J. Functional Programming 5 (1): 65–80, January 1995 © 1995 Cambridge University Press

Strong normalisation for the linear term calculus

P. N. BENTON[†]

Computer Laboratory, University of Cambridge, New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK
(e-mail: Nick.Benton@cl.cam.ac.uk*)*

Abstract

We prove a strong normalisation result for the linear term calculus of Benton, Bierman, Hyland and de Paiva. Rather than prove the result from first principles, we give a translation of linear terms into terms in the second-order polymorphic lambda calculus ($\lambda 2$) which allows the result to be proved by appealing to the well-known strong normalisation property of $\lambda 2$. An interesting feature of the translation is that it makes use of the $\lambda 2$ coding of a coinductive datatype as the translation of the !-types (exponentials) of the linear calculus.

Capsule Review

The author proves that reduction in a linear term calculus introduced elsewhere is strongly normalising. The result is obtained by interpreting the types and the terms of the linear term calculus as types and terms of the second-order λ-calculus $\lambda 2$. This proof is quite astute, and the verifications that the translation works are rather involved. This is a nice result.

1 Introduction

The Linear Term Calculus (LTC) is the term assignment system for propositional intuitionistic linear logic (ILL) which was introduced by Benton, Bierman, Hyland and de Paiva (1992a; 1992b; 1993). LTC arises via the propositions-as-types analogy from a natural deduction formulation of ILL in the same way that the simply-typed lambda calculus arises from propositional intuitionistic logic (Howard, 1980; Girard *et al.*, 1989). Just as in the familiar non-linear case, normalisation steps on the natural deduction proofs (removing the 'detours' which occur when a logical connective is introduced and then immediately eliminated) induce β-reduction steps on the associated terms.

From a computer science perspective, the linear term calculus is interesting for several reasons. Linear terms make resource usage very explicit; for example, β-reduction shows exactly how copying and discarding (garbage collection) proceed recursively on the structure of the object in question. This suggests that a linear

† Research supported by a SERC Fellowship.

functional language could be of value either as a source language, giving the programmer greater control over resource usage, or as an intermediate language, allowing a compiler to express resource-based optimisations. This is currently an active field of research (see Chirimar *et al.*, 1993, for example), though much remains to be done, particularly as regards the differences between term and graph rewriting. The linear term calculus is also an appropriate metalanguage for reasoning about some programming language features which are not particularly resource-like, such as the relationship between call-by-value and call-by-name.

This paper concerns the important proof-theoretic question, which was left open in the above-cited work of Benton *et al.*, of whether or not the normalisation process on derivations in the natural deduction formulation of the logic (or, equivalently, the induced β-reduction of terms) always terminates. Here we answer that question in the affirmative, showing that that there are no infinite β-reduction sequences from well-typed linear terms.†

The literature contains many strong normalisation proofs for various systems, many of which use variants of Tait's (1967) reducibility argument. Strong normalisation for the linear term calculus can be proved this way (see Bierman, 1993), but here we take a somewhat different approach. If we wish to prove strong normalisation for a language L_1, and we already know strong normalisation holds for a language L_2 (by a reducibility argument, for example), then it suffices to exhibit a translation $t \mapsto t^\circ$ from L_1 to L_2 with the property that if $s \rightarrow_1 t$ in L_1 then $s^\circ \rightarrow_2^+ t^\circ$ in L_2 (where $\rightarrow_1, \rightarrow_2$ are the one-step reduction relations in L_1 and L_2, respectively, and $\rightarrow_2^+$ is transitive closure of $\rightarrow_2$). For, given such a translation, if there were an infinite reduction sequence:

$$t_0 \rightarrow_1 t_1 \rightarrow_1 t_2 \rightarrow_1 \cdots$$

in L_1, it would induce an infinite sequence:

$$t_0^\circ \rightarrow_2^+ t_1^\circ \rightarrow_2^+ t_2^\circ \rightarrow_2^+ \cdots$$

of reductions in L_2, contradicting strong normalisation for that language. This is the technique which we shall use here, with L_1 the linear term calculus and L_2 the Girard/Reynolds second-order polymorphic lambda calculus (Girard, 1972; Reynolds, 1974).

1.1 The linear term calculus

The types of LTC are given by the following grammar, where G ranges over some given collection of base types:

$$A ::= G \mid I \mid A \otimes A \mid A \& A \mid A \oplus A \mid A \multimap A \mid !A$$

The term-formation rules of LTC are recalled in Fig. 1, and the associated β-reductions are shown in Fig. 2, where, for example:

$$\textsf{discard}\ e_i\ \textsf{in}\ u$$

† LTC β-reduction is also confluent (Bierman, 1993).

is an abbreviation for

$$\mathsf{discard}\ e_1\ \mathsf{in}\ \mathsf{discard}\ e_2\ \mathsf{in}\ \ldots \mathsf{discard}\ e_n\ \mathsf{in}\ u$$

The one-step reduction relation $\rightarrow$ is defined by augmenting the β axioms with inference rules making $\rightarrow$ into a congruence, which can be summarised by the rule:

$$\frac{e \rightarrow e'}{C[e] \rightarrow C[e']}$$

The version of LTC which we are using here is an extension of that considered in Benton *et al.* (1992a), in that we have included the additives & and $\oplus$. The additive units have been omitted as no β-reductions are associated with either of them. This conveniently avoids the need to discuss some rather unpleasant syntax. Even for the binary versions of the additive connectives, however, there is some choice as to how the natural deduction proof rules and the associated syntax should be formulated. We have picked the simplest and most obvious version, but there are reasons why one might favour a formulation in which, for example, the introduction rule for & looks like:

$$\frac{\Delta_1 \vdash e_1 : A_1 \cdots \Delta_n \vdash e_n : A_n \qquad \vec{x_i} : \vec{A_i} \vdash f : B \qquad \vec{y_i} : \vec{A_i} \vdash g : C}{\Delta_1, \ldots, \Delta_n \vdash \mathsf{use}\ e_1, \ldots, e_n\ \mathsf{as}\ x_1, \ldots x_n\ \mathsf{in}\ f\ \mathsf{or}\ y_1, \ldots, y_n\ \mathsf{in}\ g\ : B\&C}$$

Further discussion can be found in Bierman (1993), but for now we merely remark that the choice makes no real difference to the strong normalisation proof.

It should be noted that in Benton *et al.* (1992a) we also considered a secondary form of reduction rule: the so-called *commuting conversions*. We shall not consider such conversions here, and at present we do not have a proof that $\rightarrow_{\beta,c}$ is strongly normalising for LTC.

1.2 The second-order polymorphic lambda calculus

The types of the second-order lambda calculus (also known as System F or $\lambda 2$) are given by the following grammar, where G ranges over a given set of base types and X over a set of type variables:

$$A ::= G \mid X \mid A \times \cdots \times A \mid A + A \mid A \rightarrow A \mid \forall X.A$$

Note that, for notational convenience in what follows, we have presented the system with n-ary product and binary coproduct types. These are not strictly necessary, as they can both be coded within the $\forall \rightarrow$ fragment in the usual way (Girard *et al.*, 1989). The term-formation rules for $\lambda 2$ are recalled in Fig. 3. We will frequently omit type annotations in terms when they are clear from context. We shall need the following trivial fact about typing derivations in $\lambda 2$:

Lemma 1

If $\Gamma \vdash e : A$ and $x \notin \Gamma$ then $\Gamma, x : B \vdash e : A$. $\square$

$$x{:}A \vdash x{:}A \; (\mathcal{A}x)$$

$$\frac{\Gamma, x{:}A \vdash e{:}B}{\Gamma \vdash (\lambda x{:}A.e){:}A\multimap B}\;(\multimap_{\mathcal{I}}) \qquad \frac{\Gamma \vdash e{:}A\multimap B \qquad \Delta \vdash f{:}A}{\Gamma,\Delta \vdash ef{:}B}\;(\multimap_{\mathcal{E}})$$

$$\vdash * : I\;(I_{\mathcal{I}}) \qquad \frac{\Gamma \vdash e : A \qquad \Delta \vdash f : I}{\Gamma,\Delta \vdash \mathsf{let}\, f\, \mathsf{be}\, *\, \mathsf{in}\, e : A}\;(I_{\mathcal{E}})$$

$$\frac{\Gamma \vdash e{:}A \qquad \Delta \vdash f{:}B}{\Gamma,\Delta \vdash e\otimes f{:}A\otimes B}\;(\otimes_{\mathcal{I}}) \qquad \frac{\Gamma \vdash e{:}A\otimes B \qquad \Delta, x{:}A, y{:}B \vdash f{:}C}{\Gamma,\Delta \vdash \mathsf{let}\, e\, \mathsf{be}\, x\otimes y\, \mathsf{in}\, f{:}C}\;(\otimes_{\mathcal{E}})$$

$$\frac{\Gamma \vdash e{:}A \qquad \Gamma \vdash f{:}B}{\Gamma \vdash (e,f){:}A\&B}\;(\&_{\mathcal{I}})$$

$$\frac{\Gamma \vdash e{:}A\&B}{\Gamma \vdash \mathsf{fst}\, e{:}A}\;(\&_{\mathcal{E}-1}) \qquad \frac{\Gamma \vdash e{:}A\&B}{\Gamma \vdash \mathsf{snd}\, e{:}B}\;(\&_{\mathcal{E}-2})$$

$$\frac{\Gamma \vdash e{:}A}{\Gamma \vdash \mathsf{inl}_B(e){:}A\oplus B}\;(\oplus_{\mathcal{I}-1}) \qquad \frac{\Gamma \vdash e{:}B}{\Gamma \vdash \mathsf{inr}_A(e){:}A\oplus B}\;(\oplus_{\mathcal{I}-2})$$

$$\frac{\Gamma \vdash e{:}A\oplus B \qquad \Delta, x{:}A \vdash f{:}C \qquad \Delta, y{:}B \vdash g{:}C}{\Gamma,\Delta \vdash \mathsf{case}\, e\, \mathsf{of}\, \mathsf{inl}(x) \Rightarrow f \mid \mathsf{inr}(y) \Rightarrow g{:}C}\;(\oplus_{\mathcal{E}})$$

$$\frac{\Delta_1 \vdash e_1 : !A_1 \quad \cdots \quad \Delta_n \vdash e_n : !A_n \qquad x_1 : !A_1, \ldots, x_n : !A_n \vdash f{:}B}{\Delta_1, \ldots, \Delta_n \vdash \mathsf{promote}\, e_1, \ldots, e_n\, \mathsf{for}\, x_1, \ldots, x_n\, \mathsf{in}\, f : !B}\;\textit{Promotion}$$

$$\frac{\Gamma \vdash e : !A \qquad \Delta, x : !A, y : !A \vdash f{:}B}{\Gamma,\Delta \vdash \mathsf{copy}\, e\, \mathsf{as}\, x, y\, \mathsf{in}\, f{:}B}\;\textit{Contraction}$$

$$\frac{\Gamma \vdash e : !A \qquad \Delta \vdash f{:}B}{\Gamma,\Delta \vdash \mathsf{discard}\, e\, \mathsf{in}\, f{:}B}\;\textit{Weakening} \qquad \frac{\Gamma \vdash e : !A}{\Gamma \vdash \mathsf{derelict}(e){:}A}\;\textit{Dereliction}$$

Fig. 1. The Linear Term Calculus (LTC).

The reduction rules for $\lambda 2$ consist of the following β axioms:

$$\begin{array}{lcl}
(\lambda x{:}A.e)\, f & \rightarrow & e[f/x] \\
\pi_i\langle e_1, \ldots, e_n\rangle & \rightarrow & e_i \\
\mathit{case}\; \mathit{inl}_B(e)\; \mathit{of}\; \mathit{inl}(x) \Rightarrow f \mid \mathit{inr}(y) \Rightarrow g & \rightarrow & f[e/x] \\
\mathit{case}\; \mathit{inr}_A(e)\; \mathit{of}\; \mathit{inl}(x) \Rightarrow f \mid \mathit{inr}(y) \Rightarrow g & \rightarrow & g[e/y]
\end{array}$$

together with the following axiom for reduction on types:

$$(\Lambda X.e)\, A \;\rightarrow\; e[A/X]$$

and a collection of inference rules extending $\rightarrow$ to a congruence. The important fact about reduction of terms in $\lambda 2$ is that it always terminates:

$(\lambda x{:}A.t)\,e$	$\longrightarrow$	$t[e/x]$
let $*$ be $*$ in e	$\longrightarrow$	e
let $e \otimes t$ be $x \otimes y$ in u	$\longrightarrow$	$u[e/x, t/y]$
fst (e, f)	$\longrightarrow$	e
snd (e, f)	$\longrightarrow$	f
case $\mathsf{inl}_B(e)$ of $\mathsf{inl}(x) \Rightarrow f \mid \mathsf{inr}(y) \Rightarrow g$	$\longrightarrow$	$f[e/x]$
case $\mathsf{inr}_A(e)$ of $\mathsf{inl}(x) \Rightarrow f \mid \mathsf{inr}(y) \Rightarrow g$	$\longrightarrow$	$g[e/y]$
derelict(promote e_i for x_i in t)	$\longrightarrow$	$t[e_i/x_i]$
discard (promote e_i for x_i in t) in u	$\longrightarrow$	discard e_i in u
copy (promote e_i for x_i in t) as y, z in u	$\longrightarrow$	copy e_i as x_i', x_i'' in u[promote x_i' for x_i in t/y, promote x_i'' for x_i in t/z]

Fig. 2. β-reductions for the linear term calculus.

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x : A.e) : A \rightarrow B} \qquad \frac{\Gamma \vdash e : A \rightarrow B \qquad \Gamma \vdash f : A}{\Gamma \vdash e f : B}$$

$$\frac{\Gamma \vdash e_1 : A_1 \ \cdots \ \Gamma \vdash e_n : A_n}{\Gamma \vdash \langle e_1, \ldots, e_n \rangle : A_1 \times \cdots \times A_n} \qquad \frac{\Gamma \vdash e : A_1 \times \cdots \times A_n}{\Gamma \vdash \pi_i e : A_i}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathit{inl}_B(e) : A + B} \qquad \frac{\Gamma \vdash e : B}{\Gamma \vdash \mathit{inr}_A(e) : A + B}$$

$$\frac{\Gamma \vdash e : A + B \qquad \Gamma, x : A \vdash f : C \qquad \Gamma, y : B \vdash g : C}{\Gamma \vdash \mathit{case}\, e \ \mathit{of}\ \mathit{inl}(x) \Rightarrow f \mid \mathit{inr}(y) \Rightarrow g : C}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \Lambda X.e : \forall X.A}\ X \notin FTV(\Gamma) \qquad \frac{\Gamma \vdash e : \forall X.A}{\Gamma \vdash e B : A[B/X]}$$

Fig. 3. The second-order polymorphic lambda calculus ($\lambda 2$).

Theorem 2
All well-typed terms of $\lambda 2$ are strongly normalising.

Proof
The original proof is in Girard (1972). A good discussion may be found in Gallier (1990). $\square$

2 Coding coinductive datatypes in $\lambda 2$

The way in which inductive datatypes such as finite lists, trees and natural numbers may be coded in $\lambda 2$ is well-known. The dual codings of coinductive types are perhaps less familiar, so we give a brief account here.

Let $\Phi(X)$ be a $\lambda 2$ type in which the free type variable X appears positively. Then the greatest fixed point of Φ is given by:

$$\nu_\Phi = \exists X.(X \to \Phi(X)) \times X$$

where:

$$\exists X.A \stackrel{\text{def}}{=} \forall Y.(\forall X.A \to Y) \to Y$$

Expanding this out and currying gives:

$$\nu_\Phi \stackrel{\text{def}}{=} \forall Y.(\forall X.(X \to \Phi(X)) \to X \to Y) \to Y$$

Note that $\Phi(X)$ is functorial in X as X occurs positively. This means that given $f : A \to B$ we can define a term $\Phi[f] : \Phi(A) \to \Phi(B)$ by induction on the structure of Φ.

Terms of type ν_Φ are built using:

$$\begin{aligned} \mathit{build}_\Phi &: \forall X.(X \to \Phi(X)) \to X \to \nu_\Phi \\ &\stackrel{\text{def}}{=} \Lambda X.\lambda f.\lambda x.\Lambda C.\lambda h.(h\, X\, f\, x) \end{aligned}$$

and associated destructor is:

$$\begin{aligned} \mathit{out}_\Phi &: \nu_\Phi \to \Phi(\nu_\Phi) \\ &\stackrel{\text{def}}{=} \lambda m.m\, \Phi(\nu_\Phi)\ (\Lambda X.\lambda f.\lambda x.(\Phi[\mathit{build}_\Phi\, X\, f]\, (f\, x))) \end{aligned}$$

The relationship between *build* and *out* is given by the following easily verified reduction:

$$\mathit{out}_\Phi\, (\mathit{build}_\Phi\, X\, f\, x) \ \to^+\ \Phi[\mathit{build}_\Phi\, X\, f]\, (f\, x)$$

The categorically-motivated reader will note that the equality implied by the previous reduction is expressed by the commutativity of:

$$\begin{tikzcd} \Phi(X) \arrow[r, "{\Phi[\mathit{build}_\Phi\, X\, f]}"] & \Phi(\nu_\Phi) \\ X \arrow[u, "f"] \arrow[r, "\mathit{build}_\Phi\, X\, f"'] & \nu_\Phi \arrow[u, "\mathit{out}_\Phi"'] \end{tikzcd}$$

which characterises (ν_Φ, out_Φ) as a weakly terminal Φ-coalgebra. It is strongly terminal in models satisfying appropriate parametricity conditions—see Plotkin and Abadi (1993), for example.

3 The translation

3.1 The translation of types

Given a LTC type A, the $\lambda 2$ type A° is defined by induction on the structure of A as follows:

$$\begin{aligned}
G^\circ &\stackrel{\text{def}}{=} G \\
(A \multimap B)^\circ &\stackrel{\text{def}}{=} A^\circ \to B^\circ \\
(A \otimes B)^\circ &\stackrel{\text{def}}{=} \forall X.(A^\circ \to B^\circ \to X) \to X \\
I^\circ &\stackrel{\text{def}}{=} \forall X.X \to X \\
(A \& B)^\circ &\stackrel{\text{def}}{=} A^\circ \times B^\circ \\
(A \oplus B)^\circ &\stackrel{\text{def}}{=} A^\circ + B^\circ \\
(!A)^\circ &\stackrel{\text{def}}{=} \nu_{\Phi^{A^\circ}}
\end{aligned}$$

where:

$$\Phi^B(X) \stackrel{\text{def}}{=} (\forall Z.Z \to Z) \times B \times (\forall Z.(X \to X \to Z) \to Z)$$

Note that the translations of $\otimes$ and I use directly the $\lambda 2$ encodings of binary products and the unit type, as these turn out to be technically more convenient. Similarly, $\Phi^B(X)$ can be thought of as:

$$1 \times B \times (X \times X)$$

and thus ν_{Φ^B} is essentially the type of infinite binary trees with nodes labelled by elements of B. This translation of exponentials is motivated by, apart from the fact that it works, the linear logic treatment of $!A$ as satisfying:

$$!A \cong I \& A \& (!A \otimes !A)$$

and also by more operational concerns. The types $!A_i$ of the promoted terms e_i in promote e_i for x_i in f are not apparent in the type $!B$ of the whole term, but when the promoted term is broken apart by a reduction, they become revealed. Thus the translation of $!B$ needs to be an abstract datatype which 'hides' the (translations of the) types $!A_i$, which can be achieved by the use of an existential type (Mitchell and Plotkin, 1988). See Section 5 for further discussion of the intuition behind the translation of !.

If $f : A \to B$ is a $\lambda 2$ term, then $\Phi^C[f] : \Phi^C(A) \to \Phi^C(B)$ is given by:

$$\begin{aligned}
\Phi^C[f] \;=\; & \lambda w : \Phi^C(A). \langle \pi_1 w, \pi_2 w, \\
& \qquad \Lambda Z.\lambda h : B \to B \to Z.(\pi_3 w) Z\, (\lambda x : A.\lambda y : A.h\,(f\,x)\,(f\,y)) \rangle
\end{aligned}$$

3.2 The translation of terms

From a derivation Π of the judgement $\Gamma \vdash e : A$ in the linear term calculus, we now define a derivation Π° in $\lambda 2$ which proves $\Gamma^\circ \vdash e^\circ : A^\circ$ by induction on Π. Of course, since terms code derivations uniquely (in both systems), the translation is really from terms to terms. It is, however, easier to present it using derivations as that makes the partitioning of linear contexts more explicit. For the rules which do not explicitly involve exponentials, the translation is straightforward:

- If Π is

$$x : A \vdash x : A$$

  then Π° is

$$x : A^\circ \vdash x : A^\circ$$

- If Π ends in

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x : A.e) : A \multimap B}$$

  then by induction there is a derivation of $\Gamma^\circ, x : A^\circ \vdash e^\circ : B^\circ$ so we can form

$$\frac{\Gamma^\circ, x : A^\circ \vdash e^\circ : B^\circ}{\Gamma^\circ \vdash (\lambda x : A^\circ.e^\circ) : A^\circ \to B^\circ}$$

- If Π ends in

$$\frac{\Gamma \vdash e : A \multimap B \qquad \Delta \vdash f : A}{\Gamma, \Delta \vdash (e f) : B}$$

  then by induction there are derivations of

$$\Gamma^\circ \vdash e^\circ : A^\circ \to B^\circ$$

  and

$$\Delta^\circ \vdash f^\circ : A^\circ$$

  By Lemma 1, this means that there are derivations of

$$\Gamma^\circ, \Delta^\circ \vdash e^\circ : A^\circ \to B^\circ$$

  and

$$\Gamma^\circ, \Delta^\circ \vdash f^\circ : A^\circ$$

  so that we can form

$$\frac{(\Gamma, \Delta)^\circ \vdash e^\circ : A^\circ \to B^\circ \qquad (\Gamma, \Delta)^\circ \vdash f^\circ : A^\circ}{(\Gamma, \Delta)^\circ \vdash (e^\circ f^\circ) : B^\circ}$$

- If Π is

$$\vdash * : I$$

  then Π° is

$$\frac{\dfrac{x : X \vdash x : X}{\vdash (\lambda x : X.x) : X \to X}}{\vdash (\Lambda X.\lambda x : X.x) : \forall X.X \to X}$$

- If Π ends in
$$\frac{\Delta \vdash f : I \qquad \Gamma \vdash e : A}{\Gamma, \Delta \vdash \mathsf{let}\, f\, \mathsf{be}\, *\, \mathsf{in}\, e : A}$$
then by induction and Lemma 1 we can form
$$\frac{\dfrac{(\Gamma,\Delta)^\circ \vdash f^\circ : \forall X.X \to X}{(\Gamma,\Delta)^\circ \vdash (f^\circ A^\circ) : A^\circ \to A^\circ} \qquad (\Gamma,\Delta)^\circ \vdash e^\circ : A^\circ}{(\Gamma,\Delta)^\circ \vdash (f^\circ A^\circ)\, e^\circ : A^\circ}$$
- Similarly, if Π ends in
$$\frac{\Gamma \vdash e : A \qquad \Delta \vdash f : B}{\Gamma, \Delta \vdash e \otimes f : A \otimes B}$$
then Π° is the obvious derivation of
$$(\Gamma,\Delta)^\circ \vdash \Lambda X.\lambda h : A^\circ \to B^\circ \to X.(h\, e^\circ)\, f^\circ : \forall X.(A^\circ \to B^\circ \to X) \to X$$
- If Π ends in
$$\frac{\Gamma \vdash e : A \otimes B \qquad \Delta, x : A, y : B \vdash f : C}{\Gamma, \Delta \vdash \mathsf{let}\, e\, \mathsf{be}\, x \otimes y\, \mathsf{in}\, f : C}$$
then Π° is the derivation of
$$(\Gamma,\Delta)^\circ \vdash (e^\circ C^\circ)(\lambda x : A^\circ.\lambda y : B^\circ.f^\circ) : C^\circ$$

The additives are particularly easy to deal with, so we just give the term-to-term translations:

- $(e,f)^\circ \overset{\text{def}}{=} \langle e^\circ, f^\circ \rangle$.
- $(\mathsf{fst}\, e)^\circ \overset{\text{def}}{=} \pi_1(e^\circ)$ and similarly for $\mathsf{snd}\, e$.
- $(\mathsf{inl}_B(e))^\circ \overset{\text{def}}{=} \mathit{inl}_{B^\circ}(e^\circ)$ and similarly for $\mathsf{inr}_A(e)$.
- $(\mathsf{case}\, e\, \mathsf{of}\, \mathsf{inl}(x) \Rightarrow f \mid \mathsf{inr}(y) \Rightarrow g)^\circ \overset{\text{def}}{=} \mathit{case}\, e^\circ\, \mathit{of}\, \mathit{inl}(x) \Rightarrow f^\circ \mid \mathit{inr}(y) \Rightarrow g^\circ$.

The case of dereliction is dealt with as follows:

- If Π ends with
$$\frac{\Gamma \vdash e : !A}{\Gamma \vdash \mathsf{derelict}(e) : A}$$
then by induction there is a derivation of $\Gamma^\circ \vdash e^\circ : \nu_{\Phi^{A^\circ}}$ so that we can form Π° as follows:
$$\frac{\dfrac{\Gamma^\circ \vdash \mathit{out}_{\Phi^{A^\circ}} : \nu_{\Phi^{A^\circ}} \to \Phi^{A^\circ}(\nu_{\Phi^{A^\circ}}) \qquad \Gamma^\circ \vdash e^\circ : \nu_{\Phi^{A^\circ}}}{\Gamma^\circ \vdash (\mathit{out}_{\Phi^{A^\circ}}\, e^\circ) : (\forall Z.Z \to Z) \times A^\circ \times (\forall Z.(\nu_{\Phi^{A^\circ}} \to \nu_{\Phi^{A^\circ}} \to Z) \to Z)}}{\Gamma^\circ \vdash \pi_2(\mathit{out}_{\Phi^{A^\circ}}\, e^\circ) : A^\circ}$$

To simplify the presentation of the translations of the remaining three rules for the exponentials, it is helpful to define the following abbreviations for $\lambda 2$ terms:

$$\begin{aligned} \mathit{discard}^{B,C}\; e\; \mathit{in}\; f &\overset{\text{def}}{=} (\pi_1(\mathit{out}_{\Phi^B}\, e))\, C\, f \\ \mathit{copy}^{B,C}\; e\; \mathit{as}\; x,y\; \mathit{in}\; f &\overset{\text{def}}{=} (\pi_3(\mathit{out}_{\Phi^B}\, e))\, C\, (\lambda x : \nu_{\Phi^B}.\lambda y : \nu_{\Phi^B}.f) \end{aligned}$$

- If Π ends in

$$\frac{\Gamma \vdash e :!A \qquad \Delta \vdash f : B}{\Gamma, \Delta \vdash \mathsf{discard}\, e\, \mathsf{in}\, f : B}$$

then by induction and Lemma 1 we have derivations of

$$(\Gamma, \Delta)^\circ \vdash e^\circ : \nu_{\Phi A^\circ}$$

$$(\Gamma, \Delta)^\circ \vdash f^\circ : B^\circ$$

from which it is easy to check that we can form Π° proving

$$(\Gamma, \Delta)^\circ \vdash \mathit{discard}^{A^\circ, B^\circ}\, e^\circ\, \mathit{in}\, f^\circ : B^\circ$$

- If Π ends with

$$\frac{\Gamma \vdash e :!A \qquad \Delta, x :!A, y :!A \vdash f : B}{\Gamma, \Delta \vdash \mathsf{copy}\, e\, \mathsf{as}\, x, y\, \mathsf{in}\, f : B}$$

then there are derivations of

$$\Gamma^\circ \vdash e^\circ : \nu_{\Phi A^\circ}$$

$$\Delta^\circ, x : \nu_{\Phi A^\circ}, y : \nu_{\Phi A^\circ} \vdash f^\circ : B^\circ$$

from which we can form Π° proving

$$(\Gamma, \Delta)^\circ \vdash \mathit{copy}^{A^\circ, B^\circ}\, e^\circ\, \mathit{as}\, x, y\, \mathit{in}\, f^\circ : B^\circ$$

- If Π ends in

$$\frac{\Delta_1 \vdash e_1 :!A_1 \quad \cdots \quad \Delta_n \vdash e_n :!A_n \qquad x_1 :!A_1, \ldots, x_n :!A_n \vdash f : B}{\Delta_1, \ldots, \Delta_n \vdash \mathsf{promote}\, e_1, \ldots, e_n\, \mathsf{for}\, x_1, \ldots, x_n\, \mathsf{in}\, f :!B}$$

then we form Π° proving

$$(\Delta_1, \ldots, \Delta_n)^\circ \vdash \mathit{build}_{\Phi B^\circ} \prod_{i=1}^{n} (!A_i)^\circ\, h\, x : (!B)^\circ$$

where

$$\begin{aligned} x &= \langle e_1^\circ, \ldots, e_n^\circ \rangle \\ h &= \lambda p : \textstyle\prod (!A_i)^\circ . \langle a, b, c \rangle \\ a &= \Lambda Z . \lambda z : Z . \mathit{discard}^{A_1^\circ, Z}\, \pi_1 p\, \mathit{in} \ldots \mathit{discard}^{A_n^\circ, Z}\, \pi_n p\, \mathit{in}\, z \\ b &= (\lambda x_1 : (!A_1)^\circ . \ldots \lambda x_n : (!A_n)^\circ . f^\circ)(\pi_1 p) \ldots (\pi_n p) \end{aligned}$$

and

$$\begin{aligned} c = {} & \Lambda Z . \lambda g : \textstyle\prod (!A_i)^\circ \to \prod (!A_i)^\circ \to Z . \mathit{copy}^{A_1^\circ, Z}\, \pi_1 p\, \mathit{as}\, x_1', x_1''\, \mathit{in} \ldots \\ & \ldots \mathit{copy}^{A_n^\circ, Z}\, \pi_n p\, \mathit{as}\, x_n', x_n''\, \mathit{in}\, (g\, \langle x_1', \ldots, x_n' \rangle\, \langle x_1'', \ldots, x_n'' \rangle) \end{aligned}$$

Note that the translation of terms is compositional (i.e. a congruence):

Lemma 3
For all appropriately typeable terms t and s of LTC, $(t[s/x])^\circ = t^\circ[s^\circ/x]$. □

4 Reduction

Having given the translation, we now show that it behaves well with respect to reduction:

Theorem 4
If $\Gamma \vdash e_1 : A$ in the linear term calculus and $e_1 \rightarrow e_2$ then $e_1^\circ \rightarrow^+ e_2^\circ$ in $\lambda 2$.

Proof
By induction on the derivation of $e_1 \rightarrow e_2$. We omit the verification of the cases of the congruence rules, as these all follow trivially from the compositional nature of the translation. For the β axioms, we give a few cases:

- In the case of a $\otimes$ introduction/elimination pair we have a derivation of

$$\Gamma_1, \Gamma_2, \Delta \vdash \mathsf{let}\, e{\otimes}f\, \mathsf{be}\, x{\otimes}y\, \mathsf{in}\, u : C$$

  where

$$\Gamma_1 \vdash e : A$$

$$\Gamma_2 \vdash f : B$$

$$\Delta, x : A, y : B \vdash u : C$$

  The translation of this redex is

$$(\Lambda X.\lambda h : A^\circ \rightarrow B^\circ \rightarrow X.h\, e^\circ\, f^\circ)\, C^\circ\, (\lambda x : A.\lambda y : B.u^\circ)$$

  which reduces in four steps to

$$(u^\circ[e^\circ/x])[f^\circ/y]$$

  which is

$$u^\circ[e^\circ/x, f^\circ/y]$$

  as the free variables of u, e and f are all distinct and this is inherited by their translations. As $(-)^\circ$ is a congruence, this is in turn equal to

$$(u[e/x, f/y])^\circ$$

  as required.
- In the case of promotion followed by dereliction we have

$$\Delta_1, \ldots, \Delta_n \vdash \mathsf{derelict}(\mathsf{promote}\, e_i\, \mathsf{for}\, x_i\, \mathsf{in}\, f) : B$$

  where

$$\Delta_i \vdash e_i : !A_i$$

  and

$$x_1 : !A_1, \ldots, x_n : !A_n \vdash f : B$$

  The translation of this redex is

$$\pi_2(out_{\Phi B^\circ}\, (build_{\Phi B^\circ} \prod (!A_i)^\circ\, h\, x))$$

with h and x as before. This reduces to

$$\pi_2(\Phi^{B^\circ}[build_{\Phi B^\circ} \prod(!A_i)^\circ\, h]\,(h\,x))$$

which is

$$\pi_2((\lambda w{:}\,\Phi^{B^\circ}(\prod(!A_i)^\circ).\langle\pi_1 w, \pi_2 w, \ldots\rangle)\,(h\,x))$$

and this reduces to

$$\pi_2(h\,x)$$

Expanding h and x, this is

$$\pi_2((\lambda p{:}\prod(!A_i)^\circ.\langle a,b,c\rangle)\,\langle e_1^\circ,\ldots,e_n^\circ\rangle)$$

which reduces to

$$b[\langle e_1^\circ,\ldots,e_n^\circ\rangle/p]$$

The expansion of this last term then reduces to

$$f^\circ[e_1^\circ/x_1]\cdots[e_n^\circ/x_n]$$

which is

$$(f[e_i/x_i])^\circ$$

as required.

- In the case of promotion followed by weakening we have

$$\Delta_1,\ldots,\Delta_n,\Gamma \vdash \mathsf{discard}\,(\mathsf{promote}\;e_i\;\mathsf{for}\;x_i\;\mathsf{in}\;f)\;\mathsf{in}\;u{:}\,C$$

where

$$\Gamma \vdash u{:}\,C$$

$$\Delta_i \vdash e_i{:}\,!A_i$$

$$x_1{:}\,!A_1,\ldots,x_n{:}\,!A_n \vdash f\,:B$$

This LTC redex translates to

$$\pi_1(out_{\Phi B^\circ}\,(build_{\Phi B^\circ}\prod(!A_i)^\circ\,h\,x))\,C^\circ\,u^\circ$$

which reduces to

$$\pi_1(\Phi^{B^\circ}[build_{\Phi B^\circ}\prod(!A_i)^\circ\,h]\,(h\,x))\,C^\circ\,u^\circ$$

which is

$$\pi_1((\lambda w{:}\,\Phi^{B^\circ}(\prod(!A_i)^\circ).\langle\pi_1 w,\pi_2 w,\ldots\rangle)\,(h\,x))\,C^\circ\,u^\circ$$

and this reduces to

$$\pi_1(h\,x)\,C^\circ\,u^\circ$$

Expanding h and x, this is

$$\pi_1((\lambda p{:}\prod(!A_i)^\circ.\langle a,b,c\rangle)\,\langle e_1^\circ,\ldots,e_n^\circ\rangle)\,C^\circ\,u^\circ$$

which reduces to

$$a[\langle e_1^\circ,\ldots,e_n^\circ\rangle/p]\,C^\circ\,u^\circ$$

Expanding out the definition of a and reducing, this gives

$$discard^{A_1^\circ, C^\circ}\ e_1^\circ\ in \ldots discard^{A_n^\circ, C^\circ}\ e_n^\circ\ in\ u^\circ$$

which is

$$(\mathsf{discard}\ e_1\ \mathsf{in}\ \ldots \mathsf{discard}\ e_n\ \mathsf{in}\ u)^\circ$$

as required.

- In the case of promotion followed by contraction we have

$$\Delta_1, \ldots, \Delta_n, \Gamma \vdash \mathsf{copy}\ (\mathsf{promote}\ e_i\ \mathsf{for}\ x_i\ \mathsf{in}\ f)\ \mathsf{as}\ y, z\ \mathsf{in}\ u : C$$

where

$$\Delta_i \vdash e_i : !A_i$$

$$x_1 : !A_1, \ldots, x_n : !A_n \vdash f : B$$

$$\Gamma, y : !B, z : !B \vdash u : C$$

The translation of this redex is

$$\pi_3(out_{\Phi B^\circ}\ (build_{\Phi B^\circ} \prod(!A_i)^\circ\ h\, x))\ C^\circ\ (\lambda y : \nu_{\Phi B^\circ}.\lambda z : \nu_{\Phi B^\circ}.u^\circ)$$

which reduces to

$$\pi_3(\Phi^{B^\circ}[build_{\Phi B^\circ} \prod(!A_i)^\circ\ h]\ (h\, x))\ C^\circ\ (\lambda y : \nu_{\Phi B^\circ}.\lambda z : \nu_{\Phi B^\circ}.u^\circ)$$

which, after expanding out the definition of $\Phi^{B^\circ}[\cdot]$, reduces to

$$\begin{aligned}&(\Lambda Z.\lambda j : \nu_{\Phi B^\circ} \to \nu_{\Phi B^\circ} \to Z.\pi_3(h\, x)\, Z\ (\lambda m : \textstyle\prod(!A_i)^\circ.\lambda n : \prod(!A_i)^\circ\\ &\quad .j\ (build_{\Phi B^\circ} \textstyle\prod(!A_i)^\circ\ h\, m)\ (build_{\Phi B^\circ} \prod(!A_i)^\circ\ h\, n))\\ &)\ C^\circ\ (\lambda y : \nu_{\Phi B^\circ}.\lambda z : \nu_{\Phi B^\circ}.u^\circ)\end{aligned}$$

This expression then reduces to

$$\begin{aligned}&\pi_3(h\, x)\ C^\circ\ (\lambda m : \textstyle\prod(!A_i)^\circ.\lambda n : \prod(!A_i)^\circ\\ &\quad .u^\circ[(build_{\Phi B^\circ} \textstyle\prod(!A_i)^\circ\ h\, m)/y, (build_{\Phi B^\circ} \prod(!A_i)^\circ\ h\, n)/z])\end{aligned}$$

and then to

$$\begin{aligned}&c[\langle e_1^\circ, \ldots, e_n^\circ\rangle/p]\ C^\circ\ (\lambda m : \textstyle\prod(!A_i)^\circ.\lambda n : \prod(!A_i)^\circ\\ &\quad .u^\circ[(build_{\Phi B^\circ} \textstyle\prod(!A_i)^\circ\ h\, m)/y, (build_{\Phi B^\circ} \prod(!A_i)^\circ\ h\, n)/z])\end{aligned}$$

After expanding the definition of c, this leads to

$$\begin{aligned}&copy^{A_1^\circ, C^\circ}\ e_1^\circ\ as\ x_1', x_1''\ in \ldots copy^{A_n^\circ, C^\circ}\ e_n^\circ\ as\ x_n', x_n''\ in\\ &\quad u^\circ[(build_{\Phi B^\circ} \textstyle\prod(!A_i)^\circ\ h\ \langle x_1', \ldots, x_n'\rangle)/y,\\ &\qquad (build_{\Phi B^\circ} \textstyle\prod(!A_i)^\circ\ h\ \langle x_1'', \ldots, x_n''\rangle)/z]\end{aligned}$$

which is

$$\begin{aligned}&(\mathsf{copy}\ e_1\ \mathsf{as}\ x_1', x_1''\ \mathsf{in}\ \ldots \mathsf{copy}\ e_n\ \mathsf{as}\ x_n', x_n''\ \mathsf{in}\\ &\quad u[(\mathsf{promote}\ x_i'\ \mathsf{for}\ x_i\ \mathsf{in}\ f)/y, (\mathsf{promote}\ x_i''\ \mathsf{for}\ x_i\ \mathsf{in}\ f)/z])^\circ\end{aligned}$$

as required.

□

Corollary 5
All well-typed terms of the linear term calculus are strongly normalising. □

5 Conclusions

We have presented a strong normalisation proof for the linear term calculus which uses a translation into the second-order polymorphic lambda calculus.

The proof technique used here appears to be very general, and can probably be applied to many other systems. It is therefore worth trying to comment on the motivation and intuition behind the construction. There is a strong sense that one is constructing a categorical model of LTC within one of $\lambda 2$. This, however, is not in itself enough to lead to the translation, as we need to model reduction rather than equality. For example, every cartesian closed category is symmetric monoidal closed so one might hope to be able to prove the result by translating LTC into the simply typed lambda calculus, with tensor mapping to product, linear function space to function space, and ! interpreted as the identity (which is certainly a comonad satisfying the necessary conditions). Unfortunately, such a simple-minded approach fails: although $t_1 = t_2$ implies $t_1^\circ = t_2^\circ$ (for the appropriate $\beta\eta$ equalities), the implication fails when equality is replaced by reduction. By getting a feel for how a naive translation fails one can use what is essentially a programmer's, rather than a mathematician's, intuition to derive a translation which works. As we have already mentioned, in this case we gain further guidance from the isomorphism

$$!A \cong I \& A \& (!A \otimes !A)$$

and from work on type-theoretic encodings of abstract datatypes. Indeed, our coding of ! has something of an object-oriented flavour: one cannot copy or discard a promoted object 'from the outside' since one does not know the number or types of its components. Instead, one asks the object to perform the operation on itself by selecting and calling the appropriate method from within the object. This type-theoretic view of object-oriented programming can actually be taken much further; Hofmann and Pierce (1992), for example, use coinductive types in more powerful variants of System F to model many features of real object-oriented languages.

One might still feel that, as LTC is first-order, the use of $\lambda 2$ is excessive, and that a simply typed lambda calculus extended with coinductive definitions should suffice (assuming that one knew that such a system is strongly normalising). This does not appear to be the case, essentially because of the 'active' nature of promoted objects. Firstly, observe that it is perfectly possible to translate $\otimes$ and I by $\times$ and 1, rather than using the $\lambda 2$ *encoding* of products. This is not true for the encoded products which appear as the first and third components of $\Phi^B(X)$ (and which morally represent I and $\otimes$). This is because in either of the following two redexes:

$$\mathsf{discard}\,(\mathsf{promote}\ \ldots\ \mathsf{for}\ \ldots\ \mathsf{in}\ \ldots)\ \mathsf{in}\ e$$

$$\mathsf{copy}\,(\mathsf{promote}\ \ldots\ \mathsf{for}\ \ldots\ \mathsf{in}\ \ldots)\ \mathsf{as}\ x, y\ \mathsf{in}\ e$$

the expression e ends up buried in the reduct at a depth which depends upon the structure of the promoted object. Thus the promoted object has to act on *e*. This means that the discard and copy methods have to be functions, each of which accepts as argument the context in which the object is to be discarded or copied. As it is impossible to know the type of that context at the point when the promoted object is constructed, the methods have to be polymorphic. Of course, this informal argument does not rule out that possibility of there being a proof which works via a translation into a simpler calculus than $\lambda 2$, but even if that were the case, the present proof has the advantage of extending trivially to cover the obvious extension of LTC with second-order quantification.

It should also be noted that even when one has the right translation of types, the translation of terms does not follow automatically. For example, the translation of promoted terms has to contain a very explicit coding up of the reduction rules associated with such terms. This is probably a strength, rather than a weakness, as it means that a wide class of calculi should be treatable using this technique.

Acknowledgements

This work owes a great deal to my collaborators Gavin Bierman, Martin Hyland and Valeria de Paiva.

References

Benton, P. N., Bierman, G. M., Hyland, J. M. E. and de Paiva, V. C. V. (1992) Term Assignment for Intuitionistic Linear Logic (Preliminary Report). Technical Report 262, Computer Laboratory, University of Cambridge.

Benton, P. N., Bierman, G. M., Hyland, J. M. E. and de Paiva, V. C. V. (1992) Linear lambda calculus and categorical models revisited. In E. Börger *et al.* (eds.), *Selected Papers from Computer Science Logic 1992. LNCS 702*, Springer-Verlag, pp. 61–84.

Benton, P. N., Bierman, G. M., Hyland, J. M. E. and de Paiva, V. C. V. (1993) A term calculus for intuitionistic linear logic. In M. Bezem and J. F. Groote (eds.), *Proc. Int. Conf. Typed Lambda Calculi and Applications*, Utrecht, The Netherlands. *LNCS 664*, Springer-Verlag, pp. 75–90.

Bierman, G. M. (1993) On Intuitionistic Linear Logic. PhD thesis, Computer Laboratory, University of Cambridge.

Chirimar, J., Gunter, C. A. and Riecke, J. G. (1995) Reference Counting as a Computational Interpretation of Linear Logic. *Journal of Functional Programming*, to appear.

Gallier, J. H. (1990) On Girard's "candidats de reducibilité". In P. Odifreddi (ed.), *Logic and Computer Science*. Academic Press, pp.123–203.

Girard, J.-Y. (1972) Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse de Doctorat d'Etat, Université de Paris VII.

Girard, J.-Y., Lafont, Y. and Taylor, P. (1989) *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

Hofmann, M. and Pierce, B. C. (1992) An Abstract View of Objects and Subtyping (Preliminary Report). Technical Report ECS-LFCS-92-226, Department of Computer Science, University of Edinburgh.

Howard, W. A. (1980) The formulæ-as-types notion of construction. In J. R. Hindley and J. P. Seldin (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press.

Mitchell, J. C. and Plotkin, G. D. (1988) Abstract types have existential type. *ACM Trans. on Programming Languages and Systems*, **10**(3): 470–502.

Plotkin, G. D. and Abadi, M. (1993) A logic for parametric polymorphism. In M. Bezem and J. F. Groote (eds.), *Proc. Int. Conf. Typed Lambda Calculi and Applications*, Utrecht, The Netherlands. *LNCS 664*, Springer-Verlag, pp. 361–375.

Reynolds, J. C. (1974) Towards a theory of type structure. *Proc. Paris Colloquium on Programming. LNCS 19*. Springer-Verlag.

Tait, W. W. (1967) Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, **32**.