, \ldots, x_i, \ldots, x_k \text{ in } N\end{array}}$$

$$\frac{M \Rightarrow M'}{\text{derelict}(M) \Rightarrow \text{derelict}(M')}$$

$$\frac{M \Rightarrow M'}{\text{discard } M \text{ in } N \Rightarrow \text{discard } M' \text{ in } N}$$

$$\frac{M \Rightarrow M'}{\text{copy } M \text{ as } x, y \text{ in } N \Rightarrow \text{copy } M' \text{ as } x, y \text{ in } N}$$

Figure 3: Transition Semantics for Linear PCF.

where M and M' are closed terms. In contrast to the operational semantics given earlier, this is a single-step semantics. The rules are given in Figure 3.[2]

The two systems are equivalent in the following sense.

Theorem 1. $M \Downarrow v$ iff $M \Rightarrow^* v$.

Proof. From left to right by induction on $M \Downarrow v$. From right to left by proving that if $M \Rightarrow M'$ and $M' \Downarrow v$ then $M \Downarrow v$ (by induction on $M \Rightarrow M'$). ∎

I shall use the big-step semantics of Figure 2 in the rest of this paper as it is more abstract. Next we need to introduce the notion of a *context.* This is to be thought of as a linear PCF-term with a typed 'hole' in it; in to which another term may be substituted. Unlike ordinary substitution, this process is permitted to capture variables.

Unlike the case for PCF, formally defining this notion of a linear context is surprisingly difficult. More details are given in Appendix A. For now it is sufficient just to understand the notation:

$$\bullet(\Gamma)\colon\phi; \Delta \triangleright \mathcal{L}[\bullet^{\phi}]\colon\psi$$

denotes a linear context, $\mathcal{L}$, of type ψ, with free variables Δ, and with a single hole of type ϕ and free variable set given by Γ. An example may be useful. Consider

$$\bullet(x\colon\phi)\colon\phi\otimes\mathsf{bool}; \emptyset \triangleright \lambda x\colon\phi.\bullet\colon\phi\multimap(\phi\otimes\mathsf{bool}),$$

which is a linear context with a single hole. The rule governing placement of a context for a hole is

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon\phi \qquad \mathcal{H}', \bullet(\Gamma)\colon\phi; \Delta \triangleright N\colon\psi}{\mathcal{H}, \mathcal{H}'; \Delta \triangleright N[M/\bullet]\colon\psi}\ \textit{Placement.}$$

Thus for the example above we could place the term

$$x\colon\phi \triangleright x\otimes\mathtt{true}\colon\phi\otimes\mathsf{bool}$$

in the hole to yield the term

$$\emptyset \triangleright \lambda x\colon\phi.x\otimes\mathtt{true}\colon\phi\multimap(\phi\otimes\mathsf{bool})$$

where one should notice that the free variable x has become bound by the process of placement.

It is now possible to define a notion of observation. As is the case for PCF we are able to give two definitions which differ on what can be observed: either elements of the ground type(s) (in this case just booleans) or termination at all types. For call-by-value PCF these notions coincide, but there is some doubt as to whether they do for linear PCF despite the fact it is a call-by-value language. This point will be considered in more detail at the end of this section.

Definition 1. Given $\Gamma \triangleright M\colon\phi$ and $\Gamma \triangleright N\colon\phi$ we shall say that M *ground-observationally refines* N, written $\Gamma \triangleright M \sqsubseteq^{gnd}_{\phi} N$, where for all closing boolean contexts, $\bullet(\Gamma)\colon\phi; \emptyset \triangleright \mathcal{L}[\bullet^{\phi}]\colon\mathsf{bool}$, if $\mathcal{L}[M] \Downarrow \mathtt{true}$ then $\mathcal{L}[N] \Downarrow \mathtt{true}$.

[2]One could present these rules more succinctly by defining a notion of evaluation context.

We would then write $\Gamma \rhd M \approx^{gnd}_{\phi} N$ iff $\Gamma \rhd M \sqsubseteq^{gnd}_{\phi} N$ and $\Gamma \rhd N \sqsubseteq^{gnd}_{\phi} M$ and refer to this relation as *ground observational equivalence.*

Definition 2. Given $\Gamma \rhd M : \phi$ and $\Gamma \rhd N : \phi$ we shall say that M *lazy-observationally refines* N, written $\Gamma \rhd M \sqsubseteq^{lazy}_{\phi} N$, where for all closing contexts, $\bullet(\Gamma) : \phi; \emptyset \rhd \mathcal{L}[\bullet^{\phi}] : \psi$, if $\mathcal{L}[M] \Downarrow$ then $\mathcal{L}[N] \Downarrow$.

We write $\Gamma \rhd M \approx^{lazy}_{\phi} N$ iff $\Gamma \rhd M \sqsubseteq^{lazy}_{\phi} N$ and $\Gamma \rhd N \sqsubseteq^{lazy}_{\phi} M$ and refer to this relation as *lazy observational equivalence.* It is clear that lazy observational equivalence is a stronger notion than ground observational equivalence.

Lemma 1. If $\Gamma \rhd M \approx^{lazy}_{\phi} N$ then $\Gamma \rhd M \approx^{gnd}_{\phi} N$.

Lemma 2.

1. If $\Gamma \rhd M \sqsubseteq^{gnd}_{\phi} N$ then $\Delta \rhd \mathcal{L}[M] \sqsubseteq^{gnd}_{\psi} \mathcal{L}[N]$ for all contexts $\bullet(\Gamma) : \phi; \Delta \rhd \mathcal{L}[\bullet^{\phi}] : \psi$.
2. If $\Gamma \rhd M \sqsubseteq^{lazy}_{\phi} N$ then $\Delta \rhd \mathcal{L}[M] \sqsubseteq^{lazy}_{\psi} \mathcal{L}[N]$ for all contexts $\bullet(\Gamma) : \phi; \Delta \rhd \mathcal{L}[\bullet^{\phi}] : \psi$.

Proof. Effectively by composition of contexts. ∎

A linear pre-congruence relation is essentially a relation on linear PCF-terms which respects the rules of term formation.

Definition 3.

1. If $\mathcal{R}$ is a family of relations $\mathcal{R}_{\Gamma,\phi} \subseteq Exp_{\Gamma}(\phi) \times Exp_{\Gamma}(\phi)$ which satisfies the rules in Figure 4, then it is said to be a *pre-congruence.*
2. If $\mathcal{R}$ is a pre-congruence and, in addition, satisfies the rule

$$\frac{\Gamma \rhd M \,\mathcal{R}\, N : \phi}{\Gamma \rhd N \,\mathcal{R}\, M : \phi}\ \mathit{Symmetry}$$

   then it is said to be a *congruence.*

Lemma 3. Let $\mathcal{R}$ be a pre-congruence relation. Suppose that $\Gamma \rhd M \,\mathcal{R}\, N : \phi$. Then for a linear context $\bullet(\Gamma) : \phi; \Delta \rhd \mathcal{L}[\bullet] : \psi$ it is the case that $\Delta \rhd \mathcal{L}[M] \,\mathcal{R}\, \mathcal{L}[N] : \psi$.

Proof. By induction on the derivation of $\mathcal{L}[\bullet]$. (The rules for formation are given in Appendix A.) ∎

Lemma 4. $\sqsubseteq$ is a pre-congruence.

The problem with both definitions of observational equivalence is that the quantification over all contexts makes them very difficult to work with. Instead a co-inductive definition of program equivalence called applicative bisimulation can be given, which can then be compared to the notions of observational equivalence.

The relation of applicative simularity is defined as the greatest fixed point of a certain monotone operation on relations. This operation is given in two stages.

$$\frac{\Gamma \triangleright M : \phi}{\Gamma \triangleright M \mathcal{R} M : \phi}\ \textit{Reflexivity}$$

$$\frac{\Gamma \triangleright M \mathcal{R} N : \phi \qquad \Gamma \triangleright N \mathcal{R} P : \phi}{\Gamma \triangleright M \mathcal{R} P : \phi}\ \textit{Transitivity}$$

$$\frac{\Gamma \triangleright M : \phi \qquad \Delta, x : \phi \triangleright N \mathcal{R} P : \psi}{\Gamma, \Delta \triangleright N[x := M] \mathcal{R} P[x := M] : \psi}\ \textit{Subs}_1$$

$$\frac{\Gamma \triangleright M \mathcal{R} N : \phi \qquad \Delta, x : \phi \triangleright P : \psi}{\Gamma, \Delta \triangleright P[x := M] \mathcal{R} P[x := N] : \psi}\ \textit{Subs}_2$$

$$\frac{\Gamma, x : \phi \triangleright M \mathcal{R} N : \psi}{\Gamma \triangleright \lambda x : \phi . M \mathcal{R} \lambda x : \phi . N : \phi \multimap \psi}\ (\multimap_{\mathcal{I}})$$

$$\frac{\Gamma_1 \triangleright M_1 \mathcal{R} M_1' : !\phi_1 \cdots \Gamma_n \triangleright M_n \mathcal{R} M_n' : !\phi_n \quad x_1 : !\phi_1, \ldots, x_n : !\phi_n \triangleright N \mathcal{R} N' : \psi}{\Gamma_1, \ldots, \Gamma_n \triangleright \mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N \mathcal{R} \mathsf{promote}\ \vec{M'}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N' : !\psi}\ \textit{Promotion}$$

$$\frac{\Gamma \triangleright M \mathcal{R} M' : !\phi \qquad \Delta, x : !\phi, y : !\phi \triangleright N \mathcal{R} N' : \psi}{\Gamma, \Delta \triangleright \mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N \mathcal{R} \mathsf{copy}\ M'\ \mathsf{as}\ x, y\ \mathsf{in}\ N' : \psi}\ \textit{Contraction}$$

$$\frac{\Gamma \triangleright M \mathcal{R} M' : \phi \otimes \psi \qquad \Delta, x : \phi, y : \psi \triangleright N \mathcal{R} N' : \varphi}{\Gamma, \Delta \triangleright \mathsf{let}\ M\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ N \mathcal{R} \mathsf{let}\ M\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ N : \varphi}\ (\otimes\varepsilon)$$

Figure 4: Rules for a Linear Pre-Congruence Relation.

Definition 4. Given a family of (type-indexed) relations $R = (R_\phi \subseteq Exp(\phi) \times Exp(\phi))$ between closed linear PCF-terms, we 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 $v_1 \langle R \rangle_\phi v_1'$ and $v_2 \langle R \rangle_\psi v_2'$,
- $\lambda x\colon \phi.M \langle R \rangle_{\phi \multimap \psi} \lambda x\colon \phi.M'$ if $\forall v\colon \phi.M[x := v] \, R_\psi \, M'[x := v]$, 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 can be extended to closed linear PCF-terms as follows.

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

Lemma 5. The function $R \mapsto [R]$ is monotone.

Proof. Essentially by induction over $v \langle R \rangle_\phi v'$. ∎

A family of relations, R, satisfying $R \subseteq [R]$, is called a (linear PCF) simulation. As the function described in Lemma 5 is monotone and the families indexed by their types form a complete lattice then it has a *greatest* fixed point, which is denoted by $\leq$, and referred to as *(linear PCF) applicative simularity*. Associated with this greatest fixed point is a co-inductive principle:

To show $(M, N) \in \leq$ it is sufficient to find an $\mathcal{S}$ such that $\mathcal{S} \subseteq [\mathcal{S}]$ and $(M, N) \in \mathcal{S}$.

Applicative simularity can be extended to open linear PCF-terms as follows.

$$x_1\colon \phi_1, \ldots, x_n\colon \phi_n \triangleright M \leq^\circ_\psi N \iff \forall v_i.\, \emptyset \triangleright M[\vec{x} := \vec{v}] \leq_\psi N[\vec{x} := \vec{v}] \text{ where } \emptyset \triangleright v_i\colon \phi_i,$$

where the v_i are values. It is easy to show that the relation $\leq$ is a partial order. Applicative bisimularity, written $\approx^{app}$ is defined as the symmetrisation of $\leq$, *viz.*

$$\Gamma \triangleright M \approx^{app}_\phi N \text{ iff } \Gamma \triangleright M \leq^\circ_\phi N \text{ and } \Gamma \triangleright N \leq^\circ_\phi M.$$

An important property is that $\leq$ is a pre-congruence. It turns out, as is the case for PCF, that this is rather difficult to prove. A well-established and ingenious technique was given by Howe [14]. One defines another relation, which is trivially a pre-congruence, and, rather less trivially, is an applicative simulation. I shall adapt his technique here for linear PCF.

Definition 5. The relation $\leq^\star$ between two well-typed expressions is defined inductively

as follows.

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

An important property of this relation is the following

Lemma 6. If $\Gamma \triangleright M \leq^{\star}_{\phi} N$ and $\Gamma \triangleright N \leq^{\circ}_{\phi} P$ then $\Gamma \triangleright M \leq^{\star}_{\phi} P$.

Proof. By induction over $\Gamma \triangleright M \leq^{\star}_{\phi} N$. Here I give three example cases.

1. $\Gamma \triangleright \mathsf{derelict}(M) \leq^{\star}_{\phi} N$. Thus $\exists M'.\Gamma \triangleright M \leq^{\star}_{!\phi} M'$ and $\Gamma \triangleright \mathsf{derelict}(M') \leq^{\circ}_{\phi} N$. By assumption and transitivity of $\leq^{\circ}$ then $\Gamma \triangleright \mathsf{derelict}(M') \leq^{\circ}_{\phi} P$ and we are done.

2. $\Gamma \triangleright \mathsf{promote}\ M\ \mathsf{for}\ x\ \mathsf{in}\ Q \leq^{\star}_{!\phi} N$. Thus $\exists M', Q'.\Gamma \triangleright M \leq^{\star}_{!\psi} M'$ and $x{:}!\psi \triangleright Q \leq^{\star}_{\phi} Q'$ and $\Gamma \triangleright \mathsf{promote}\ M'\ \mathsf{for}\ x'\ \mathsf{in}\ Q' \leq^{\circ}_{!\phi} N$. By assumption and transitivity of $\leq^{\circ}$ then $\Gamma \triangleright \mathsf{promote}\ M'\ \mathsf{for}\ x'\ \mathsf{in}\ Q' \leq^{\circ}_{!\phi} P$ and we are done.

3. $\Gamma, \Delta \triangleright \mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ Q \leq^{\star}_{\phi} N$. Thus $\exists M', Q'.\Gamma \triangleright M \leq^{\star}_{!\psi} M'$ and $\Delta, x{:}!\psi, y{:}!\psi \triangleright Q \leq^{\star}_{\phi} Q'$ and $\Gamma, \Delta \triangleright \mathsf{copy}\ M'\ \mathsf{as}\ x, y\ \mathsf{in}\ Q' \leq^{\circ}_{\phi} N$. By assumption and transitivity of $\leq^{\circ}$ then $\Gamma, \Delta \triangleright \mathsf{copy}\ M'\ \mathsf{as}\ x, y\ \mathsf{in}\ Q' \leq^{\circ}_{\phi} P$ and we are done.

■

One direction of the equivalence is now immediate.

Proposition 2. If $\Gamma \triangleright M \leq^{\circ}_{\phi} N$ then $\Gamma \triangleright M \leq^{\star}_{\phi} N$.

Proof. It is clear that $\Gamma \triangleright M \leq^{\star}_{\phi} M$ (reflexivity) and, by assumption, that $\Gamma \triangleright M \leq^{\circ}_{\phi} N$. From Lemma 6 we conclude $\Gamma \triangleright M \leq^{\star}_{\phi} N$. ■

It is also relatively straightforward to prove the following.

Lemma 7. If $\Delta \triangleright M \leq^{\star}_{\phi} M'$ and $\Gamma, x{:}\,\phi \triangleright P \leq^{\star}_{\psi} P'$ then $\Gamma, \Delta \triangleright P[x := M] \leq^{\star}_{\psi} P'[x := M']$.

Proof. By induction on the structure of P. ■

An important property is the following.

Lemma 8. If $\emptyset \triangleright M \leq^{\star}_{\phi} N$ and $M \Downarrow v$ then $\exists v'$ such that $N \Downarrow v'$ and $\emptyset \triangleright v \leq^{\star}_{\phi} v'$.

Proof. By induction on structure of $M \Downarrow v$. Four 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 $\exists M', Q'.\emptyset \triangleright M \leq^{\star}_{!\psi} M'$ and $x{:}\,!\psi \triangleright Q \leq^{\star}_{\phi} Q'$ and $\emptyset \triangleright \mathsf{promote}\ M'\ \mathsf{for}\ x\ \mathsf{in}\ Q' \leq_{!\phi} N$. By induction we have $M' \Downarrow v'$ and $\emptyset \triangleright v \leq^{\star}_{\phi} v'$. 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 $\emptyset \triangleright \mathsf{promote}\ v'\ \mathsf{for}\ x\ \mathsf{in}\ Q' \leq_{!\phi} w$. From Proposition 2 we can conclude $\emptyset \triangleright \mathsf{promote}\ v\ \mathsf{for}\ x\ \mathsf{in}\ Q \leq^{\star}_{!\phi} w$ and we are done.

2. $\mathsf{derelict}(M) \Downarrow v$: By assumption $\exists M'.\emptyset \triangleright M \leq^{\star}_{!\phi} M'$ and $\emptyset \triangleright \mathsf{derelict}(M') \leq_{\phi} N$. By induction we have $M' \Downarrow v''$ and $\emptyset \triangleright \mathsf{promote}\ v'\ \mathsf{for}\ x\ \mathsf{in}\ P \leq^{\star}_{!\phi} v''$. By definition $\exists w, P'.\emptyset \triangleright v' \leq^{\star}_{!\psi} w$ and $x{:}\,!\psi \triangleright P \leq^{\star}_{\phi} P'$ and $\emptyset \triangleright \mathsf{promote}\ w\ \mathsf{for}\ x\ \mathsf{in}\ P' \leq_{!\phi} v''$. By Lemma 7 we have that $\emptyset \triangleright P[x := v'] \leq^{\star}_{\phi} P'[x := w]$. By determinancy of evaluation we have that $v'' \equiv \mathsf{promote}\ w''\ \mathsf{for}\ y\ \mathsf{in}\ Q$ and then as $\emptyset \triangleright \mathsf{promote}\ v'\ \mathsf{for}\ x\ \mathsf{in}\ P' \leq_{!\phi} \mathsf{promote}\ w''\ \mathsf{for}\ y\ \mathsf{in}\ Q$ we can conclude that $\emptyset \triangleright w \leq_{!\psi} w''$ and $\emptyset \triangleright P'[x := w] \leq_{\phi} Q[y := w'']$. From Lemma 6 we have that $\emptyset \triangleright P[x := v'] \leq^{\star}_{\phi} Q[y := w']$ and then by induction $Q[y := w'] \Downarrow a$ and $\emptyset \triangleright v \leq^{\star}_{\phi} a$. We can now conclude that $\mathsf{derelict}(M') \Downarrow a$ and hence that $N \Downarrow c$ and $\emptyset \triangleright a \leq_{\phi} c$. From $\emptyset \triangleright v \leq^{\star}_{\phi} a$ and $\emptyset \triangleright a \leq_{\phi} c$ we can conclude that $\emptyset \triangleright v \leq^{\star}_{\phi} c$ and we are done.

3. $\mathsf{discard}\ M\ \mathsf{in}\ P \Downarrow v$: By assumption $\exists M', P'.\emptyset \triangleright M \leq^{\star}_{!\phi} M'$ and $\emptyset \triangleright P \leq^{\star}_{\psi} P'$ and $\emptyset \triangleright \mathsf{discard}\ M'\ \mathsf{in}\ P' \leq_{\psi} N$. By induction we have both $M' \Downarrow w$ and $\emptyset \triangleright v' \leq^{\star}_{!\phi} w$; and $P' \Downarrow w'$ and $\emptyset \triangleright v \leq^{\star}_{\psi} w'$. We can deduce that $\mathsf{discard}\ M'\ \mathsf{in}\ P' \Downarrow w'$ and hence $N \Downarrow a$ and $\emptyset \triangleright w' \leq_{\psi} a$. By Lemma 6 we can conclude $\emptyset \triangleright v \leq^{\star}_{\psi} a$ and we are done.

4. $\mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ P \Downarrow v$: By assumption $\exists M', P'.\emptyset \triangleright M \leq^{\star}_{!\phi} M'$ and $x{:}\,!\phi, y{:}\,!\phi \triangleright P \leq^{\star}_{\psi} P'$ and $\emptyset \triangleright \mathsf{copy}\ M'\ \mathsf{as}\ x, y\ \mathsf{in}\ P' \leq_{\psi} N$. By induction we have that $M' \Downarrow v''$ and $\emptyset \triangleright v' \leq^{\star}_{!\phi} v''$. By Lemma 7 we have that $\emptyset \triangleright P[x, y := v'] \leq^{\star}_{\psi} P'[x, y := v'']$. By induction we have that $P'[x, y := v''] \Downarrow v'''$ and $\emptyset \triangleright v \leq^{\star}_{\psi} v'''$. We can deduce that $\mathsf{copy}\ M'\ \mathsf{as}\ x, y\ \mathsf{in}\ P' \Downarrow v'''$ and by assumption then $N \Downarrow w$ and $\emptyset \triangleright v''' \leq_{\psi} w$. From $\emptyset \triangleright v \leq^{\star}_{\psi} v'''$ and $\emptyset \triangleright v''' \leq_{\psi} w$, we can conclude that $\emptyset \triangleright v \leq^{\star}_{\psi} w$ and we are done.

■

Lemma 9. If $\emptyset \triangleright v \leq^{\star}_{\phi} v'$ then $v(\leq^{\star})v'$.

Proof. By induction on the structure of v. For example, assume that $\emptyset \triangleright$ promote $\vec{v}$ for $\vec{x}$ in $M \leq^{\star}_{!\phi}$ promote $\vec{v'}$ for $\vec{x'}$ in M' *viz.* $\exists \vec{w}, N$ such that $\emptyset \triangleright v_i \leq^{\star}_{!\psi_1} w_i, \vec{x} \triangleright M \leq^{\star}_{\phi} N$ and $\emptyset \triangleright$ promote $\vec{w}$ for $\vec{x}$ in $N \leq_{!\phi}$ promote $\vec{v'}$ for $\vec{x'}$ in M'. By definition this implies $\emptyset \triangleright N[\vec{x} := \vec{w}] \leq_{\phi} M'[\vec{x'} := \vec{v'}]$. By Lemma 7 we have also that $\emptyset \triangleright M[\vec{x} := \vec{v}] \leq^{\star}_{\phi} N[\vec{x} := \vec{w}]$. By Lemma 6 we can conclude that $\emptyset \triangleright M[\vec{x} := \vec{v}] \leq^{\star}_{\phi} M'[\vec{x'} := \vec{v'}]$ and we are done. ■

We can now prove the other direction of the equivalence.

Proposition 3. If $\Gamma \triangleright M \leq^{\star}_{\phi} N$ then $\Gamma \triangleright M \leq^{\circ}_{\phi} N$.

Proof. Form the set

$$\mathcal{S} \stackrel{\text{def}}{=} \{(M, N) \mid \emptyset \triangleright M \leq^{\star}_{\phi} N\} \subseteq Exp(\phi) \times Exp(\phi)$$

and show that $\mathcal{S} \subseteq [\mathcal{S}]$ which holds given Lemmas 8 and 9. Thus we have that $\emptyset \triangleright M \leq^{\star}_{\phi} N$ implies $\emptyset \triangleright M \leq_{\phi} N$. We have that $\forall v. \emptyset \triangleright v \leq^{\star}_{\psi} v$, hence given any open terms $\vec{x}$: $\Gamma \triangleright M \leq^{\star}_{\phi} N$ we have by Lemma 7 that $\emptyset \triangleright M[\vec{x} := \vec{v}] \leq^{\star}_{\phi} N[\vec{x} := \vec{v}]$ and then we can invoke the above reasoning for the resulting closed terms. ■

Proposition 4. $\leq^{\star}$ is a pre-congruence.

Proof. Simply by checking that $\leq^{\star}$ satisfies the rules given in Figure 4. The $Subs_1$ and $Subs_2$ rules follow from Lemma 7. The other rules hold trivially by definition and given that $\leq$ is reflexive. ■

Theorem 2. $\leq^{\circ}$ is a pre-congruence.

Proof. Immediate from Propositions 2, 3 and 4. ■

It is now possible to consider to what extent the two notions of program equivalence, observational equivalence and applicative bisimularity, are equivalent.

Proposition 5. If $\Gamma \triangleright M \leq^{\circ}_{\phi} N$ then $\Gamma \triangleright M \sqsubseteq^{lazy}_{\phi} N$.

Proof. Suppose that $\Gamma \triangleright M \leq^{\circ}_{\phi} N$. As $\leq^{\circ}$ is a pre-congruence we have that for any closing linear context $\bullet(\Gamma)$: $\phi; \emptyset \triangleright \mathcal{L}[\bullet^{\phi}]$: ψ that

$$\emptyset \triangleright \mathcal{L}[M] \leq_{\psi} \mathcal{L}[N]$$

which by definition gives that if $\mathcal{L}[M] \Downarrow v$ then $\exists v'$ such that $\mathcal{L}[N] \Downarrow v'$ and we are done. ■

Corollary 1. If $\Gamma \triangleright M \leq^{\circ}_{\phi} N$ then $\Gamma \triangleright M \sqsubseteq^{gnd}_{\phi} N$.

Proposition 6. If $\emptyset \triangleright M \sqsubseteq^{lazy}_{\phi} N$ then $\emptyset \triangleright M \leq_{\phi} N$.

Proof. We form the set

$$\mathcal{S} = \{(M, N) \mid \emptyset \triangleright M \sqsubseteq_{\phi}^{lazy} N\}$$

and show that $\mathcal{S} \subseteq [\mathcal{S}]$. Thus take M and N such that $\emptyset \triangleright M \sqsubseteq_{\phi}^{lazy} N$ and $M \Downarrow v$. As they are lazy-observationally equivalent, taking the identity context gives that there exists a v' such that $N \Downarrow v'$. We show that $v \, \langle \mathcal{S} \rangle_{\phi} \, v'$ by induction on the type ϕ.

1. ($\phi \equiv \mathsf{bool}$) Build the context

$$\mathcal{L}[\bullet^{\mathsf{bool}}] \stackrel{\mathsf{def}}{=} \mathsf{if} \ \bullet \ \mathsf{then} \ \mathtt{true} \ \mathsf{else} \ \Omega.$$

   Thus $\mathcal{L}[M] \Downarrow \mathtt{true}$ iff $M \Downarrow \mathtt{true}$ iff $\mathcal{L}[N] \Downarrow \mathtt{true}$ iff $N \Downarrow \mathtt{true}$ and we are done. The case for $M \Downarrow \mathtt{false}$ is similar.

2. ($\phi \equiv \phi \otimes \psi$) This holds by induction.

3. ($\phi \equiv \phi \multimap \psi$) We have that $M \Downarrow \lambda x.P$ and $N \Downarrow \lambda x.Q$. Take any context $\mathcal{L}[\bullet^{\psi}]$ and call $\mathcal{L}'$ the context which results from replacing the occurrence of the hole $\bullet^{\psi}$ with the term $\bullet^{\phi \multimap \psi} v$. Thus

$$\begin{aligned}
\mathcal{L}[P[x := v]] \Downarrow &\iff \mathcal{L}[(\lambda x.P)v] \Downarrow \\
&\iff \mathcal{L}'[\lambda x.P] \Downarrow \\
&\iff \mathcal{L}'[M] \Downarrow \\
&\iff \mathcal{L}'[N] \Downarrow \\
&\iff \mathcal{L}'[\lambda x.Q] \Downarrow \\
&\iff \mathcal{L}[(\lambda x.Q)v] \Downarrow \\
&\iff \mathcal{L}[Q[x := v]] \Downarrow .
\end{aligned}$$

4. ($\phi \equiv !\phi$) Thus we have that $M \Downarrow \mathsf{promote} \ \vec{v} \ \mathsf{for} \ \vec{x} \ \mathsf{in} \ P$ and $N \Downarrow \mathsf{promote} \ \vec{v'} \ \mathsf{for} \ \vec{x'} \ \mathsf{in} \ Q$. Take any context $\mathcal{L}[\bullet^{\phi}]$ and call $\mathcal{L}'$ the context which results from replacing the hole $\bullet^{\phi}$ with the term $\mathsf{derelict}(\bullet^{!\phi})$. Thus

$$\begin{aligned}
\mathcal{L}[P[\vec{x} := \vec{v}]] \Downarrow &\iff \mathcal{L}[\mathsf{derelict}(\mathsf{promote} \ \vec{v} \ \mathsf{for} \ \vec{x} \ \mathsf{in} \ P)] \Downarrow \\
&\iff \mathcal{L}'[\mathsf{promote} \ \vec{v} \ \mathsf{for} \ \vec{x} \ \mathsf{in} \ P] \Downarrow \\
&\iff \mathcal{L}'[M] \Downarrow \\
&\iff \mathcal{L}'[N] \Downarrow \\
&\iff \mathcal{L}'[\mathsf{promote} \ \vec{v'} \ \mathsf{for} \ \vec{x'} \ \mathsf{in} \ Q] \Downarrow \\
&\iff \mathcal{L}[\mathsf{derelict}(\mathsf{promote} \ \vec{v'} \ \mathsf{for} \ \vec{x'} \ \mathsf{in} \ Q)] \Downarrow \\
&\iff \mathcal{L}[Q[\vec{x'} := \vec{v'}]] \Downarrow .
\end{aligned}$$

■

Lemma 10. If $\Gamma, x\colon\phi \triangleright M \sqsubseteq^{lazy}_{\psi} N$ then $\Gamma \triangleright M[x := v] \sqsubseteq^{lazy}_{\psi} N[x := v]$ for any $\emptyset \triangleright v\colon\phi$.

Proof. Assume that $\Gamma, x\colon\phi \triangleright M \sqsubseteq^{lazy}_{\psi} N$. Then for a given context $\bullet(\Gamma)\colon\psi; \emptyset \triangleright \mathcal{L}[\bullet^{\psi}]\colon\varphi$, we call $\mathcal{L}'$ the context which results from replacing the hole $\bullet^{\psi}$ which the term $(\lambda x\colon\phi.\bullet^{\psi})v$ (hence $\bullet(\Gamma, x\colon\phi); \emptyset \triangleright \mathcal{L}'[\bullet^{\psi}]\colon\varphi$).

$$\begin{aligned}
\mathcal{L}[M[x := v]] \Downarrow &\iff \mathcal{L}[(\lambda x.M)v] \Downarrow \\
&\iff \mathcal{L}'[M] \Downarrow \\
&\iff \mathcal{L}'[N] \Downarrow \\
&\iff \mathcal{L}[(\lambda x.N)v] \Downarrow \\
&\iff \mathcal{L}[N[x := v]] \Downarrow
\end{aligned}$$

■

Proposition 7. If $\Gamma \triangleright M \sqsubseteq^{lazy}_{\phi} N$ then $\Gamma \triangleright M \leq^{\circ}_{\phi} N$.

Proof. By definition $\vec{x}\colon\Gamma \triangleright M \leq^{\circ}_{\phi} N$ iff $\emptyset \triangleright M[\vec{x} := \vec{v}] \leq_{\phi} N[\vec{x} := \vec{v}]$ for values $\vec{v}$. From Lemma 10 we have that $\emptyset \triangleright M[\vec{x} := \vec{v}] \sqsubseteq^{lazy}_{\phi} N[\vec{x} := \vec{v}]$ and then we can apply Proposition 6 to get $\emptyset \triangleright M[\vec{x} := \vec{v}] \leq_{\phi} N[\vec{x} := \vec{v}]$ and we are done. ■

Thus two notions of program equivalence coincide; this is often called *operational extensionality.*

Corollary 2. $\Gamma \triangleright M \approx^{lazy}_{\phi} N$ iff $\Gamma \triangleright M \approx^{app}_{\phi} N$.

Discussion. However it does not appear that *ground* observational equivalence coincides with applicative bisimularity (unlike call-by-value PCF). This seems to be a problem with linearity: there are very few linear contexts of type bool. For example there appears to be no *boolean* context which distinguishes[3]

$$\lambda x\colon !\mathsf{bool}.\mathsf{discard}\ x\ \mathsf{in}\ \Omega^{\mathsf{bool}} \qquad \text{from} \qquad \Omega^{!\mathsf{bool}\multimap\mathsf{bool}},$$

thus $\emptyset \triangleright \lambda x\colon !\mathsf{bool}.\mathsf{discard}\ x\ \mathsf{in}\ \Omega \sqsubseteq^{gnd}_{!\mathsf{bool}\multimap\mathsf{bool}} \Omega$. However given Definition 4 they are clearly *not* applicatively similar. A co-inductive definition of applicative bisimularity which coincides with ground observational equivalence must wait for a future paper.

3 PCF

3.1 Syntax

We shall consider a PCF which is simply the λ-calculus extended with pairs, booleans, a conditional and non-terminating constants. More precisely, types are given by the grammar

$$\sigma ::= \mathsf{bool} \mid \sigma \supset \sigma \mid \sigma \wedge \sigma,$$

[3]Any context either entirely discards the term (in which case they are observably the same), or uses the term, *viz.* 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 type system.

and raw terms by the grammar

$$
\begin{array}{rcll}
e & ::= & \texttt{true}, \texttt{false} & \text{Booleans} \\
& | & x & \text{Variable} \\
& | & \lambda x{:}\sigma.e & \text{Abstraction} \\
& | & ee & \text{Application} \\
& | & \langle e, e \rangle & \text{Pair} \\
& | & \mathsf{fst}(e) & \text{First Projection} \\
& | & \mathsf{snd}(e) & \text{Second Projection} \\
& | & \text{if } e \text{ then } e \text{ else } e & \text{Conditional} \\
& | & \Omega^{\sigma} & \text{Non-termination;}
\end{array}
$$

where x is taken from some countable set of variables and σ is a well-formed type. A typing judgement is written $\Gamma \rhd e{:}\sigma$ where Γ is a set of (variable,type)-pairs. Again we shall consider only well-typed terms. The rules for forming typing judgements are given in Figure 5.

$$\Gamma, x{:}\phi \rhd x{:}\phi \qquad \Gamma \rhd b{:}\mathsf{bool} \qquad \Gamma \rhd \Omega^{\sigma}{:}\sigma$$

$$\frac{\Gamma \rhd e{:}\sigma \qquad \Gamma \rhd f{:}\tau}{\Gamma \rhd \langle e, f \rangle{:}\sigma \wedge \tau}\,(\wedge_{\mathcal{I}}) \qquad \frac{\Gamma \rhd e{:}\sigma \wedge \tau}{\Gamma \rhd \mathsf{fst}(e){:}\sigma}\,(\wedge_{\mathcal{E}-1}) \qquad \frac{\Gamma \rhd e{:}\sigma \wedge \tau}{\Gamma \rhd \mathsf{snd}(e){:}\tau}\,(\wedge_{\mathcal{E}-2})$$

$$\frac{\Gamma, x{:}\sigma \rhd e{:}\tau}{\Gamma \rhd \lambda x{:}\sigma.e{:}\sigma \supset \tau}\,(\supset_{\mathcal{I}}) \qquad \frac{\Gamma \rhd e{:}\sigma \supset \tau \qquad \Gamma \rhd f{:}\sigma}{\Gamma \rhd ef{:}\tau}\,(\supset_{\mathcal{E}})$$

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

Figure 5: Type Assignment for PCF.

There is a simple translation on formulae from **ILL** to intuitionistic logic (**IL**), which replaces the linear connectives with their intuitionistic counterparts and removes any occurrences of the exponential. This is denoted by $(-)^s$ and is given by

$$
\begin{array}{rcl}
\mathsf{bool}^s & \stackrel{\text{def}}{=} & \mathsf{bool} \\
(\phi \multimap \psi)^s & \stackrel{\text{def}}{=} & \phi^s \supset \psi^s \\
(\phi \otimes \psi)^s & \stackrel{\text{def}}{=} & \phi^s \wedge \psi^s \\
(!\phi)^s & \stackrel{\text{def}}{=} & \phi^s.
\end{array}
$$

It can be extended to terms (in context) as follows, where Γ^s denotes the application of the translation to all the members of Γ.

$$
\begin{array}{rcl}
|x\colon\phi \rhd x\colon\phi|^s & \stackrel{\text{def}}{=} & x\colon\phi^s \rhd x\colon\phi^s \\
|\emptyset \rhd b\colon \mathsf{bool}|^s & \stackrel{\text{def}}{=} & \emptyset \rhd b\colon \mathsf{bool} \\
|\emptyset \rhd \Omega^{\phi}\colon\phi|^s & \stackrel{\text{def}}{=} & \emptyset \rhd \Omega^{\phi^s}\colon\phi^s \\
|\Gamma \rhd \lambda x\colon\phi.M\colon\phi\multimap\psi|^s & \stackrel{\text{def}}{=} & \Gamma^s \rhd \lambda x\colon\phi^s.|M|^s\colon\phi^s \supset \psi^s \\
|\Gamma,\Delta \rhd MN\colon\psi|^s & \stackrel{\text{def}}{=} & \Gamma^s,\Delta^s \rhd |M|^s|N|^s\colon\psi^s \\
|\Gamma,\Delta \rhd M\otimes N\colon\phi\otimes\psi|^s & \stackrel{\text{def}}{=} & \Gamma^s,\Delta^s \rhd \langle|M|^s,|N|^s\rangle\colon\phi^s\wedge\psi^s \\
|\Gamma,\Delta \rhd \mathsf{let}\ M\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ N\colon\phi|^s & \stackrel{\text{def}}{=} & \Gamma^s,\Delta^s \rhd |N|^s[x := \mathsf{fst}(|M|^s), y := \mathsf{snd}(|M|^s)]\colon\phi^s \\
|\Gamma,\Delta \rhd \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P\colon\phi|^s & \stackrel{\text{def}}{=} & \Gamma^s,\Delta^s \rhd \mathsf{if}\ |M|^s\ \mathsf{then}\ |N|^s\ \mathsf{else}\ |P|^s\colon\phi^s \\
|\Gamma,\Delta \rhd \mathsf{discard}\ M\ \mathsf{in}\ N\colon\phi|^s & \stackrel{\text{def}}{=} & \Delta^s \rhd |N|^s\colon\phi^s \\
|\Gamma,\Delta \rhd \mathsf{copy}\ M\ \mathsf{as}\ x,y\ \mathsf{in}\ N\colon\psi|^s & \stackrel{\text{def}}{=} & \Gamma^s,\Delta^s \rhd |N|^s[x,y := |M|^s]\colon\psi^s \\
|\Gamma \rhd \mathsf{derelict}(M)\colon\phi|^s & \stackrel{\text{def}}{=} & \Gamma^s \rhd |M|^s\colon\phi^s \\
|\vec{\Gamma},\Delta \rhd \mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N\colon!\phi|^s & \stackrel{\text{def}}{=} & \vec{\Gamma}^s,\Delta^s \rhd |N|^s[\vec{x} := |\vec{M}|^s]\colon\phi^s
\end{array}
$$

3.2 Call-by-Name Semantics

The defining feature of call-by-name is that arguments are passed in unevaluated, *viz.*

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

Another feature is that pairs are considered to be values (we do not evaluate the elements of the pairs). Values are given by the inductive definition

$$
\begin{array}{rcl}
c & ::= & \mathtt{true}, \mathtt{false} \\
& | & \lambda x\colon\sigma.e \\
& | & \langle e,e\rangle.
\end{array}
$$

The (call-by-name) operational semantics are given in Figure 6.

Now we need to define both observational equivalence and applicative bisimularity for the PCF as well. Both Gordon [12] and Pitts [20] have offered definitions—here I shall follow those given by Pitts.[4]

A context is a PCF-term with typed hole(s) in it. One can carry over the definition of a linear context to the non-linear setting (the details are left to the reader). I shall adopt similar notation, *viz.*

$$\bullet(\Gamma)\colon\sigma;\Delta \rhd \mathcal{C}[\bullet]\colon\tau$$

to represent a (PCF) context of type τ, with free variables contained in Δ and with hole(s) $\bullet$, of type σ and free variable set given by Γ.

Definition 6. Given $\Gamma \rhd e\colon\sigma$ and $\Gamma \rhd f\colon\sigma$ we shall say that e *observationally refines* f, written $\Gamma \rhd e \sqsubseteq^{gnd}_{\sigma} f$, where for all closing boolean contexts, $\bullet(\Gamma)\colon\sigma;\emptyset \rhd \mathcal{C}[\bullet]\colon\mathsf{bool}$, if $\mathcal{C}[e] \Downarrow \mathtt{true}$ then $\mathcal{C}[f] \Downarrow \mathtt{true}$.

[4] I shall use the same symbols as for the linear counterparts. The types should ensure that there is no confusion.

$$b \Downarrow b \quad \lambda x{:}\sigma.e \Downarrow \lambda x{:}\sigma.e$$

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

$$\langle e, f\rangle \Downarrow \langle e, f\rangle$$

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

$$\frac{e \Downarrow \mathtt{true} \qquad f \Downarrow c}{\mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g \Downarrow c} \quad \frac{e \Downarrow \mathtt{false} \qquad g \Downarrow c}{\mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g \Downarrow c}$$

Figure 6: Call-By-Name Operational Semantics for PCF.

We write $\Gamma \rhd e \approx^{gnd}_{\sigma} f$ iff $\Gamma \rhd e \sqsubseteq^{gnd}_{\sigma} f$ and $\Gamma \rhd f \sqsubseteq^{gnd}_{\sigma} e$ and refer to this relation as *ground observational equivalence*.[5]

Definition 7. Given a family of (type-indexed) relations $R = (R_\sigma \subseteq Exp(\sigma) \times Exp(\sigma))$ between closed PCF-terms, we can define a family of relations $[R]_\sigma$ as follows.

- $e[R]_{\mathsf{bool}} f$ iff $\forall b.$ if $e \Downarrow b$ then $f \Downarrow b$,
- $e[R]_{\sigma\wedge\tau} f$ iff $\mathsf{fst}(e)\ R_\sigma\ \mathsf{fst}(f)$ and $\mathsf{snd}(e)\ R_\tau\ \mathsf{snd}(f)$,
- $e[R]_{\sigma\supset\tau} f$ 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 denoted by $\leq$, and referred to as *(PCF) applicative simularity*.

Applicative simularity is extended to open PCF-terms as follows.

$$x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n \rhd e \leq^{\circ}_{\tau} f \iff \forall g_i.\, \emptyset \rhd e[\vec{x} := \vec{g}] \leq_\tau f[\vec{x} := \vec{g}] \quad \text{where } \emptyset \rhd g_i{:}\sigma_i,$$

where the g_i are PCF-terms. Applicative bisimularity, written $\approx^{app}$ is defined as the symmetrisation of $\leq$, *viz.*

$$\Gamma \rhd e \approx^{app}_{\sigma} f \text{ iff } \Gamma \rhd e \leq^{\circ}_{\sigma} f \text{ and } \Gamma \rhd f \leq^{\circ}_{\sigma} e.$$

It is then possible to show that these two notions of program equivalence coincide.

Theorem 3. $\Gamma \rhd e \approx^{gnd}_{\sigma} f$ iff $\Gamma \rhd e \approx^{app}_{\sigma} f$.

Proof. An analogous proof is given in detail in Pitts's notes [20, §4]. ∎

[5] One could also develop a notion of observation at all types but observation at ground (boolean) type seems to be the norm.

4 The Call-by-Name Translation

In his seminal paper, Girard presented a translation of formulae from **IL** to **ILL** which he denoted by $(-)^\circ$. It has been folklore that this corresponds to a call-by-name translation. The translation is as follows.[6]

$$\begin{array}{rcl} \mathsf{bool}^\circ & \stackrel{\text{def}}{=} & \mathsf{bool} \\ (\sigma \supset \tau)^\circ & \stackrel{\text{def}}{=} & !\sigma^\circ \multimap \tau^\circ \\ (\sigma \wedge \tau)^\circ & \stackrel{\text{def}}{=} & !\sigma^\circ \otimes !\tau^\circ \end{array}$$

The operational intuition is that objects of type $!\phi$ are left unevaluated. Thus the translation of a function type $\sigma \supset \tau$ to $!\sigma^\circ \multimap \tau^\circ$ indicates that arguments are passed in unevaluated, *viz.* a call-by-name strategy. The translation can be given at the level of typing derivations as follows.

$$\begin{array}{rcl} |\vec{x}\colon \Gamma \rhd b\colon \mathsf{bool}|^\circ & \stackrel{\text{def}}{=} & \vec{x}\colon !\Gamma^\circ \rhd \mathsf{discard}\ \vec{x}\ \mathsf{in}\ b\colon \mathsf{bool} \\ |\vec{x}\colon \Gamma \rhd \Omega^\sigma\colon \sigma|^\circ & \stackrel{\text{def}}{=} & \vec{x}\colon !\Gamma^\circ \rhd \mathsf{discard}\ \vec{x}\ \mathsf{in}\ \Omega^{\sigma^\circ}\colon \sigma^\circ \\ |\vec{x}\colon \Gamma, y\colon \sigma \rhd y\colon \sigma|^\circ & \stackrel{\text{def}}{=} & \vec{x}\colon !\Gamma^\circ, y\colon !\sigma^\circ \rhd \mathsf{discard}\ \vec{x}\ \mathsf{in}\ \mathsf{derelict}(y)\colon \sigma^\circ \\ |\Gamma \rhd \lambda y\colon \sigma.e\colon \sigma \supset \tau|^\circ & \stackrel{\text{def}}{=} & !\Gamma^\circ \rhd \lambda y\colon !\sigma^\circ.|e|^\circ\colon !\sigma^\circ \multimap \tau^\circ \\ |\vec{x}\colon \Gamma \rhd ef\colon \tau|^\circ & \stackrel{\text{def}}{=} & \vec{x}\colon !\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))\colon \tau^\circ \\ |\vec{x}\colon \Gamma \rhd \langle e, f\rangle\colon \sigma \wedge \tau|^\circ & \stackrel{\text{def}}{=} & \vec{x}\colon !\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 \\ & & \quad (\mathsf{promote}\ \vec{x''}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ |f|^\circ)\colon !\sigma^\circ \otimes !\tau^\circ \\ |\Gamma \rhd \mathsf{fst}(e)\colon \sigma|^\circ & \stackrel{\text{def}}{=} & !\Gamma^\circ \rhd \mathsf{let}\ |e|^\circ\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ (\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x))\colon \sigma^\circ \\ |\Gamma \rhd \mathsf{snd}(e)\colon \tau|^\circ & \stackrel{\text{def}}{=} & !\Gamma^\circ \rhd \mathsf{let}\ |e|^\circ\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ (\mathsf{discard}\ x\ \mathsf{in}\ \mathsf{derelict}(y))\colon \tau^\circ \\ |\vec{x}\colon \Gamma \rhd \mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g\colon \sigma|^\circ & \stackrel{\text{def}}{=} & \vec{x}\colon !\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)\colon \sigma^\circ \end{array}$$

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

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

and

$$\begin{array}{rcl} |(\lambda x.\lambda y.x)\mathtt{true}|^\circ & \stackrel{\text{def}}{=} & (\lambda x.\lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x))\mathsf{promote}(\mathtt{true}), \\ |\lambda y.\mathtt{true}|^\circ & \stackrel{\text{def}}{=} & \lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathtt{true}; \end{array}$$

where I have used the shorthand $\mathsf{promote}(M)$ for $\mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ M$. However

$$\frac{\begin{array}{l}\lambda x.\lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x) \Downarrow \lambda x.\lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x) \\ \mathsf{promote}(\mathtt{true}) \Downarrow \mathsf{promote}(\mathtt{true}) \\ \lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(\mathsf{promote}(\mathtt{true})) \Downarrow \lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(\mathsf{promote}(\mathtt{true}))\end{array}}{(\lambda x.\lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x))\mathsf{promote}(\mathtt{true}) \Downarrow \lambda y.\mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(\mathsf{promote}(\mathtt{true}))}$$

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

If attention is restricted to programs which evaluates to booleans then a similar result does hold.

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

There are maps between PCF and linear PCF in both directions. The maps are related in the following sense.

Proposition 9. For all PCF-terms e, $||\Gamma \triangleright e\colon \sigma|^\circ|^s \equiv \Gamma \triangleright e\colon \sigma$.

Proof. By induction on the typing derivation $\Gamma \triangleright e\colon \sigma$. ∎

However there is little interesting to say about the composition of the maps in the other direction.[7] The $|-|^s$ translation erases all the information concerning the exponential, which are 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

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

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

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

but a moment's thought shows that this is not true; the $|-|^s$ translation does not even preserve values. For example the term

$$\mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ (\lambda x.M)N$$

is a (linear) value and thus evaluates to itself, but

$$|\mathsf{promote}\ -\ \mathsf{for}\ -\ \mathsf{in}\ (\lambda x.M)N|^s \stackrel{\text{def}}{=} (\lambda x.|M|^s)|N|^s$$

which contains a top-level redex. However, we can prove the converse to Proposition 8.

Proposition 10. If $|e|^\circ \Downarrow b$ then $e \Downarrow b$.

There is now enough information to consider whether the call-by-name translation preserves and reflects observational equivalence—these properties are commonly known as *full abstraction* and *adequacy* respectively. Surprisingly full abstraction *fails.*

Theorem 4. The call-by-name translation is *not* fully abstract, *viz.* there are PCF-terms e and f such that $\Gamma \triangleright e \approx^{gnd}_{\sigma} f$ and $!\Gamma^\circ \triangleright |e|^\circ \not\approx^{gnd}_{\sigma^\circ} |f|^\circ$.

Proof. In call-by-name PCF we have that $\Gamma \triangleright e \approx^{gnd}_{\sigma \wedge \tau} \langle \mathsf{fst}(e), \mathsf{snd}(e)\rangle$, for all PCF-terms e [20, Equation 25]. Consider the case when $e \equiv \Omega^{\mathsf{bool} \wedge \mathsf{bool}}$, thus

$$\begin{aligned} |\Omega|^\circ &\stackrel{\text{def}}{=} \Omega^{!\mathsf{bool} \otimes !\mathsf{bool}}, \text{ and} \\ |\langle \mathsf{fst}(\Omega), \mathsf{snd}(\Omega)\rangle|^\circ &\stackrel{\text{def}}{=} \mathsf{promote}(\mathsf{let}\ \Omega\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x)) \\ &\quad \otimes \\ &\quad \mathsf{promote}(\mathsf{let}\ \Omega\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ \mathsf{discard}\ x\ \mathsf{in}\ \mathsf{derelict}(y)). \end{aligned}$$

[7] This contrasts with the case for Ritter and Pitts [22] who consider translations between a fragment of SML and an idealised λ-calculus with references. There the translations are mutually inverse.

Unfortunately these two terms can be distinguished by the boolean context

$$\mathcal{L}[\bullet^{!\mathsf{bool}\otimes!\mathsf{bool}}] \stackrel{\text{def}}{=} \mathsf{let}\ \bullet\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ \mathsf{discard}\ x\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathtt{true}.$$

■

However we can prove that the call-by-name translation is adequate, the essence of which is given in the following proposition.

Proposition 11. If $\emptyset \triangleright |e|^\circ \approx^{gnd}_{\sigma^\circ} |f|^\circ$ then $\emptyset \triangleright e \approx^{gnd}_{\sigma} f$.

Proof. Form the set

$$\mathcal{S} \stackrel{\text{def}}{=} \{(e, f) \mid \emptyset \triangleright |e|^\circ \approx^{gnd}_{\sigma^\circ} |f|^\circ\}$$

and show that $\mathcal{S} \subseteq [\mathcal{S}]$. We consider the types of σ.

- $\sigma \equiv \mathsf{bool}$. By assumption $\emptyset \triangleright |e|^\circ \approx^{gnd}_{\mathsf{bool}} |f|^\circ$. Assume that $e \Downarrow b$ then by Proposition 8, $|e|^\circ \Downarrow b$ and then by taking the identity (linear) context we can conclude $|f|^\circ \Downarrow b$. From Proposition 10, $f \Downarrow b$ and we are done.
- $\sigma \equiv \sigma \supset \tau$. We have that $\emptyset \triangleright |e|^\circ \approx^{gnd}_{!\sigma^\circ \multimap \tau^\circ} |f|^\circ$. Take any context $\mathcal{L}[\bullet^{\tau^\circ}]$ and call $\mathcal{L}'$ the context which results from replacing the hole $\bullet^{\tau^\circ}$ with the term $\bullet^{!\sigma^\circ \multimap \tau^\circ}\mathsf{promote}(|g|^\circ)$, where g is an arbitrary PCF-term (of the appropriate type). Thus

$$\begin{aligned}
\mathcal{L}[|eg|^\circ] \Downarrow b &\iff \mathcal{L}[|e|^\circ \mathsf{promote}(|g|^\circ)] \Downarrow b \\
&\iff \mathcal{L}'[|e|^\circ] \Downarrow b \\
&\iff \mathcal{L}'[|f|^\circ] \Downarrow b \\
&\iff \mathcal{L}[|f|^\circ \mathsf{promote}(|g|^\circ)] \Downarrow b \\
&\iff \mathcal{L}[|fg|^\circ] \Downarrow b.
\end{aligned}$$

- $\sigma \equiv \sigma \wedge \tau$. We have by assumption $\emptyset \triangleright |e|^\circ \approx^{gnd}_{!\sigma^\circ \otimes !\tau^\circ} |f|^\circ$. Take any context $\mathcal{L}[\bullet^{\sigma^\circ}]$ and call $\mathcal{L}'$ the context which results from replacing the hole $\bullet^{\sigma^\circ}$ with the term $\mathsf{let}\ \bullet^{!\sigma^\circ \otimes !\tau^\circ}\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x)$. Thus

$$\begin{aligned}
\mathcal{L}[|\mathsf{fst}(e)|^\circ] \Downarrow b &\iff \mathcal{L}[\mathsf{let}\ |e|^\circ\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x)] \Downarrow b \\
&\iff \mathcal{L}'[|e|^\circ] \Downarrow b \\
&\iff \mathcal{L}'[|f|^\circ] \Downarrow b \\
&\iff \mathcal{L}[\mathsf{let}\ |f|^\circ\ \mathsf{be}\ x\otimes y\ \mathsf{in}\ \mathsf{discard}\ y\ \mathsf{in}\ \mathsf{derelict}(x)] \Downarrow b \\
&\iff \mathcal{L}[|\mathsf{fst}(f)|^\circ] \Downarrow b.
\end{aligned}$$

  A similar argument holds for $|\mathsf{snd}(e)|^\circ$.

■

Corollary 3. The call-by-name translation is adequate.

Discussion. Rather than translate PCF pairs into linear multiplicative pairs we could extend linear PCF with additive pairs and change the translation to

$$\begin{aligned}(\sigma \wedge \tau)^\circ &\stackrel{\text{def}}{=} \sigma^\circ \& \tau^\circ, \text{ and}\\ |\Gamma \rhd \langle e, f\rangle : \sigma \wedge \tau|^\circ &\stackrel{\text{def}}{=} !\Gamma^\circ \rhd \langle |e|^\circ, |f|^\circ\rangle : \sigma^\circ \& \tau^\circ\\ |\Gamma \rhd \mathsf{fst}(e) : \sigma|^\circ &\stackrel{\text{def}}{=} !\Gamma^\circ \rhd \mathsf{fst}(|e|^\circ) : \sigma^\circ\\ |\Gamma \rhd \mathsf{snd}(e) : \tau|^\circ &\stackrel{\text{def}}{=} !\Gamma^\circ \rhd \mathsf{snd}(|e|^\circ) : \tau^\circ.\end{aligned}$$

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

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

These two (translated) terms are easily seen to be applicatively similar (after a suitable reworking of the definition). It is still an open question as to whether this modified translation is fully abstract.

The reader will recall that at the end of §2 it was conjectured that the notion of applicative bisimularity and ground observation equivalence did *not* coincide for linear PCF (failure of operational extensionality). If this conjecture turns out to be false then it will entail that full abstraction fails for both call-by-name translations.

Lemma 11. If for all linear PCF-terms M and N, $\Gamma \rhd M \approx^{gnd}_{\phi} N$ iff $\Gamma \rhd M \approx^{app}_{\phi} N$ then full abstraction fails for the call-by-name translation.

Proof. In call-by-name PCF we have [20, Equation 133]

$$\emptyset \rhd \lambda x.\Omega \approx^{gnd}_{\mathsf{bool}\supset\mathsf{bool}} \Omega$$

but it is clear that $|\lambda x.\Omega|^\circ \stackrel{\text{def}}{=} \lambda x.\mathsf{discard}\ x\ \mathsf{in}\ \Omega \not\approx^{app} \Omega \stackrel{\text{def}}{=} |\Omega|^\circ$, and hence

$$\emptyset \rhd |\lambda x.\Omega|^\circ \not\approx^{gnd}_{!\mathsf{bool}\multimap\mathsf{bool}} |\Omega|^\circ.$$

■

5 A Linear SECD Machine

In this section I shall describe a simple implementation of linear PCF by a variant of Landin's SECD machine. For the most part this machine has previously been described by Abramsky [1] and implemented by Mackie [17]; although, as mentioned earlier, this was for a slightly different calculus. At the very least this section demonstrates that despite some of the syntactic complications of linear PCF, it can be quite easily implemented.

The machine consists of four stacks: (S)tack, (E)nvironment, (C)ode and (D)ump. In what follows I shall use SML list notation for the stacks; thus '::' for the cons operation, '[]' for the empty list, and '@' for the append operation.

Definition 8.

$$
\begin{array}{lcl}
S, E, \textsc{pushenv} :: C, D & \longrightarrow & \texttt{env}(E) :: S, E, C, D \\
\texttt{env}(v :: E) :: S, E', \textsc{hd} :: C, D & \longrightarrow & v :: S, E', C, D \\
\texttt{env}(v :: E) :: S, E', \textsc{tl} :: C, D & \longrightarrow & \texttt{env}(E) :: S, E', C, D \\
S, E, \textsc{true} :: C, D & \longrightarrow & \texttt{true} :: S, E, C, D \\
S, E, \textsc{false} :: C, D & \longrightarrow & \texttt{false} :: S, E, C, D \\
S, E, \textsc{bomb} :: C, D & \longrightarrow & S, E, \textsc{bomb} :: C, D \\
v :: S, E, \textsc{push} :: C, D & \longrightarrow & S, v :: E, C, D \\
s, v :: E, \textsc{pop} :: C, D & \longrightarrow & v :: S, E, C, D \\
v :: v' :: S, E, \textsc{tensor} :: C, D & \longrightarrow & \texttt{tensor}(v, v') :: S, E, C, D \\
\texttt{tensor}(v, v') :: S, E, \textsc{split} :: C, D & \longrightarrow & v :: v' :: S, E, C, D \\
S, E, \textsc{makefcl}(C') :: C, D & \longrightarrow & \texttt{fcl}(C', E) :: S, E, C, D \\
\texttt{fcl}(C', E') :: S, v :: E, \textsc{ap} :: C, D & \longrightarrow & [], v :: E', C', (S, E, C) :: D \\
v :: S, E, \textsc{ret} :: C, (S', E', C') :: D & \longrightarrow & v :: S', E', C', D \\
\texttt{true} :: S, E, \textsc{cond}(C', C'') :: C, D & \longrightarrow & [], E, C', (S, E, C) :: D \\
\texttt{false} :: S, E, \textsc{cond}(C', C'') :: C, D & \longrightarrow & [], E, C'', (S, E, C) :: D \\
S, [v_1, \ldots, v_n]@E, \textsc{makeecl}(n, C') :: C, D & \longrightarrow & \texttt{ecl}(C', [v_1, \ldots, v_n]) :: S, E, C, D \\
\texttt{ecl}(C', E') :: S, E, \textsc{der} :: C, D & \longrightarrow & [], E', C', (S, E, C) :: D \\
\texttt{ecl}(C', E') :: S, E, \textsc{disc} :: C, D & \longrightarrow & S, E, C, D \\
\texttt{ecl}(C', E') :: S, E, \textsc{dupl} :: C, D & \longrightarrow & \texttt{ecl}(C', E') :: \texttt{ecl}(C', E') :: S, E, C, D
\end{array}
$$

Figure 7: Transition Rules for the Linear SECD machine.

- An *instruction* is of the form

TRUE	FALSE	PUSH	POP	BOMB
TENSOR	SPLIT	MAKEFCL(c)	AP	
RET	COND(c, c')	MAKEECL(n, c)	DER	
DISC	DUPL	HD	TL	

  where n is a number and c, c' are *codes* (lists of instructions).

- A *value* is of the form

`true`	`false`	`tensor`(v, v')
`fcl`(c, e)	`ecl`(c, e)	

  where v, v' are values, c is a code and e is an *environment* (list of values).

The stacks are then of the form

- S: a list of values,
- E: a list of values,
- C: a list of instructions,
- D: a list of (S,E,C)-triples.

Computation steps are simply transition rules for each possible state of the machine. These are given in Figure 7.

One can now define a compilation from linear PCF-terms to SECD instructions. This is achieved by defining a function $\mathcal{S}(M, \vec{x})$, where M is a linear PCF-term and $\vec{x}$ is the list of free variables of M, which returns a list of instructions.

$$
\begin{array}{rcl}
\mathcal{S}(x,l) & \stackrel{\text{def}}{=} & [\text{PUSHENV}]@lookup(x,l) \\
\mathcal{S}(\texttt{true},l) & \stackrel{\text{def}}{=} & [\text{TRUE}] \\
\mathcal{S}(\texttt{false},l) & \stackrel{\text{def}}{=} & [\text{FALSE}] \\
\mathcal{S}(\Omega,l) & \stackrel{\text{def}}{=} & [\text{BOMB}] \\
\mathcal{S}(M \otimes N,l) & \stackrel{\text{def}}{=} & \mathcal{S}(M,l)@\mathcal{S}(N,l)@[\text{TENSOR}] \\
\mathcal{S}(\text{let } M \text{ be } x \otimes y \text{ in } N,l) & \stackrel{\text{def}}{=} & \mathcal{S}(M,l)@[\text{SPLIT,PUSH,PUSH}]@\mathcal{S}(N,y :: x :: l)@[\text{POP,POP}] \\
\mathcal{S}(\lambda x.M,l) & \stackrel{\text{def}}{=} & [\text{MAKEFCL}(\mathcal{S}(M,x :: l)@[\text{POP,RET}])] \\
\mathcal{S}(MN,l) & \stackrel{\text{def}}{=} & \mathcal{S}(N,l)@[\text{PUSH}]@\mathcal{S}(M,l)@[\text{AP}] \\
\mathcal{S}(\text{if } M \text{ then } N \text{ else } P,l) & \stackrel{\text{def}}{=} & \mathcal{S}(M,l)@[\text{COND}(\mathcal{S}(N,l),\mathcal{S}(P,l)),\text{RET}] \\
\mathcal{S}(\text{promote } \vec{M} \text{ for } \vec{x} \text{ in } N,l) & \stackrel{\text{def}}{=} & \mathcal{S}(M_n,l)@[\text{PUSH}]@\cdots@\mathcal{S}(M_1,l)@[\text{PUSH}]@ \\
 & & [\text{MAKEECL}(n,\mathcal{S}(N,l)@[\text{RET}])] \\
\mathcal{S}(\text{derelict}(M),l) & \stackrel{\text{def}}{=} & \mathcal{S}(M,l)@[\text{DER}] \\
\mathcal{S}(\text{discard } M \text{ in } N,l) & \stackrel{\text{def}}{=} & \mathcal{S}(M,l)@[\text{DISC}]@\mathcal{S}(N,l) \\
\mathcal{S}(\text{copy } M \text{ as } x,y \text{ in } N,l) & \stackrel{\text{def}}{=} & \mathcal{S}(M,l)@[\text{DUPL,PUSH,PUSH}]@\mathcal{S}(N,x :: y :: l)@[\text{POP,POP}]
\end{array}
$$

where $lookup(x, y :: l) =$ if $x \equiv y$ then [HD] else [TL]@$lookup(x,l)$

Following Abramsky [1], we can prove that this is a correct implementation of the operational semantics of Figure 2 in the following sense.

Theorem 5. If $M \Downarrow v$ then $\exists c.([],[],\mathcal{S}(M,[]),[]) \longrightarrow^* ([c],[],[],[])$ and $([],[],\mathcal{S}(v,[]),[]) \longrightarrow^* ([c],[],[],[])$.

Remark. One might be tempted to 'optimise' the compilation of the *Weakening* rule to

$$\mathcal{S}(\text{discard } M \text{ in } N,l) \stackrel{\text{def}}{=} \mathcal{S}(N,l).$$

Of course, with the presence of non-termination this would mean that the termination properties of the SECD machine would not match the operational semantics from Figure 2. For example, consider the term

$$\text{discard } \Omega \text{ in } \texttt{true};$$

clearly discard Ω in true $\Uparrow$ but

$$\mathcal{S}(\text{discard } \Omega \text{ in } \texttt{true},[]) \stackrel{\text{def}}{=} [\text{TRUE}],$$

which is a *terminating* program.

6 Conclusions

In this paper I have developed the operational theory of a linear PCF: the typed linear λ-calculus extended with booleans, conditional and non-terminating constants.[8] I have shown how to define a notion of context and, using this, two variants of observational equivalence. I then gave a co-inductive notion of program equivalence, applicative (bi)simularity, and showed that it coincided with one of the notions of observational equivalence. After recalling some of the details of an operational theory of a call-by-name PCF, I considered translations to and from linear PCF. I showed that the translation is adequate but not fully abstract. I then addressed more practical concerns by demonstrating how Landin's SECD machine can be adapted to execute linear PCF-terms.

One obvious outstanding piece of work is to give a co-inductive definition of program equivalence which does coincide with ground observational equivalence. I have also dodged the question of how to allow recursive definitions by opting instead for non-terminating constants. At the time of writing, there is still no real consensus for the correct form of recursion in the linear setting. Braüner [9] presents a comprehensive study of one proposal. Another possibility, currently being investigated, is the use of a form of *trace* operator [16]. However, whatever form linear recursion takes, I would expect it to be relatively straightforward to include it in this work. Another outstanding topic, currently being studied, is the call-by-value translation of PCF into linear PCF (which is a considerably more complicated translation).

Benton and Wadler [4] have shown that both Girard translations are related to Moggi's translations of the λ-calculus into the computational λ-calculus. I have been unable to find any 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 those in this paper. Maraist *et al.* [18] have also considered the Girard translations, but only for term reduction and also for a formulation which does not include syntax for the rules of *Weakening* and *Contraction*.

My original motivation for this work was not only practical but theoretical. I intend to investigate to what extent notions of observation are useful in proof theory. Current technology is quite weak: proofs are compared with respect to their cut normal forms (maybe modulo Kleene permutabilities). Is observational equivalence a useful notion? One possible test is to reconsider work by Schellinx [23] on optimal translations.

Future applications of this work is to study operational aspects of both the *classical* linear λ-calculus [7] and the *untyped* linear λ-calculus [8]. Another interesting exercise would be to give a categorical explanation of the treatment of contexts in Appendix A. On a more practical level, it would be interesting to develop more fully the implementation side of this work. The SECD machine, whilst a standard implementation technique, is not terribly efficient. Two possibilities are to develop a categorical abstract machine (which would be given by work on the categorical models of the linear λ-calculus [6]) and to develop a more low-level abstract machine which includes details of memory access.

Acknowledgments

I am grateful to Nick Benton, Andrew Gordon, Martin Hyland and Andrew Pitts for many useful discussions. Andrew Gordon and Valeria de Paiva offered useful comments on an

[8] Just before this paper went to press I received the PhD thesis of Braüner [10], who also studies a similar linear PCF but from a denotational perspective.

earlier draft of this paper.

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1–2):3–57, 1993. Previously Available as Department of Computing, Imperial College Technical Report 90/20, 1990.

[2] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (Extended Abstract). In *Proceedings of Conference on Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15, 1994.

[3] P.N. Benton, G.M. Bierman, V.C.V. de Paiva, and J.M.E. Hyland. A term calculus for intuitionistic linear logic. In M. Bezem and J.F. Groote, editors, *Proceedings of First International Conference on Typed λ-calculi and applications*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90, 1993.

[4] P.N. Benton and P. Wadler. Linear logic, monads and the lambda calculus. In *Proceedings of Symposium on Logic in Computer Science*, 1996.

[5] G.M. Bierman. *On Intuitionistic Linear Logic.* PhD thesis, Computer Laboratory, University of Cambridge, December 1993. Published as Computer Laboratory Technical Report 346, August 1994.

[6] G.M. Bierman. What is a categorical model of intuitionistic linear logic? In *Proceedings of Second International Conference on Typed λ-calculi and applications*, volume 902 of *Lecture Notes in Computer Science*, pages 78–93, April 1995. Previously available as Technical Report 333, University of Cambridge Computer Laboratory, March 1994.

[7] G.M. Bierman. A classical linear λ-calculus. Technical Report 401, Computer Laboratory, University of Cambridge, July 1996.

[8] G.M. Bierman. The untyped linear λ-calculus. Unpublished Manuscript, 1996.

[9] T. Braüner. The Girard translation extended with recursion. Technical Report RS–95–13, BRICS, Department of Computer Science, University of Århus, February 1995.

[10] T. Braüner. *An Axiomatic Approach to Adequacy.* PhD thesis, Department of Computer Science, University of Århus, Denmark, July 1996. Available as BRICS Technical Report DS–96–4, November 1996.

[11] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–101, 1987.

[12] A.D. Gordon. Bisimilarity as a theory of functional programming. In *Eleventh Annual Conference on Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1995.

[13] M. Hashimoto and A. Ohori. A typed context calculus. Technical Report RIMS–1098, Research Institute for Mathematical Sciences, Kyoto University, 1996.

[14] D.J. HOWE. Equality in lazy computation systems. In *Proceedings of Symposium on Logic in Computer Science*, pages 198–203, August 1989.

[15] S.L. PEYTON JONES. *The Implementation of Functional Programming Languages.* Prentice-Hall International, April 1987.

[16] A. JOYAL, R.H. STREET, AND D.R. VERITY. Traced monoidal categories. Technical Report 94/156, School of MPCE, Macquarie University, Sydney, Australia, August 1994.

[17] I. MACKIE. Lilac: A functional programming language based on linear logic. *Journal of Functional Programming*, 4(4):1–39, October 1994.

[18] J. MARAIST, M. ODERSKY, D.N. TURNER, AND P. WADLER. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. In *Proceedings of Conference on Mathematical Foundations of Programming Semantics*, Electronic Notes in Theoretical Computer Science. Elsevier, 1995.

[19] A.M. PITTS. 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, December 1994.

[20] A.M. PITTS. Operationally-based theories of program equivalence. Lecture Notes from course given at Isaac Newton Institute for Mathematical Sciences, Cambridge, England, November 1995.

[21] G.D. PLOTKIN. Type theory and recursion (extended abstract). In *Proceedings of Symposium on Logic in Computer Science*, page 374, 1993.

[22] E. RITTER AND A.M. PITTS. A fully abstract translation between a λ-calculus with reference types and Standard ML. In *Proceedings of Second International Conference on Typed λ-calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 397–413, April 1995.

[23] H. SCHELLINX. *The Noble Art of Linear Decorating.* PhD thesis, University of Amsterdam, February 1994.

A Linear Contexts

To formalise the notion of placing terms into larger programs we define the notion of 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 of terms for holes is permitted to capture free variables, *pace* substitution of terms for variables.

For PCF a traditional treatment (e.g. [20, Page 13]) is first to extend the syntactic class of terms to allow holes (here written '•') and add a new typing rule[9]

$$\bullet\colon\sigma, \Gamma \rhd \bullet\colon\sigma.$$

For linear PCF we would add the rule

$$\bullet\colon\phi \rhd \bullet\colon\phi.$$

However it is not hard to see that this is insufficient. One can not even form the linear context

$$\bullet\colon\phi \rhd \lambda x\colon\phi.\bullet\colon\phi\multimap\phi.$$

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 I propose to parameterise holes with these free variables. Thus the typing rule for holes becomes

$$\bullet(\vec{x}\colon\Gamma)\colon\phi; \vec{x}\colon\Gamma \rhd \bullet(\vec{x}\colon\Gamma)\colon\phi.$$

I separate the holes and variables in the antecedent with a semi-colon but this is simply a matter of hygiene. The typing rules for contexts are given in Figure 8. The earlier example is thus well-typed, *viz.*

$$\frac{\bullet(x\colon\phi)\colon\phi; x\colon\phi \rhd \bullet(x\colon\phi)\colon\phi}{\bullet(x\colon\phi)\colon\phi \rhd \lambda x\colon\phi.\ \bullet\ (x\colon\phi)\colon\phi\multimap\phi}\ (\multimap_{\mathcal{I}}).$$

The action of placing a term (actually, context) for a hole is then given by the rule

$$\frac{\mathcal{H}; \Gamma \rhd M\colon\phi \qquad \mathcal{H}', \bullet(\Gamma)\colon\phi; \Delta \rhd N\colon\psi}{\mathcal{H}, \mathcal{H}'; \Delta \rhd N[M/\bullet]\colon\psi}\ \textit{Placement.}$$

Thus one can only place a term, M, for the hole $\bullet(\Gamma)$ if its set of free variables is Γ. The result of this placement is then defined by induction on the structure of N. If a context N has only one hole $\bullet$, we often write it as $\mathcal{L}[\bullet]$, or even $\mathcal{L}[\bullet^{\phi}]$ and the result of placing another context M for the hole as $\mathcal{L}[M]$.

It should be noted that there is nothing inherently linear about this treatment of contexts, indeed I suggest its use with any calculus. A number of other people have suggested extensions to the notion of context; for example, Pitts [19] and Hashimoto and Ohori [13].

[9]The recording of the hole in the antecedent is often omitted.

$$\emptyset; x\colon \phi \triangleright x\colon \phi \qquad \emptyset; \emptyset \triangleright b\colon \mathsf{bool} \qquad \emptyset; \emptyset \triangleright \Omega^{\phi}\colon \phi$$

$$\bullet(\vec{x}\colon \Gamma)\colon \phi; \vec{x}\colon \Gamma \triangleright \bullet(\vec{x}\colon \Gamma)\colon \phi$$

$$\frac{\mathcal{H}; \Gamma, x\colon \phi \triangleright M\colon \psi}{\mathcal{H}; \Gamma \triangleright \lambda x\colon \phi.M\colon \phi \multimap \psi}\ (\multimap_{\mathcal{I}}) \qquad \frac{\mathcal{H}; \Gamma \triangleright M\colon \phi \multimap \psi \qquad \mathcal{H}'; \Delta \triangleright N\colon \phi}{\mathcal{H}, \mathcal{H}'; \Gamma, \Delta \triangleright MN\colon \psi}\ (\multimap_{\mathcal{E}})$$

$$\frac{\mathcal{H}_1; \Gamma_1 \triangleright M_1\colon !\phi_1 \cdots \mathcal{H}_n; \Gamma_n \triangleright M_n\colon !\phi_n \quad \mathcal{H}'; x_1\colon !\phi_1, \ldots, x_n\colon !\phi_n \triangleright N\colon \psi}{\mathcal{H}_1, \ldots, \mathcal{H}_n, \mathcal{H}'; \Gamma_1, \ldots, \Gamma_n \triangleright \mathsf{promote}\ \vec{M}\ \mathsf{for}\ \vec{x}\ \mathsf{in}\ N\colon !\psi}\ \textit{Promotion}$$

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon !\phi}{\mathcal{H}; \Gamma \triangleright \mathsf{derelict}(M)\colon \phi}\ \textit{Dereliction}$$

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon !\phi \qquad \mathcal{H}'; \Delta \triangleright N\colon \psi}{\mathcal{H}, \mathcal{H}'; \Gamma, \Delta \triangleright \mathsf{discard}\ M\ \mathsf{in}\ N\colon \psi}\ \textit{Weakening}$$

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon !\phi \qquad \mathcal{H}'; \Delta, x\colon !\phi, y\colon !\phi \triangleright N\colon \psi}{\mathcal{H}, \mathcal{H}'; \Gamma, \Delta \triangleright \mathsf{copy}\ M\ \mathsf{as}\ x, y\ \mathsf{in}\ N\colon \psi}\ \textit{Contraction}$$

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon \phi \qquad \mathcal{H}'; \Delta \triangleright N\colon \psi}{\mathcal{H}, \mathcal{H}'; \Gamma, \Delta \triangleright M \otimes N\colon \phi \otimes \psi}\ (\otimes_{\mathcal{I}})$$

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon \phi \otimes \psi \qquad \mathcal{H}'; \Delta, x\colon \phi, y\colon \psi \triangleright N\colon \varphi}{\mathcal{H}, \mathcal{H}'; \Gamma, \Delta \triangleright \mathsf{let}\ M\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ N\colon \varphi}\ (\otimes_{\mathcal{E}})$$

$$\frac{\mathcal{H}; \Gamma \triangleright M\colon \mathsf{bool} \qquad \mathcal{H}'; \Delta \triangleright N\colon \phi \qquad \mathcal{H}'; \Delta \triangleright P\colon \phi}{\mathcal{H}, \mathcal{H}'; \Gamma, \Delta \triangleright \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P\colon \phi}\ \textit{Conditional}$$

Figure 8: Type Assignment for Linear Contexts.