How Linear is Howe?

Roy L. Crole

email: rlc3@mcs.le.ac.uk
http://www.mcs.le.ac.uk/~rlc3

April 18, 1997

Abstract

In recent years there has been much progress in the use of operationally based methods for reasoning about programs. Notions such as contextual equivalence and bisimilarity, which are historically more usually associated with functional programming and concurrency respectively, have been shown to be very closely connected, and such connections can be seen throughout the functional, concurrent and indeed imperative paradigms. In this paper we show that operational techniques from these three settings also apply in a *linear* functional setting, by proving that in a simple linear type theory involving function types and tensor types, one can establish *linear* contextual equivalences by coinductive techniques.

A version of this paper is due to appear in the Proceedings of *Theory and Formal Methods of Computing '96*.

1 Introduction

During the past five or so years, many advances in the theory and practice of operational methods for proving program equivalences have been made. Many of the fundamental ideas which are employed in the underlying theory originated in the world of concurrency [Mil89]. Abramsky [Abr90b] was the first person to investigate such ideas in the setting of functional programming. A technique which underpins the work in this paper originated with Howe [How89]. This technique has been significantly refined and adapted by Gordon [Gor95], Pitts [Pit95], Sands [San91] and others, and the results presented in this paper have been developed using ideas from those in loc. cit.

In this paper we study a linear type theory whose equational theories correspond to autonomous categories. Here we shall not be concerned with such equational theories, but instead with an operational semantics for the expressions of the linear type theory. Such operational notions are discussed by Abramsky [Abr90a]. In this work, Abramsky notes that the expressions of a linear type theory have a natural operational evaluation strategy, which possesses a mixture of both lazy and eager computations. While his observations are very persuasive, there are no absolute reasons for utilising the strategies he suggests. Ironically, for the fragment of linear type theory considered in this paper, the "natural" evaluation strategy would be eager (see Abramsky [Abr90a]), but we shall only study lazy evaluations, as this simplifies the technical details of many of the proofs. However, our results remain true for eager evaluations.

Why are we studying a linear system? Linearity has proved to be a useful framework for discussions of resource control and optimisation in Computer Science. As such it seems sensible to think about methods for reasoning about linear systems. That is our aim here, and on the way to our result, it is interesting to note many of the more delicate technical issues which arise due to linearity constraints.

The structure of the paper is as follows: In Section 2 we define the linear type theory we wish to study. In Section 3 we define a linear contextual preorder. In Section 4 we define similarity for the linear type theory. In Section 5 we show that similarity is a linear precongruence, and prove our main theorem, namely that the linear contextual preorder contains similarity. In the final section we draw our conclusions. Finally, note that due to shortage of space, many ideas and proofs are only sketched; unfortunately one of the points of this paper is to illustrate technical problems due to linearity, and these too are only sketched.

2 A Linear Type Theory

We shall specify a linear type theory by giving its types, expressions, a type assignment relation, and a (lazy) evaluation semantics. The set of (linear) types is given by the grammar $\sigma ::= I \mid \sigma \multimap \sigma \mid \sigma \otimes \sigma$. The set of **terms** is defined by the following grammar:

$$M ::= x \mid \lambda x.M \mid M\,M \mid * \mid M \otimes M \mid \mathsf{split}\ M\ \mathsf{as}\ x \otimes x\ \mathsf{in}\ M$$

We define the notion of a *bound* or *binding* occurrence of any variable v in any term T as follows: Let M and S range over expressions, and v and x over variables. An occurrence of a variable v in a term T is **binding** if the occurrence is in a subterm of the form $\lambda v.S$ or $\mathsf{split}\ M\ \mathsf{as}\ v \otimes x\ \mathsf{in}\ S$ or $\mathsf{split}\ M\ \mathsf{as}\ x \otimes v\ \mathsf{in}\ S$ and moreover the occurrence is not in M or S. In each case, S is the **scope** of v. An occurrence of a variable v in a term T is **bound** if it occurs in a subterm of the form $\lambda v.S$ or $\mathsf{split}\ M\ \mathsf{as}\ v \otimes x\ \mathsf{in}\ S$ or $\mathsf{split}\ M\ \mathsf{as}\ x \otimes v\ \mathsf{in}\ S$, and v is not a binding occurrence. An occurrence of v in T is **free** if it is neither binding nor bound.

The terms described in this paper require a very careful treatment of variable binding and substitution. We could adopt a Martin-Löf style metalanguage [NPS90] for the presentation of the expressions of our type theory, but the details are omitted due to space constraints. Using such a metalanguage it is standard to define a **simultaneous substitution function** whose action is denoted by

$$(N, M_1, \ldots, M_n, x_1, \ldots, x_n) \mapsto N[M_1, \ldots, M_n/x_1, \ldots, x_n]$$

where $N[M_1, \ldots, M_n/x_1, \ldots, x_n]$ is a term described as "the term N with the (finite list) of terms $M_1, \ldots, M_n$ simultaneously replacing the (finite list) of variables $x_1, \ldots, x_n$". We shall sometimes write $N[M_i/x_i]$ when the meaning is clear. Given the existence of the substitution function, one can then inductively define $\sim_\alpha$-**equivalence** of terms (see for example Pitts [Pit94] or Crole [Cro95]).

As usual, an **environment** Γ is a list of (variable, type) pairs in which each of the variables is required to be distinct. A typical environment will be written $x_1\colon \sigma_1, \ldots, x_n\colon \sigma_n$. We shall define a **type assignment** relation which takes the form $\Gamma \vdash M : \sigma$. The (obvious/usual) rules are omitted from this report—see for example [Abr90a]. We shall let $Exp_\sigma(\Gamma)$ denote the set of terms which can be assigned the type σ in the environment Γ, identified up to $\sim_\alpha$-equivalence. We shall write $M : \sigma$ instead of $[\,] \vdash M : \sigma$ and we write Exp_σ instead of $Exp_\sigma([\,])$. Given that Γ and Δ are environments, we shall write Γ, Δ to mean that the latter is indeed a valid environment (so in particular that $var(\Gamma) \cap var(\Delta) = \varnothing$). We shall write M for both a term and the expression which it represents, as is the usual practice. The following is crucial for the results in this paper, and its proof requires the use of simultaneous substitution. The proof is also quite delicate due to linearity, but space prevents a detailed exposition.

Theorem 2.1 If Γ, Δ is an environment, then there is a well-defined function of the form $Exp_\tau(\Gamma, x\colon \sigma) \times Exp_\sigma(\Delta) \longrightarrow Exp_\tau(\Gamma, \Delta)$ whose action is written $(M, N) \mapsto M[N/x]$.

Lemma 2.2 For any type assignment $\Gamma \vdash M : \sigma$, $fv(M) = var(\Gamma)$.

We shall now define a notion of a *program*, *value* and an evaluation relation style *operational semantics*. The intuition and motivation for these concepts is now quite standard (see for example Abramsky [Abr90a] or Crole [Cro95]) and so we just give the definitions. A **program** (of type σ) is an element of Exp_σ for some type σ; P and Q range over programs. The set of **values** is given by those V generated by the grammar $V ::= * \mid P \otimes Q \mid \lambda x.M$ which are themselves programs (of some type σ). We define the lazy evaluation relation $\Downarrow$ between the set of all programs and the set of all values, with typical relationships denoted by $P \Downarrow V$, by the usual rules which are omitted.

3 Contextual Equivalence

We introduce a notion of contextual equivalence in a linear setting. The fundamental notion of a program context as some code with a hole is used here, but contexts

$$\frac{}{[\,] \vdash^{[\,]}_{[\,]} -_\rho : \rho} \qquad \frac{\Delta, x{:}\,\sigma_1, y{:}\,\sigma_2 \vdash^\beta_\eta \mathcal{C} : \sigma}{\Gamma, \Delta \vdash^\beta_{\eta, x:\sigma_1, y:\sigma_2} \textsf{split } M \textsf{ as } x \otimes y \textsf{ in } \mathcal{C} : \sigma}\,[\Gamma \vdash M : \sigma_1 \otimes \sigma_2]$$

$$\frac{\Delta \vdash^\beta_\eta \mathcal{C} : \sigma}{\Gamma, \Delta \vdash^{\beta, x:\sigma_1, y:\sigma_2}_\eta \textsf{split } M \textsf{ as } x \otimes y \textsf{ in } \mathcal{C} : \sigma}\,[\Gamma \vdash M : \sigma_1 \otimes \sigma_2,\ x, y \notin \Delta,\ x, y \notin \eta](\dagger)$$

$$\frac{\Gamma \vdash^\beta_\eta \mathcal{C} : \sigma}{\Gamma, \Delta \vdash^\beta_\eta \textsf{split } \mathcal{C} \textsf{ as } x \otimes y \textsf{ in } M : \sigma}\,[\Delta, x{:}\,\sigma_1, y{:}\,\sigma_2 \vdash M : \sigma]$$

Table 1: Examples of Rules for Generating Linear Contexts $\Gamma \vdash^\beta_\eta \mathcal{C} : \sigma$

are required to possess a single hole. However, because of the constraints linearity imposes on the occurrences of variables in expressions, we have to be extremely careful in our definition of an "expression with a hole". We first define *pre-contexts* which are un-typed templates for typed contexts: A **pre-context** is given by the grammar

$$\mathcal{C} ::= -_\rho \mid \lambda x.\mathcal{C} \mid \mathcal{C}\, M \mid M\, \mathcal{C} \mid \mathcal{C} \otimes M \mid M \otimes \mathcal{C} \mid \textsf{split } \mathcal{C} \textsf{ as } x \otimes y \textsf{ in } M \mid \textsf{split } M \textsf{ as } x \otimes y \textsf{ in } \mathcal{C}$$

where ρ ranges over the linear types, and M over terms. A **linear context** is one which takes the form $\Gamma \vdash^\beta_\eta \mathcal{C} : \sigma$ and is generated by rules such as those in Table 1. In reading the rules, they apply only when any Γ or β is an environment. Each η is a list of (variable,type) pairs, but need not be an environment.

The intended meaning of a linear context $\Gamma \vdash^\beta_\eta \mathcal{C} : \sigma$ is that $\mathcal{C}$ denotes a linear expression with one hole (denoted by $-_\rho$) whose free variables are exactly those appearing in Γ. For each $\Gamma \vdash^\beta_\eta \mathcal{C} : \sigma$ the following statements hold:

- β is an environment which lists exactly those binding variables whose scope includes $-_\rho$ but whose scope does not include any occurrence of the binding variable. We call the variables in β **context binding variables**.
- Each η is a list of (variable, type) pairs which lists each binding variable v whose scope includes $-_\rho$ *and* one bound occurrence of the binding variable v.

Each η only plays a role in the construction of linear contexts, ensuring that under appropriate conditions we can substitute a (representative of) a linear expression for a hole and guarantee that a linear type assignment results. Each β lists those variables which, if they occur free in an expression and the expression is substituted for $-_\rho$, will become bound in the resulting expression. These ideas are made precise in the next lemma, in which if $\mathcal{C}$ is a pre-context with a hole $-_\rho$, which we denote by $\mathcal{C}[-_\rho]$, and $M = [T]_{\sim_\alpha}$ is an expression of type ρ, $\mathcal{C}[M]$ denotes the expression given by $[\mathcal{C}[T]]_{\sim_\alpha}$ where $\mathcal{C}[T]$ means "replace the hole $-_\rho$ by the term T".

Lemma 3.1 For any linear context of the form $\Gamma \vdash^\beta_\eta \mathcal{C}[-_\rho] : \tau$ and expression $M \in \mathit{Exp}_\rho(\beta, \Delta)$, where Γ, Δ and η, Δ, we have $\mathcal{C}[M] \in \mathit{Exp}_\tau(\Gamma, \Delta)$.

Proof Induction on the derivation of the linear context. We illustrate for the rule $(\dagger)$. Let us suppose that we have

$$\beta, x{:}\, \sigma_1, y{:}\, \sigma_2, \Theta \vdash M' : \rho \qquad (\Gamma, \Delta), \Theta \quad \ldots (1) \qquad \eta, \Theta$$

It follows from the well-formedness of the environments, and the assumption that $x, y \notin \mathit{var}(\Delta)$ and $x, y \notin \mathit{var}(\eta)$ for each instance of $(\dagger)$, that

$$\Delta, (x{:}\, \sigma_1, y{:}\, \sigma_2, \Theta) \qquad\qquad \eta, (x{:}\, \sigma_1, y{:}\, \sigma_2, \Theta)$$

Thus by induction we have $\Delta, x{:}\, \sigma_1, y{:}\, \sigma_2, \Theta \vdash \mathcal{C}[M'] : \sigma$, and so using (1), and the rules for linear type assignments (including exchange) we may deduce

$$\Gamma, \Delta, \Theta \vdash \mathsf{split}\ M\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ \mathcal{C}[M'] : \sigma$$

$\square$

Corollary 3.2 For any linear context of the form $[\,] \vdash^\beta_\eta \mathcal{C}[-_\rho] : \tau$ and type assignment of the form $\beta \vdash M : \rho$, we have $\mathcal{C}[M] : \tau$.

For any program P, $P \Downarrow$ means $\exists V. P \Downarrow V$. We now write $\Gamma \vdash M \leq M' {:}\, \sigma$ just in case

(i) $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M' : \sigma$; and

(ii) for all linear contexts of the form $[\,] \vdash^\Gamma_\eta \mathcal{C}[-_\sigma] : \tau$, we have $\mathcal{C}[M] \Downarrow$ implies $\mathcal{C}[M'] \Downarrow$.

We have thus defined a family of relations $(\leq_{\Gamma,\sigma} \subseteq \mathit{Exp}_\sigma(\Gamma) \times \mathit{Exp}_\sigma(\Gamma) \mid \Gamma, \sigma)$ which we refer to as the **linear contextual preorder**. If $(M, M') \in \leq_{\Gamma,\sigma}$ we say that M is **contextually below** M'. We now aim to give an alternative, coinductive method for establishing instances of the linear contextual preorder.

4 Similarity

We can define a notion of **similarity**, denoted by $\preccurlyeq$, which is a type indexed family of binary relations between programs of the same type:

$$\preccurlyeq \; = (\preccurlyeq_\sigma \mid \sigma) \in \Pi_\sigma \mathcal{P}(\mathit{Exp}_\sigma \times \mathit{Exp}_\sigma)$$

Note that when the powerset is given the inclusion partial order, it becomes a complete lattice, and thus the product set is a complete lattice with the pointwise partial order—we use $\lhd$ to denote the order on the product. The definition of $\preccurlyeq$ is given by the greatest pre-fixed point of a certain monotone function of the form $\Phi : \Pi_\sigma \mathcal{P}(Exp_\sigma \times Exp_\sigma) \longrightarrow \Pi_\sigma \mathcal{P}(Exp_\sigma \times Exp_\sigma)$ and as usual such a pre-fixed point exists by the Knaster-Tarski theorem. Given any $R \in \Pi_\sigma \mathcal{P}(Exp_\sigma \times Exp_\sigma)$, we define $\Phi(R)$ by the following clauses:

- $P \Phi(R)_I P' \iff P \Downarrow * \Longrightarrow P' \Downarrow *$.
- $P \Phi(R)_{\sigma \multimap \sigma'} P' \iff$

$$(\exists M. P \Downarrow \lambda x.M) \Longrightarrow ((\exists N. P' \Downarrow \lambda x.N) \text{ and } (\forall Q.\ M[Q/x]\ R_{\sigma'}\ N[Q/x])).$$

- $P \Phi(R)_{\sigma_1 \otimes \sigma_2} P' \iff (\exists P_1, P_2. P \Downarrow P_1 \otimes P_2) \Longrightarrow (\exists P_1', P_2'. P' \Downarrow P_1' \otimes P_2'$ and $P_1\ R_{\sigma_1}\ P_1'$ and $P_2\ R_{\sigma_2}\ P_2')$.

This kind of definition is becoming standard—see for example Crole and Gordon [CG96], Gordon [Gor95], or Pitts [Pit95]. It is easy to verify that Φ is monotone, and thus we can define $\preccurlyeq \stackrel{\text{def}}{=} \nu\ R\ .\Phi(R)$. We thus have a principle of **coinduction**: in order to prove that for any set A we have $A \lhd \preccurlyeq$, all we need to prove is that $A \lhd \Phi(A)$.

Proposition 4.1 For every type σ, $\preccurlyeq_\sigma$ is a preorder.

Proof We can prove that $Id_{\Pi_\sigma \mathcal{P}(Exp_\sigma \times Exp_\sigma)} \lhd \preccurlyeq$ and $(\preccurlyeq \circ \preccurlyeq) \lhd \preccurlyeq$ by coinduction, which is precisely that $\preccurlyeq$ is a preorder. □

We write $\Gamma \vdash M \preccurlyeq^\circ M' : \sigma$ just in case

(i) $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M' : \sigma$; and

(ii) $\forall P_i \in Exp_{\sigma_i}.\ M[P_i/x_i] \preccurlyeq_\sigma M'[P_i/x_i]$, where the types σ_i are those appearing in Γ.

We have thus defined a family of relations $(\preccurlyeq^\circ_{\Gamma,\sigma} \subseteq Exp_\sigma(\Gamma) \times Exp_\sigma(\Gamma) \mid \Gamma, \sigma)$ which we refer to as **open similarity**. If $(M, M') \in \preccurlyeq^\circ_{\Gamma,\sigma}$ we say that M is **similar** to M'.

The main theorem is that the linear contextual preorder contains open similarity:

Theorem 4.2 For all Γ, M, M' and σ, we have $\Gamma \vdash M \preccurlyeq^\circ M' : \sigma$ implies $\Gamma \vdash M \leq M' : \sigma$.

We prove this by: Defining linear precongruence; defining a Howe relation; proving some properties of the Howe relation; deducing that the Howe relation and open similarity coincide; finally deducing that similarity is a linear precongruence and thus that the linear contextual preorder contains open similarity (at each type). *Note: Proofs are only sketched, or even omitted when routine.*

5 Linear Precongruence of Similarity

The major step in proving Theorem 4.2 in a non-linear setting involves a proof that similarity is a precongruence using methods due to Howe [How89], [How93] and Gordon [Gor94]. However, the usual definition of precongruence is intimately tied to the properties of weakening, substitution and strengthening (the latter is usually derivable—see page 36 of [Pit95]). In a linear setting we shall work with *linear precongruence*. This is similar to the usual notion of precongruence, but the weakening and strengthening requirements are dropped, and the usual substitution properties are modified.

Remark 5.1 In a non-linear setting the strengthening property is crucial to the proof that similarity is a precongruence—it amounts to the inductive base case in the proof of (the non-linear analogue of) Lemma 5.8. In such systems in which not all types are inhabited, strengthening is included as an explicit property in the definition of precongruence. (If all types are inhabited, it happens to be a derivable property.) In this paper, not all types are inhabited. However, the lack of strengthening is not a problem in the non-linear setting because linear contexts explicitly list their binding variables. Thus strengthening is not required in the base case of the proof of Lemma 5.8.

A family of binary relations of the form $(R_{\Gamma,\sigma} \subseteq \mathit{Exp}_\sigma(\Gamma) \times \mathit{Exp}_\sigma(\Gamma) \mid \Gamma, \sigma)$, for which we shall write $\Gamma \vdash M\ R\ M' \colon \sigma$ instead of $(M, M') \in R_{\Gamma,\sigma}$ and $M\ R_\sigma\ M'$ instead of $(M, M') \in R_{[],\sigma}$, is called a **linear precongruence** if:

(a) For any type assignment $\Gamma \vdash M : \sigma$ and relationship $\Delta, x\colon\sigma \vdash N\ R\ N' \colon \sigma'$ where Γ, Δ, we have $\Gamma, \Delta \vdash N[M/x]\ R\ N'[M/x] \colon \sigma'$.

(b) For any type assignment of the form $\Delta, x\colon\sigma \vdash N : \sigma'$ and relationship $\Gamma \vdash M\ R\ M' \colon \sigma$ where Γ, Δ, we have $\Gamma, \Delta \vdash N[M/x]\ R\ N[M'/x] \colon \sigma'$.

(c) For any $\Gamma \vdash M : \sigma$ we have $\Gamma \vdash M\ R\ M \colon \sigma$.

(d) For any $\Gamma \vdash M\ R\ M' \colon \sigma$ and $\Gamma \vdash M'\ R\ M'' \colon \sigma$ we have $\Gamma \vdash M\ R\ M'' \colon \sigma$.

(e) For any $\Gamma, x\colon\sigma \vdash M\ R\ M' \colon \sigma'$ we have $\Gamma \vdash \lambda x.M\ R\ \lambda x.M' \colon \sigma \multimap \sigma'$.

(f) For any $\Gamma \vdash M\ R\ M' \colon \sigma_1 \otimes \sigma_2$ and $\Delta, x\colon\sigma_1, y\colon\sigma_2 \vdash N\ R\ N' \colon \sigma$ where Γ, Δ we have $\Gamma, \Delta \vdash \mathsf{split}\ M\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ N\ R\ \mathsf{split}\ M'\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ N' \colon \sigma$.

(g) The remaining properties of being a precongruence hold (see page 35 of [Pit95]) but modified for linear environments.

In order to prove that similarity is a linear precongruence, we adopt a linearised version of Howe's method. We inductively define a family of binary relations of the form $(\preccurlyeq^{\bullet}_{\Gamma,\sigma} \subseteq \mathit{Exp}_\sigma(\Gamma) \times \mathit{Exp}_\sigma(\Gamma) \mid \Gamma, \sigma)$. The definition is quite similar to that found in [Gor95], except for the "linearisation of contexts", and example rules appear in Table 2.

$$\frac{}{[\,] \vdash * \precsim^{\bullet} M\colon I}\,[\,[\,] \vdash * \precsim^{\circ} M\colon I] \qquad \frac{}{x\colon\sigma \vdash x \precsim^{\bullet} M\colon \sigma}\,[x\colon\sigma \vdash x \precsim^{\circ} M\colon\sigma]$$

$$\frac{\Gamma \vdash M_1 \precsim^{\bullet} M_1'\colon \sigma_1 \quad \Delta \vdash M_2 \precsim^{\bullet} M_2'\colon \sigma_2}{\Gamma, \Delta \vdash M_1 \otimes M_2 \precsim^{\bullet} N\colon \sigma_1 \otimes \sigma_2}\,[\Gamma, \Delta \vdash M_1' \otimes M_2' \precsim^{\circ} N\colon \sigma_1 \otimes \sigma_2]$$

$$\frac{\begin{cases} \Gamma \vdash M_1 \precsim^{\bullet} M_1'\colon \sigma_1 \otimes \sigma_2 \\ \Delta, x\colon\sigma_1, y\colon\sigma_2 \vdash M_2 \precsim^{\bullet} M_2'\colon \sigma \end{cases}}{\Delta, \Gamma \vdash \mathsf{split}\ M_1\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ M_2 \precsim^{\bullet} N\colon \sigma}\,[\Delta, \Gamma \vdash \mathsf{split}\ M_1'\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ M_2' \precsim^{\circ} N\colon \sigma]$$

Table 2: Examples of Rules Generating the Howe Relation $\Gamma \vdash M \precsim^{\bullet} M'\colon \sigma$

Lemma 5.2 The family of relations $(\precsim^{\bullet}_{\Gamma,\sigma} \mid \Gamma, \sigma)$ satisfy the linear precongruence substitution clauses (a) and (b).

Proof This follows from the definition of open similarity, together with the properties of the environments. Suppose that $\Gamma \vdash M : \sigma$ and $\Delta, x\colon\sigma \vdash N \precsim^{\circ} N'\colon\sigma'$ where Γ, Δ. Let us write x_i for the variables appearing in Γ, and y_j for those in Δ. Thus $\forall j. x \neq y_j$ (though possibly $\exists i. x = x_i$). Pick arbitrary programs P_i and Q_j whose types match those of the variables x_i and y_j, for each i and j. We have $\vdash M[P_i/x_i] : \sigma$. Thus by definition of open similarity, we have

$$N[Q_j, M[P_i/x_i] \;/\; y_j, x] \precsim_{\sigma'} N'[Q_j, M[P_i/x_i] \;/\; y_j, x]$$

which on re-arranging is

$$N[M/x][P_i, Q_j \;/\; x_i, y_j] \precsim_{\sigma'} N'[M/x][P_i, Q_j \;/\; x_i, y_j]$$

As the programs were arbitrary, we have shown that

$$\Gamma, \Delta \vdash N[M/x] \precsim^{\circ} N'[M/x]\colon \sigma'$$

This shows substitution clause (a); the verification of (b) is similar.

$\square$

Lemma 5.3

(i) For all $\Gamma \vdash M \precsim^{\bullet} M'\colon\sigma$ and $\Gamma \vdash M' \precsim^{\circ} M''\colon\sigma$ we have $\Gamma \vdash M \precsim^{\bullet} M''\colon\sigma$.

(ii) For all $\Gamma \vdash M : \sigma$ we have $\Gamma \vdash M \precsim^{\bullet} M\colon\sigma$.

(iii) For all $\Gamma \vdash M \precsim^{\circ} M'\colon\sigma$ we have $\Gamma \vdash M \precsim^{\bullet} M'\colon\sigma$.

(iv) For all $\Delta, x{:}\,\sigma \vdash N \preccurlyeq^{\bullet} N'{:}\,\sigma'$ and $\Gamma \vdash M \preccurlyeq^{\bullet} M'{:}\,\sigma$, where Γ, Δ, we have $\Gamma, \Delta \vdash N[M/x] \preccurlyeq^{\bullet} N'[M'/x]{:}\,\sigma'$.

Lemma 5.4

(i) $* \preccurlyeq^{\bullet}_{I} P$ implies $P \Downarrow *$.

(ii) $\lambda x.M \preccurlyeq^{\bullet}_{\sigma \multimap \sigma'} P$ implies $\exists N.P \Downarrow \lambda x.N$ and $\forall Q.M[Q/x] \preccurlyeq^{\bullet}_{\sigma'} N[Q/x]$.

(iii) $P_1 \otimes P_2 \preccurlyeq^{\bullet}_{\sigma_1 \otimes \sigma_2} Q$ implies $\exists Q_1, Q_2.Q \Downarrow Q_1 \otimes Q_2$ and moreover $P_1 \preccurlyeq^{\bullet}_{\sigma_1} Q_1$ and $P_2 \preccurlyeq^{\bullet}_{\sigma_2} Q_2$.

Proof In each of (i), (ii) and (iii) one unravels the definition of $\preccurlyeq^{\bullet}_{\sigma}$ in the hypothesis of each implication, and then applies Lemma 5.3 to the resulting data. $\square$

Lemma 5.5 If $P \preccurlyeq^{\bullet}_{\sigma} Q$ and $P \Downarrow V$ then $V \preccurlyeq^{\bullet}_{\sigma} Q$.

Proof We use Rule Induction for $\Downarrow$, with subsidiary case analyses. We look at the rule

$$\frac{P \Downarrow P_1 \otimes P_2 \quad M[P_1, P_2/x, y] \Downarrow V}{\mathsf{split}\ P\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ M \Downarrow V} \Downarrow\text{SPLIT}$$

Let us suppose that $\mathsf{split}\ P\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ M \preccurlyeq^{\bullet}_{\sigma} Q$. It follows that $\exists P', M'$ such that

$$P \preccurlyeq^{\bullet}_{\sigma_1 \otimes \sigma_2} P' \tag{1}$$

$$x{:}\,\sigma_1, y{:}\,\sigma_2 \vdash M \preccurlyeq^{\bullet} M'{:}\,\sigma \tag{2}$$

$$\mathsf{split}\ P'\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ M' \preccurlyeq_{\sigma} Q \tag{3}$$

By the induction hypothesis and (1) we have $P \Downarrow P_1 \otimes P_2 \preccurlyeq^{\bullet}_{\sigma_1 \otimes \sigma_2} P'$ and hence $\exists P_1', P_2'$ such that

$$P_1 \preccurlyeq^{\bullet}_{\sigma_1} P_1' \qquad P_2 \preccurlyeq^{\bullet}_{\sigma_2} P_2' \qquad P_1' \otimes P_2' \preccurlyeq_{\sigma_1 \otimes \sigma_2} P' \tag{4}$$

From (4) we have $\exists P_1'', P_2''$ such that

$$P' \Downarrow P_1'' \otimes P_2'' \quad P_1' \preccurlyeq_{\sigma_1} P_1'' \quad P_2' \preccurlyeq_{\sigma_2} P_2'' \tag{5}$$

From (2), (4), and (5), plus Lemma 5.3 (i) and (iv) we have $M[P_1, P_2/x, y] \preccurlyeq^{\bullet}_{\sigma} M'[P_1'', P_2''/x, y]$. We must also have $M[P_1, P_2/x, y] \Downarrow V$, and so from the induction hypotheses $V \preccurlyeq^{\bullet}_{\sigma} M'[P_1'', P_2''/x, y] \quad (*)$. We now consider cases for V:

(*Case* $V \equiv Q_1 \otimes Q_2$): From $(*)$ we must have $\exists Q_1', Q_2'$ such that

$$Q_1 \preccurlyeq^{\bullet}_{\sigma_1} Q_1' \qquad Q_2 \preccurlyeq^{\bullet}_{\sigma_2} Q_2' \tag{6}$$

$$Q_1' \otimes Q_2' \preccurlyeq_{\sigma_1 \otimes \sigma_2} M'[P_1'', P_2''/x, y] \tag{7}$$

Note that given (6), to prove that $V \preccurlyeq^{\bullet}_{\sigma} Q$ all we need to show is

$$Q_1' \otimes Q_2' \preccurlyeq_{\sigma_1 \otimes \sigma_2} Q \tag{8}$$

From (7) we have

$$M'[P_1'', P_2''/x, y] \Downarrow Q_1'' \otimes Q_2'' \tag{9}$$

and this together with (5) gives $\mathsf{split}\ P'\ \mathsf{as}\ x \otimes y\ \mathsf{in}\ M' \Downarrow Q_1'' \otimes Q_2''$. From the definition of similarity, we must therefore have $Q \Downarrow Q_1''' \otimes Q_2'''$, where $Q_1'' \preccurlyeq_{\sigma_1} Q_1'''$ and $Q_2'' \preccurlyeq_{\sigma_2} Q_2'''$. From (7) and (9) we have $Q_1' \preccurlyeq_{\sigma_1} Q_1''$ and $Q_2' \preccurlyeq_{\sigma_2} Q_2''$. These facts, together with Proposition 4.1 and the definition of similarity, prove (8). □

Proposition 5.6 For all Γ, M, M' and σ, we have $\Gamma \vdash M \preccurlyeq^\circ M' : \sigma$ iff $\Gamma \vdash M \preccurlyeq^\bullet M' : \sigma$.

Proof

($\Longrightarrow$): This is Lemma 5.3 part (iii).

($\Longleftarrow$): From Lemma 5.3 parts (ii) and (iv) and the fact that any program is a closed expression, it is enough to prove that we have $P \preccurlyeq_\sigma^\bullet P' \Longrightarrow P \preccurlyeq_\sigma P'$ for all types and programs, which is precisely that $\preccurlyeq^\bullet \lhd \preccurlyeq$. To prove this, we use coinduction (which involves a case analysis on types): the details are omitted.

□

Theorem 5.7 The family $(\preceq^\circ_{\Gamma,\sigma} \mid \Gamma, \sigma)$ is a linear precongruence.

Proof

Clauses (a) and (b) are shown in Lemma 5.2. Clauses (c) and (d) follow from Proposition 4.1 that similarity is a preorder, together with the definition of open similarity. Clauses (e) and (f) hold due to the definition of $\preccurlyeq^\bullet$, plus clause (c): for (e), suppose that $\Gamma, x{:}\,\sigma \vdash M \preccurlyeq^\circ M' : \sigma'$. By (c) we have $\Gamma \vdash \lambda x.M' \preccurlyeq^\circ \lambda x.M' : \sigma \multimap \sigma'$. By Proposition 5.6 we have $\Gamma, x{:}\,\sigma \vdash M \preccurlyeq^\bullet M' : \sigma'$. From the definition of the Howe relation, and the latter two deductions, one has $\Gamma \vdash \lambda x.M \preccurlyeq^\bullet \lambda x.M' : \sigma \multimap \sigma'$. We are done by appeal to Proposition 5.6. Clause (f) is similar.

□

Lemma 5.8 For any open similarity relationship $\beta, \Delta \vdash M \preccurlyeq^\circ M' : \rho$ and linear context $\Gamma \vdash_\eta^\beta \mathcal{C}[-_\sigma] : \sigma$, we have $\Gamma, \Delta \vdash \mathcal{C}[M] \preccurlyeq^\circ \mathcal{C}[M'] : \sigma$.

Proof We use induction on the derivation of the linear contexts; here are two example induction steps:

(*Case* $\dfrac{}{[\,] \vdash_{[\,]}^{[\,]} -_\rho : \rho}$): In this case, $\beta = [\,]$. Hence $\Delta \vdash M \preccurlyeq^\circ M' : \rho$, and as $\Gamma = [\,]$ the conclusion is immediate.

(*Case* $\dfrac{\Gamma \vdash_\eta^\beta \mathcal{C} : \sigma'}{\Gamma \vdash_\eta^{\beta, x:\sigma} \lambda x.\mathcal{C} : \sigma \multimap \sigma'}\ [x \notin var(\Gamma), x \notin var(\eta)]$):

Suppose that $\beta, x{:}\,\sigma, \Delta \vdash M \preccurlyeq^\circ M'{:}\,\rho$. By the induction hypothesis we have $\Gamma, x{:}\,\sigma, \Delta \vdash \mathcal{C}[M] \preccurlyeq^\circ \mathcal{C}[M']{:}\,\sigma'$, and thus $\Gamma, \Delta \vdash \lambda x.\mathcal{C}[M] \preccurlyeq^\circ \lambda x.\mathcal{C}[M']{:}\,\sigma'$ using Theorem 5.7 and clause (e) of linear precongruence.

$\square$

Proof [*Of Theorem* 4.2] Follows from Lemma 5.8 and $\preccurlyeq_\tau = \Phi(\preccurlyeq)_\tau$. $\square$

6 Conclusions and Further Work

We have defined a notion of contextual preorder in a typed linear setting, and shown that preorder relationships can be established using coinductive techniques. This result hinges on the fact that similarity is a (linear) precongruence, and this can be shown using an adaptation of Howe's method. Many of the techniques used here are similar to those which appear in the corresponding non-linear setting, but the proofs of results are very delicate, arising from the fact that linearity forces an extremely controlled use of variables.

We plan to write a further paper, based on the work here, which gives examples of the contextual preorder established via coinduction, details of the exact relationship between the linear contextual preorder and open similarity, and which expands significantly on the complexity of the linear system under study.

Acknowledgements

Thanks to Lindsay Errington and Guy McCusker for comments on a draft of the paper.

References

[Abr90a] S. Abramsky. Computational interpretations of linear logic, 1990.

[Abr90b] S. Abramsky. The lazy lambda calculus. In David Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

[CG96] R. L. Crole and A. D. Gordon. Relating Operational and Denotational Semantics for Input/Output Effects. Technical Report 1996/5, Department of Mathematics and Computer Science, University of Leicester, 1996. Submitted to the Journal of Mathematical Structures in Computer Science.

[Cro95] R. L. Crole. Functional Programming Theory, 1995. Department of Mathematics and Computer Science Lecture Notes, LaTeX format iv+68pp with index.

[Gor94] Andrew D. Gordon. *Functional Programming and Input/Output.* Cambridge University Press, 1994. Distinguished dissertations in Computer Science.

[Gor95] A. D. Gordon. Bisimilarity as a theory of functional programming. *Electronic Notes in Theoretical Computer Science*, 1, 1995.

[How89] D. J. Howe. Equality in lazy computation systems. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science*, pages 198–203, 1989.

[How93] D. J. Howe. Proving congruence of simulation orderings in functional languages. 1993.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[NPS90] B. Nordström, K. Petersson, and J.M. Smith. *Programming in Martin-Löf's Type Theory*, volume 7 of *Monographs on Computer Science.* Oxford University Press, 1990.

[Pit94] A. M. Pitts. Some notes on inductive and coinductive techniques in the semantics of functional programs. Technical Report BRICS-NS-94-5, BRICS, Department of Computer Science, University of Aarhus, Denmark, December 1994 1994.

[Pit95] A. M. Pitts. Operationally based theories of program equivalence. Technical report, Cambridge University Computer Laboratory, 1995. Notes to accompany lectures given at the Summer School on Semantics and Logics of Computation, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK.

[San91] D. Sands. Operational theories of improvement in functional languages. In *Glasgow Functional Programming Workshop*, 1991.