Technical Report

UCAM-CL-TR-305
ISSN 1476-2986

Number 305

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Strong normalisation for the linear term calculus

P.N. Benton

July 1993

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

Strong Normalisation for the Linear Term Calculus

P. N. Benton*
University of Cambridge

Abstract

We prove a strong normalisation result for the linear term calculus which was introduced in (Benton *et al.* 1992). 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.

1 Introduction

This paper concerns the term assignment system for the multiplicative/exponential fragment of intuitionistic linear logic which is described in (Benton *et al.* 1992). A question left open in that work was whether 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 reducibility argument (Tait 1967). 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 $s \to_1 t$ in L_1 then $s^\circ \to_2^+ t^\circ$ in L_2 (where $\to_1$,$\to_2$ are the one-step reduction relations in L_1 and L_2 respectively, and $\to_2^+$ is transitive closure of $\to_2$). For, given such a translation, if there were an infinite reduction sequence

$$t_0 \to_1 t_1 \to_1 t_2 \to_1 \cdots$$

in L_1, it would induce an infinite sequence

$$t_0^\circ \to_2^+ t_1^\circ \to_2^+ t_2^\circ \to_2^+ \cdots$$

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 (LTC) and L_2 the Girard/Reynolds second order polymorphic lambda calculus (also called System F or $\lambda 2$).

*Author's address: University of Cambridge, Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, United Kingdom. Email: Nick.Benton@cl.cam.ac.uk. Research supported by a SERC Fellowship.

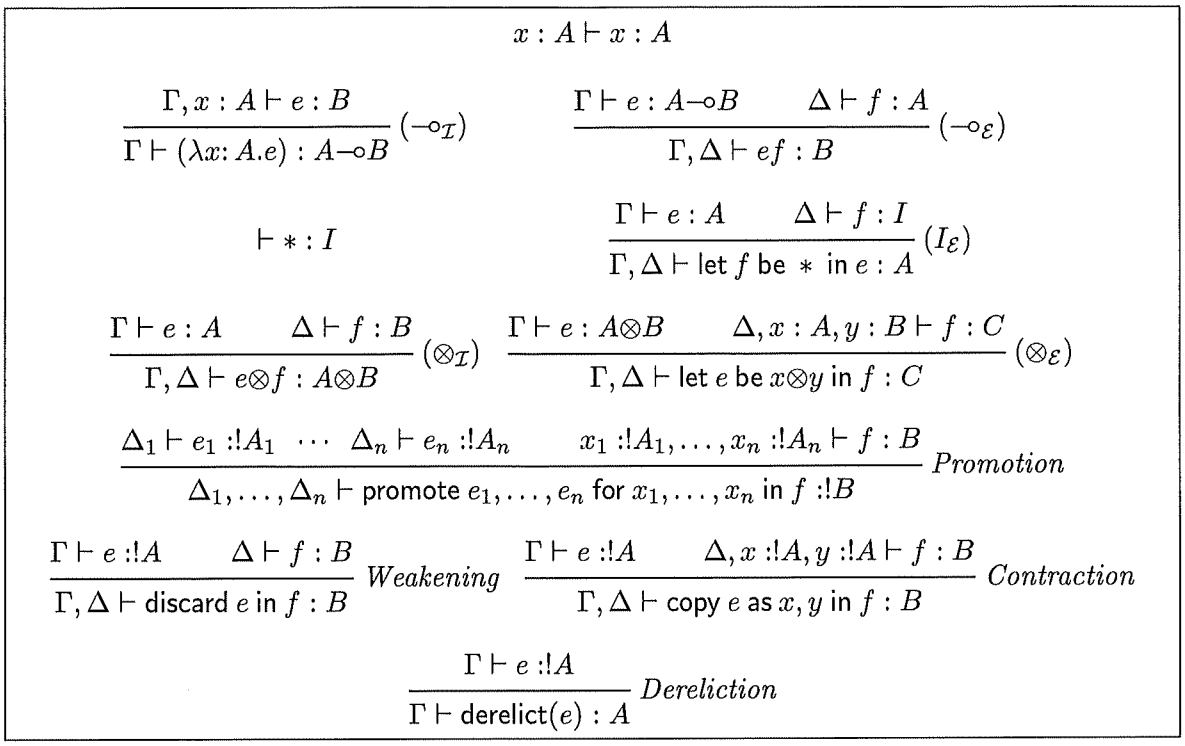


Figure 1: The linear term calculus (LTC)

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:

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

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

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

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

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

It should be noted that in (Benton *et al.* 1992) 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.

$(\lambda x.t)e$	$\rightarrow$	$t[e/x]$
let $*$ be $*$ in e	$\rightarrow$	e
let $e \otimes t$ be $x \otimes y$ in u	$\rightarrow$	$u[e/x, t/y]$
derelict(promote e_i for x_i in t)	$\rightarrow$	$t[e_i/x_i]$
discard (promote e_i for x_i in t) in u	$\rightarrow$	discard e_i in u
copy (promote e_i for x_i in t) as y, z in u	$\rightarrow$	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]

Figure 2: β-reductions for the linear term calculus

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

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

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

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

Figure 3: The second order polymorphic lambda calculus ($\lambda 2$)

1.2 The Second-Order Polymorphic Lambda Calculus

The types of $\lambda 2$ are given by the following grammar, where G ranges over a given set of base types and X over a set of type variables:

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

Note that, for notational convenience in what follows, we have presented the system with n-ary product types. These are not strictly necessary as products can be coded within the $\forall \rightarrow$ fragment in the usual way (see (Girard *et al.* 1989), for example). The term-formation rules for $\lambda 2$ are recalled in Figure 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* $\Gamma, x : B \vdash e : A$.

Proof. Induction on derivations. □

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

$$\begin{array}{rcl}(\lambda x{:}\,A.e)\,f & \to & e[f/x] \\ \pi_i\langle e_1,\ldots,e_n\rangle & \to & e_i\end{array}$$

together with the following axiom for reduction on types:

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

and a collection of inference rules extending $\to$ to a congruence.

The important fact about reduction of terms in $\lambda 2$ is that it always terminates:

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

Proof. See (Girard *et al.* 1989, Chapter 14). □

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

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

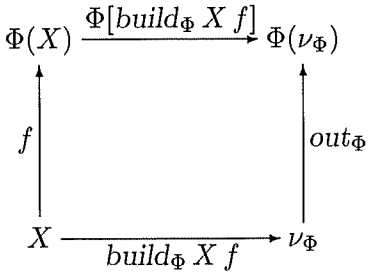
and associated destructor is

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

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



which characterises (ν_Φ, out_Φ) as a (weakly) terminal Φ-coalgebra. (This is strongly terminal in models satisfying appropriate parametricity conditions. See (Plotkin & 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:

$$G^\circ = G$$
$$(A \multimap B)^\circ = A^\circ \to B^\circ$$
$$(A \otimes B)^\circ = \forall X.(A^\circ \to B^\circ \to X) \to X$$
$$I^\circ = \forall X.X \to X$$
$$(!A)^\circ = \nu_{\Phi A^\circ}$$

where

$$\Phi^B(X) = (\forall Z.Z \to Z) \times B \times (\forall Z.(X \to X \to Z) \to Z)$$

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 $\nu_{\Phi B}$ 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

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

(where & is the additive conjunction 'with', which we do not treat here) 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 & Plotkin 1988).

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

$$\Phi^C[f] \;=\; \lambda w\colon \Phi^C(A).\, \langle \pi_1 w, \pi_2 w, \Lambda Z.\lambda h\colon B \to B \to Z.(\pi_3 w)\, Z\, (\lambda x\colon A.\lambda y\colon A.h\,(f\,x)\,(f\,y))\rangle$$

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 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:
$$\frac{\dfrac{\Gamma^\circ \vdash out_{\Phi A^\circ} : \nu_{\Phi A^\circ} \to \Phi^{A^\circ}(\nu_{\Phi A^\circ}) \qquad \Gamma^\circ \vdash e^\circ : \nu_{\Phi A^\circ}}{\Gamma^\circ \vdash (out_{\Phi A^\circ}\, e^\circ) : (\forall Z.Z \to Z) \times A^\circ \times (\forall Z.(\nu_{\Phi A^\circ} \to \nu_{\Phi A^\circ} \to Z) \to Z)}}{\Gamma^\circ \vdash \pi_2(out_{\Phi A^\circ}\, e^\circ) : A^\circ}$$

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

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

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

- 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

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

- If Π ends in

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

then we form Π° proving

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

where

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

and

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

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 $e_1 \to e_2$ then $e_1^\circ \to^+ e_2^\circ$ in $\lambda 2$.*

Proof. By induction on the derivation of $e_1 \to e_2$. 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

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

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

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

This expression then reduces to

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

and then to

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

After expanding the definition of c, this leads to

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

which is

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

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 morally 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

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

Being slightly less naive, one might still feel that the use of $\lambda 2$ is overkill, as LTC is first-order, and that a simply typed lambda calculus extended with coinductive definitions would suffice (assuming that one knows that such a system is strongly normalising). This appears to require that the lambda calculus be presented using the pattern-matching destructor split (and its corresponding nullary version) in place of projections (*cf.* our use of the $\lambda 2$ *codings* of products to translate $\otimes$ and I), and even then it does not seem possible to define (at least in any natural way) a translation which works. In any case, the present translation into $\lambda 2$ 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. Technical Report 262, Computer Laboratory, University of Cambridge.

Bierman, G. M. (1993) PhD Thesis, Computer Laboratory, University of Cambridge. To appear.

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

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