A Terminating and Confluent Linear Lambda Calculus

Yo Ohta and Masahito Hasegawa

Research Institute for Mathematical Sciences, Kyoto University
Kyoto 606-8502, Japan

Abstract. We present a rewriting system for the linear lambda calculus corresponding to the $\{!, \multimap\}$-fragment of intuitionistic linear logic. This rewriting system is shown to be strongly normalizing, and Church-Rosser modulo the trivial commuting conversion. Thus it provides a simple decision method for the equational theory of the linear lambda calculus. As an application we prove the strong normalization of the simply typed computational lambda calculus by giving a reduction-preserving translation into the linear lambda calculus.

1 Introduction

In the literature, there exist many proposals of linearly typed lambda calculi which correspond to Girard's linear logic [7] via the Curry-Howard correspondence. However, only a few of them have studied the equality between terms (or proofs) seriously. Just like the simply typed lambda calculus with the $\beta\eta$-equality is sound and complete for semantic models given by cartesian closed categories [13,5], it is desirable for a linear lambda calculus to be equipped with an equational theory which is sound and complete for the now well-established categorical models of linear logic [19,3,4,16].

Barber and Plotkin's *Dual Intuitionistic Linear Logic (DILL)* [1,2] is one of such calculi: its equational theory, determined by the standard $\beta\eta$-axioms and a few axioms for commuting conversions (for identifying the terms representing the same proof modulo trivial proof permutations), has been shown to be sound and complete for the categorical models of the multiplicative exponential fragment of the intuitionistic linear logic. Together with its natural-deduction style simple term expressions, DILL can be considered as one of the canonical calculi for this fragment of linear logic.

However, DILL is not equipped with a rewriting system. There is a symmetric un-orientable axiom for commuting conversions, thus it is not clear if the equational theory of DILL has a simple decision procedure based on a rewriting system, while it is the case for many of the standard typed lambda calculi.

Regarding decidability, the answer is actually known: Barber [1] in his PhD thesis, and independently Ghani [6] in an unpublished manuscript, have shown that the equational theory of DILL is decidable. However, their proofs are long and complicated, using some new notations and/or advanced techniques which

F. Pfenning (Ed.): RTA 2006, LNCS 4098, pp. 166–180, 2006.
© Springer-Verlag Berlin Heidelberg 2006

are not always easy to follow. Barber's approach involves a translation into a net-like system and rewriting on equivalence classes of expressions. Ghani have used the η-expansion technique which again is a rather heavy machinery. At least, they do not present a simple and intuitively understandable rewriting system in the traditional sense.

Here we propose a simpler solution for the $\{\multimap, !\}$-fragment (which is enough to mimic the simply typed lambda calculus via Girard's translation as $\sigma \to \tau = !\sigma \multimap \tau$) by a classical rewriting-theoretic method. Specifically, we appeal to the seminal result by Huet on *reduction modulo equivalence* [12]. We provide a rewriting system $\succ$ together with a (trivially) decidable equational theory $\sim^*$ generated by the symmetric commuting conversion $\sim$ on linear lambda terms such that (following the terminology of Terese [20])

1. The equivalence relation generated from $\succ$ and $\sim$ agrees with the equational theory of the linear lambda calculus,
2. $\succ$ is strongly normalizing,
3. $\succ$ is locally confluent modulo $\sim^*$, and
4. $\succ$ is locally coherent modulo $\sim^*$.

Then Huet's theorem implies that $\succ$ is Church-Rosser modulo $\sim^*$, and deciding the equality in this linear lambda calculus is reduced to comparing the $\succ$-normal forms up to the easily decidable equality $\sim^*$.

From rewriting-theoretical point of view, this work does not present much new idea. However, it does give an interesting case motivated by the study on the semantic and logical foundations of functional programming languages. Recent work [8,9,10] suggest that there exist many interesting translations of various calculi into this linear lambda calculus, including monadic and CPS translations. As an interesting example, we prove the strong normalization of the simply typed version of Moggi's computational lambda calculus by giving a reduction-preserving translation into the linear lambda calculus. Together with this result, our work can be considered as a follow-up of the work by Maraist et al. [15] and Sabry and Wadler [18].

The rest of this paper is organized as follows. We introduce the linear lambda calculus in Section 2, and our rewriting system in Section 3. Section 4 is a quick reminder of the classical definitions and result from the theory of reduction modulo equivalence. Section 5, 6 and 7 are devoted to show the strong normalization, local confluence modulo $\sim^*$, and local coherence modulo $\sim^*$, which jointly imply the Church-Rosser property modulo equivalence. Section 8 gives a reduction-preserving translation from the simply typed computational lambda calculus to the linear lambda calculus. Some concluding remarks are given in Section 9.

2 The Linear Lambda Calculus with $\multimap$ and !

The calculus to be considered below is a dual-context natural deduction system for the $\{!, \multimap\}$-fragment of IMELL, based on DILL of Barber and Plotkin [1,2].

The identical calculus appears in [8]. In this formulation of the linear lambda calculus, a typing judgement takes the form $\Gamma\ ;\ \Delta \vdash M : \tau$ in which Γ represents an intuitionistic (or additive) context whereas Δ is a linear (multiplicative) context. We assume that all variables in Γ and Δ are distinct. While the variables in Γ can be used in the term M as many times as we like, those in Δ must be used exactly once. A typing judgement $x_1 : \sigma_1, \ldots, x_m : \sigma_m\ ;\ y_1 : \tau_1, \ldots, y_n : \tau_n \vdash M : \sigma$ can be considered as the proof of the sequent $!\sigma_1, \ldots, !\sigma_m, \tau_1, \ldots, \tau_n \vdash \sigma$, or the proposition $!\sigma_1 \otimes \ldots \otimes !\sigma_m \otimes \tau_1 \otimes \ldots \otimes \tau_n \multimap \sigma$.

Types and Terms

$$\begin{aligned} \sigma &::= b \mid \sigma \multimap \sigma \mid\ !\sigma \\ M &::= x \mid \lambda x^{\sigma}.M \mid M\,M \mid\ !M \mid \mathsf{let}\ !x^{\sigma}\ \mathsf{be}\ M\ \mathsf{in}\ M \end{aligned}$$

where b ranges over a set of base types. We may omit the type subscripts for ease of presentation.

Typing

$$\frac{}{\Gamma\ ;\ x : \tau \vdash x : \tau}\ \text{LinAx} \qquad \frac{}{\Gamma_1, x : \tau, \Gamma_2\ ;\ \emptyset \vdash x : \tau}\ \text{IntAx}$$

$$\frac{\Gamma\ ;\ \Delta, x : \tau_1 \vdash M : \tau_2}{\Gamma\ ;\ \Delta \vdash \lambda x^{\tau_1}.M : \tau_1 \multimap \tau_2}\ \multimap\text{Intro} \qquad \frac{\Gamma; \Delta_1 \vdash M : \tau_1 \multimap \tau_2 \quad \Gamma; \Delta_2 \vdash N : \tau_1}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash MN : \tau_2}\ \multimap\text{Elim}$$

$$\frac{\Gamma\ ;\ \emptyset \vdash M : \tau}{\Gamma\ ;\ \emptyset \vdash !M : !\tau}\ !\text{Intro} \qquad \frac{\Gamma\ ;\ \Delta_1 \vdash M : !\tau_1 \quad \Gamma, x : \tau_1\ ;\ \Delta_2 \vdash N : \tau_2}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \mathsf{let}\ !x^{\tau_1}\ \mathsf{be}\ M\ \mathsf{in}\ N : \tau_2}\ !\text{Elim}$$

where $\emptyset$ is the empty context, and $\Delta_1 \sharp \Delta_2$ is a merge of Δ_1 and Δ_2 [1,2]. Thus, $\Delta_1 \sharp \Delta_2$ represents one of possible merges of Δ_1 and Δ_2 as finite lists. More explicitly, we can define the relation "Δ is a merge of Δ_1 and Δ_2" inductively as follows [1]:

- Δ is a merge of $\emptyset$ and Δ
- Δ is a merge of Δ and $\emptyset$
- if Δ is a merge of Δ_1 and Δ_2, then $x : \sigma, \Delta$ is a merge of $x : \sigma, \Delta_1$ and Δ_2
- if Δ is a merge of Δ_1 and Δ_2, then $x : \sigma, \Delta$ is a merge of Δ_1 and $x : \sigma, \Delta_2$

We assume that, when we introduce $\Delta_1 \sharp \Delta_2$, there is no variable occurring both in Δ_1 and in Δ_2. We note that any typing judgement has a unique derivation (hence a typing judgement can be identified with its derivation).

Axioms

$$\begin{array}{lll} \beta_{\multimap} & (\lambda x.M)\,N & = M[N/x] \\ \eta_{\multimap} & \lambda x.M\,x & = M \\ \beta_{!} & \mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N & = N[M/x] \\ \eta_{!} & \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ !x & = M \\ com & C[\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N] & = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[N] \end{array}$$

where $M[N/x]$ denotes the capture-free substitution, while $C[-]$ is a linear context (no ! binds $[-]$):

$$C\ ::=\ [-] \mid \lambda x.C \mid C\,M \mid M\,C \mid \mathsf{let}\ !x\ \mathsf{be}\ C\ \mathsf{in}\ M \mid \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C$$

The use of linear contexts is crucial: *com* is not allowed for non-linear contexts, e.g. the "idempotency equation" [2] $!(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ x) = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ !x$ (which implies the idempotency of !, i.e., $!!\sigma \simeq !\sigma$) is not derivable. The equality judgement $\Gamma\ ;\ \Delta \vdash M = N : \sigma$, where $\Gamma\ ;\ \Delta \vdash M : \sigma$ and $\Gamma\ ;\ \Delta \vdash N : \sigma$, is defined as the congruence relation on the well-typed terms of the same type under the same typing context, generated from these axioms.

In the sequel, we work on terms up to the α-congruence. We may write $M = N$ as a shorthand for the equality judgement $\Gamma\ ;\ \Delta \vdash M = N : \sigma$, while we will use $M \equiv N$ for expressing that M and N are the same modulo α-congruence.

The axiom *com* expresses the commuting conversions. By induction on the construction of linear contexts, *com* can be expressed by five explicit instances:

Proposition 1. *The axiom com can be replaced by the following five axioms.*

$$
\begin{array}{ll}
com_1\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\,L & = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N\,L \\
com_2\ \mathsf{let}\ !y\ \mathsf{be}\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L & = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ L \\
com_3\ \lambda y.(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N) & = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.N \\
com_4\ L\,(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N) & = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,N \\
com_5\ \mathsf{let}\ !x\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ M\ \mathsf{in}\ N & = \mathsf{let}\ !y\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !x\ \mathsf{be}\ L\ \mathsf{in}\ N
\end{array}
$$

□

Remark 1. As stated above, we only consider the equality on the well-typed terms under the same typing contexts. Thus, for example, in com_3, y cannot be free in M; and in com_5, x and y cannot be free in L and M.

Remark 2. As noted in [11], this linear lambda calculus allows a yet simpler axiomatization:

$$
\begin{array}{ll}
\beta_{\multimap}\ (\lambda x.M)\,N & = M[N/x] \\
\eta_{\multimap}\ \lambda x.M\,x & = M \\
\beta_{!}\ \ \mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N & = N[M/x] \\
\eta'_{!}\ \ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,(!x) & = L\,M
\end{array}
$$

While this is very compact, it does not immediately hint a terminating confluent rewriting system. Nevertheless, we will see later that a rewrite rule similar to this $\eta'_{!}$ is needed for obtaining such a rewriting system.

3 A Rewriting System for the Linear Lambda Calculus

3.1 Motivating the Rewriting Rules

Now let us derive a rewriting system for the linear lambda calculus from its axioms. As a natural starting point, we orient the $\beta\eta$-axioms from left to right, as the case of the standard $\beta\eta$ lambda calculus. The commuting conversions are tricky, however. First of all, it is not possible to orient the symmetric axiom com_5, so it needs to be treated separately. Here we follow the tradition of *reduction modulo equivalence*: we design our system so that com_5-reasoning can be

postponed after all other rewriting steps are done. For $com_{1\sim4}$, it seems natural to orient the axioms so that the let-bindings are pulled outside the contexts, i.e.,

$$\begin{array}{lll}
com_1 & (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\,L & \succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N\,L \\
com_2 & \mathsf{let}\ !y\ \mathsf{be}\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L & \succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ L \\
com_3 & \lambda y.(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N) & \succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.N \\
com_4 & L\,(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N) & \succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,N
\end{array}$$

thus flattening the let-expressions as possible as we can. Alas, there is a problem on these rules and $\eta_!$:

- $\eta_!$ and $com_{1\sim4}$ give a non-joinable critical pair, e.g.

$$\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,(!x) \overset{com_4}{\longleftarrow} L\,(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ !x) \xrightarrow{\eta_!} L\,M$$

- The same problem happens with com_5:

$$\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ !x \overset{com_5}{\longleftarrow} \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ !x \xrightarrow{\eta_!} \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ M$$

To overcome this difficulty, we introduce a refined version of $\eta_!$

$$\eta'_! \quad \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[!x] \succ C[M]$$

(where C ranges over the linear contexts as before) for which this problem disappears.

3.2 Rewriting System

Our rewriting system features the following rules.

$\beta_{\multimap}$	$(\lambda x.M)\,N$	$\succ M[N/x]$
$\eta_{\multimap}$	$\lambda x.M\,x$	$\succ M$
$\beta_!$	$\mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N$	$\succ N[M/x]$
$\eta'_!$	$\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[!x]$	$\succ C[M]$
com_1	$(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\,L$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N\,L$
com_2	$\mathsf{let}\ !y\ \mathsf{be}\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ L$
com_3	$\lambda y.(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.N$
com_4	$L\,(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,N$

We may use $\succ$ for the compatible relation on the well-typed terms generated by these rules (one-step rewriting), and $\succ^*$ will denote its reflexive transitive closure (many-step rewriting). We note that the *com*-rewriting rules can be summarized as

$$D[\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N] \ \succ\ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ D[N]$$

where $D ::= [-]\,L \mid \mathsf{let}\ !y\ \mathsf{be}\ [-]\ \mathsf{in}\ L \mid \lambda y.[-] \mid L\,[-]$.

We also have to consider the symmetric rule com_5:

com_5	$\mathsf{let}\ !x\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ M\ \mathsf{in}\ N$	$\sim\ \mathsf{let}\ !y\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !x\ \mathsf{be}\ L\ \mathsf{in}\ N$

We write $\sim$ for the compatible relation generated by com_5 (one-step reasoning via com_5), and $\sim^*$ for its reflexive transitive closure. A few easy facts:

Proposition 2. *The reflexive symmetric transitive closure of* $\succ \cup \sim$ *coincides with the equality of the linear lambda calculus.* □

Proposition 3. *Each equivalence class of* $\sim^*$ *is finite, and thus* $\sim^*$ *is decidable.* □

The following result, easily shown by induction, will be useful in proving the local confluence:

Lemma 1. $C[\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N] \succ^* \cdot \sim^* \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[N]$. □

Remark 3. In passing, we shall note that our rewriting system $\succ$ can simulate the $\beta\eta$-reduction in the simply typed lambda calculus via Girard translation [7]: types are translated as $b^\circ = b$ and $(\sigma \to \tau)^\circ = !\sigma^\circ \multimap \tau^\circ$, and for terms we have

$$\begin{aligned} x^\circ &\equiv x \\ (\lambda x.M)^\circ &\equiv \lambda y.\mathsf{let}\ !x\ \mathsf{be}\ y\ \mathsf{in}\ M^\circ \\ (MN)^\circ &\equiv M^\circ(!N^\circ) \end{aligned}$$

For further details, see e.g. [8]. It is immediate to see that each $\beta\eta$-reduction in the simply typed lambda calculus is sent to non-trivial reduction in $\succ$:

$$\begin{aligned} ((\lambda x.M)\,N)^\circ &\equiv (\lambda y.\mathsf{let}\ !x\ \mathsf{be}\ y\ \mathsf{in}\ M^\circ)\,(!N^\circ) & \\ &\succ \mathsf{let}\ !x\ \mathsf{be}\ !N^\circ\ \mathsf{in}\ M^\circ & (\beta_{\multimap}) \\ &\succ M^\circ[N^\circ/x] & (\beta_!) \\ &\equiv (M[N/x])^\circ & \end{aligned}$$

$$\begin{aligned} (\lambda x.M\,x)^\circ &\equiv \lambda y.\mathsf{let}\ !x\ \mathsf{be}\ y\ \mathsf{in}\ M^\circ\,(!x) & \\ &\succ \lambda y.M^\circ\,y & (\eta'_!) \\ &\succ M^\circ & (\eta_{\multimap}) \end{aligned}$$

4 Rewriting Modulo Equivalence

In the following sections, we will show that our rewriting system together with $\sim$ gives a decision procedure of the equality on the linear lambda terms. Fortunately, it turns out that a classical result due to Huet is directly applicable to our case. Below we recall basic definitions on reduction modulo equivalence for abstract rewriting systems (ARS's) and state Huet's theorem. We follow the treatment in Terese (Chapter 14.3) [20].

Definition 1. *Let* $(A, \to)$ *be an ARS, and* $\sim$ *be an equivalence relation on* A. *We say:*

1. a, b *are* joinable modulo $\sim$ *if there exist* c, d *such that* $a \to^* c$, $b \to^* d$ *and* $c \sim d$.
2. $\to$ *is* locally confluent modulo $\sim$ *if, for any* a, b, c, $a \to b$ *and* $a \to c$ *imply* b *and* c *are joinable modulo* $\sim$.

3. $\rightarrow$ *is* locally coherent modulo $\sim$ *if, for any* a, b, c, $a \rightarrow b$ *and* $a \sim c$ *imply that* b *and* c *are joinable modulo* $\sim$.
4. $\rightarrow$ *is* Church-Rosser modulo $\sim$ *if* $a \approx b$ *implies* a *and* b *are joinable modulo* $\sim$, *where* $\approx$ *is* $(\sim \cup \rightarrow \cup \leftarrow)^*$.

What we wish to establish for our system on linear lambda terms is the strong normalization of $\succ$ and the Church-Rosser property of $\succ$ modulo $\sim^*$. The following result provides a sufficient condition for this.

Theorem 1 (Huet [12]). *Let* $(A, \succ)$ *be an ARS, and* $\sim$ *be an equivalence relation on* A. *If* $\succ$ *is strongly normalizing, locally confluent modulo* $\sim$, *and locally coherent with* $\sim$, *then* $\succ$ *is Church-Rosser modulo* $\sim$. □

In the following three sections, we show that $\succ$ is (i) strongly normalizing, (ii) locally confluent modulo $\sim^*$, and (iii) locally coherent with $\sim^*$.

5 Strong Normalization

Theorem 2 (strong normalization). $\succ$ *is strongly normalizing.*

For proving this, we proceed as follows. First, by showing that a translation into the simply typed lambda calculus weakly preserves the reduction, we reduce the problem to that of the smaller rewriting system. We then show the termination of this subsystem by assigning natural numbers to expressions which are strictly decreasing with respect to the reduction steps.

5.1 Translation into the Simply Typed Lambda Calculus

There is an obvious translation from the linear lambda calculus into the simply typed $\beta\eta$-lambda calculus (an inverse to Girard's translation [8]) which weakly preserves the reductions.

$$
\begin{array}{ll}
b^\bullet & = b \\
(!\tau)^\bullet & = \tau^\bullet \\
(\tau_1 \multimap \tau_2)^\bullet & = \tau_1^\bullet \rightarrow \tau_2^\bullet \\
 & \\
x^\bullet & \equiv x \\
(\lambda x^\tau.M)^\bullet & \equiv \lambda x^{\tau^\bullet}.M^\bullet \\
(M\ N)^\bullet & \equiv M^\bullet\ N^\bullet \\
(!M)^\bullet & \equiv M^\bullet \\
(\mathsf{let}\ !x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N)^\bullet & \equiv N^\bullet[M^\bullet/x]
\end{array}
$$

Straightforward inductions show the following facts:

Lemma 2 (type soundness). $\Gamma\,;\,\Delta \vdash M : \tau$ *implies* $\Gamma^\bullet, \Delta^\bullet \vdash M^\bullet : \tau^\bullet$. □

Lemma 3 (substitution lemma). $(M[N/x])^\bullet \equiv M^\bullet[N^\bullet/x]$. □

Now we see how reductions in the linear lambda calculus are related to those on the simply typed lambda calculus.

Proposition 4. *If $M \succ N$ in the linear lambda calculus, then $M^\bullet \succ_{\beta\eta} N^\bullet$ or $M^\bullet \equiv N^\bullet$ in the simply typed lambda calculus.*

Proof. It suffices to look at the reduction rules.

$$
\begin{aligned}
((\lambda x.M)\, N)^\bullet &\equiv (\lambda x.M^\bullet)\, N^\bullet \\
&\succ_\beta M^\bullet[N^\bullet/x] \\
&\equiv (M[N/x])^\bullet \quad \text{by Lemma 3}
\end{aligned}
$$

$$
\begin{aligned}
(\lambda x.M\, x)^\bullet &\equiv \lambda x.M^\bullet\, x \\
&\succ_\eta M
\end{aligned}
$$

$$
\begin{aligned}
(\mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N)^\bullet &\equiv N^\bullet[M^\bullet/x] \\
&\equiv (N[M/x])^\bullet
\end{aligned}
$$

$$
\begin{aligned}
(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[!x])^\bullet &\equiv (C[!x])^\bullet[M^\bullet/x] \\
&\equiv (C[M])^\bullet
\end{aligned}
$$

$$
\begin{aligned}
((\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\, L)^\bullet &\equiv N^\bullet[M^\bullet/x]\, L^\bullet \\
&\equiv (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N\, L)^\bullet
\end{aligned}
$$

$$
\begin{aligned}
(\mathsf{let}\ !y\ \mathsf{be}\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L)^\bullet &\equiv L^\bullet[N^\bullet[M^\bullet/x]/y] \\
&\equiv L^\bullet[N^\bullet/y][M^\bullet/x] \\
&\equiv (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ L)^\bullet
\end{aligned}
$$

$$
\begin{aligned}
(L\, (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N))^\bullet &\equiv L^\bullet\, (N^\bullet[M^\bullet/x]) \\
&\equiv (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\, N)^\bullet
\end{aligned}
$$

$$
\begin{aligned}
(\lambda y.\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)^\bullet &\equiv \lambda y.N^\bullet[M^\bullet/x] \\
&\equiv (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.N)^\bullet
\end{aligned}
$$

□

Corollary 1. *Strong normalization of $\beta_!, \eta'_!, com_1, com_2, com_3$ and com_4 implies that of $\succ$.*

Proof. Suppose that $\succ$ is not strongly normalizing, thus there exists an infinite strict reduction sequence $M_0 \succ M_1 \succ \ldots$ in the linear lambda calculus. We then have an infinite sequence $M_0^\bullet, M_1^\bullet, \ldots$ in the simply typed lambda calculus, where $M_i^\bullet \succ_{\beta\eta} M_{i+1}^\bullet$ or $M_i^\bullet \equiv M_{i+1}^\bullet$ holds by the last proposition. Since the $\beta\eta$-reduction of the simply typed lambda calculus is strongly normalizing, there exists some n such that $M_m^\bullet \equiv M_n^\bullet$ holds for any $m \geq n$. This means that the infinite reduction sequence $M_n \succ M_{n+1} \succ \ldots$ consists just of the non-$\beta_{-\circ}\eta_{-\circ}$ reductions. □

5.2 Termination of the Subsystem

We now complete our proof of strong normalization by showing that $\beta_!, \eta'_!, com_1, com_2, com_3, com_4$ is indeed strongly normalizing.

Proposition 5. *The following set of rewriting rules is strongly normalizing.*

$\beta_!$	$\mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N$	$\succ N[M/x]$
$\eta'_!$	$\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[!x]$	$\succ C[M]$
com_1	$(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\,L$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N\,L$
com_2	$\mathsf{let}\ !y\ \mathsf{be}\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ L$
com_3	$\lambda y.(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.N$
com_4	$L\,(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$\succ \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,N$

Proof. We assign a positive natural number $|M|$ to each (possibly non-well-typed) term M by

$$\begin{aligned}
|x| &= 1\\
|\lambda x^\tau.M| &= 2|M|\\
|M\,N| &= 2|M| + 2|N|\\
|!M| &= |M|\\
|\mathsf{let}\ !x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N| &= 2|M| + |N[M/x]|
\end{aligned}$$

(note that the last line is well-defined — compare the depths of let-bindings) and show that $M \succ N$ implies $|M| > |N|$. Note that this assignment is monotone with respect to each argument. Therefore $|L[M/x]| \geq |L[N/x]|$ holds if we know $|M| \geq |N|$.

- $\beta_!$: $|\mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N| = 2|!M| + |N[!M/x]| = 2|M| + |N[M/x]| > |N[M/x]|$.
- $\eta'_!$: $|\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[!x]| = 2|M| + |C[!M]| = 2|M| + |C[M]| > |C[M]|$.
- com_1: $|(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\,L| = 4|M| + 2|N[M/x]| + 2|L|$, while $|\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N\,L| = 2|M| + 2|N[M/x]| + 2|L|$.
- com_2:

$$\begin{aligned}
&|\mathsf{let}\ !y\ \mathsf{be}\ (\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L|\\
&= 2|\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N| + |L[\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N/y]|\\
&= 4|M| + 2|N[M/x]| + |L[\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N/y]|\\
&\geq 4|M| + 2|N[M/x]| + |L[N[M/x]/y]|
\end{aligned}$$

  while

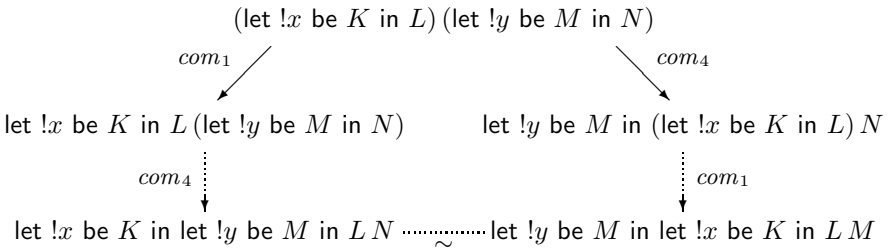
$$\begin{aligned}
&|\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ N\ \mathsf{in}\ L|\\
&= 2|M| + |\mathsf{let}\ !y\ \mathsf{be}\ N[M/x]\ \mathsf{in}\ L|\\
&= 2|M| + 2|N[M/x]| + |L[N[M/x]/y]|
\end{aligned}$$

- com_3, com_4: similar to the case of com_1. □

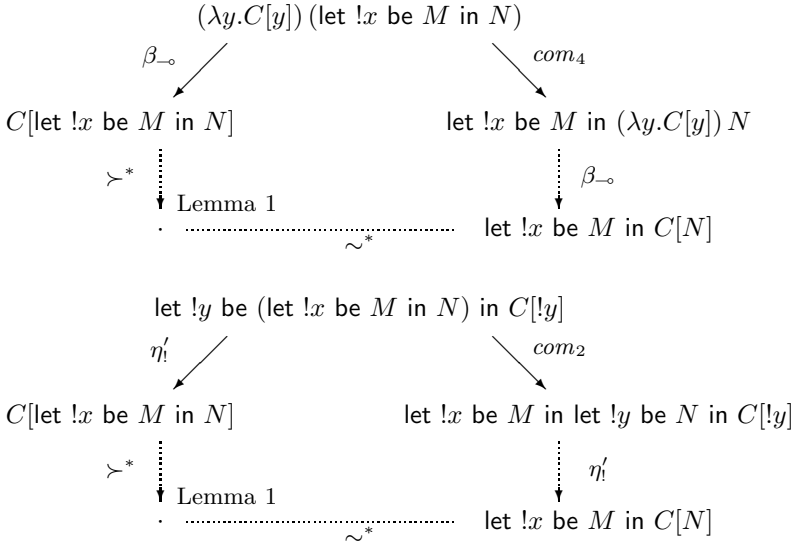
6 Local Confluence Modulo Equivalence

Theorem 3 (local confluence modulo $\sim^*$). *$\succ$ is locally confluent modulo $\sim^*$: if $L \succ M_1$ and $L \succ M_2$ then there exist N_1 and N_2 such that $M_1 \succ^* N_1$, $M_2 \succ^* N_2$ and $N_1 \sim^* N_2$.*

Proof (sketch). There are 16 cases to be considered. Many of them are joinable without $\sim^*$, except the following three cases.



Other two cases involve $\sim^*$ via Lemma 1 (Section 3.2).

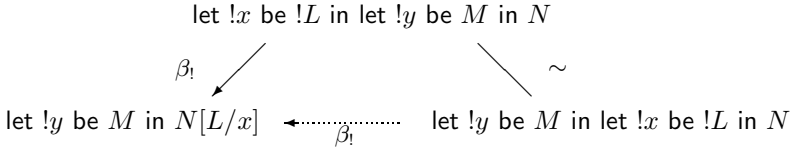


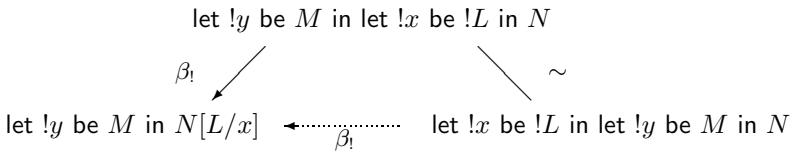
□

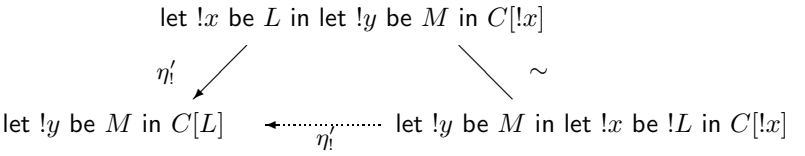
7 Local Coherence Modulo Equivalence

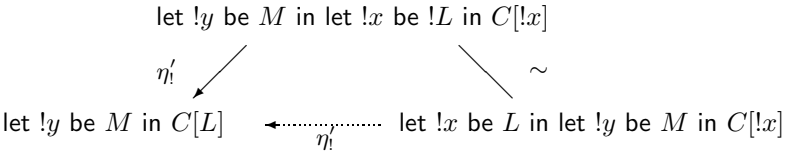
Theorem 4 (local coherence modulo $\sim^*$). *$\succ$ is locally coherent modulo $\sim^*$, i.e., if $L \sim^* M \succ N$ then there exists some L', N' such that $L \succ^* L'$, $N \succ^* N'$ and $L' \sim^* N'$.*

Proof (sketch). Note that it suffices to show: if $L \sim M \succ N$ then there exists some L', N' such that $L \succ^* L'$, $N \succ^* N'$ and $L' \sim^* N'$. There are six cases to be considered. The first four are rather obvious:

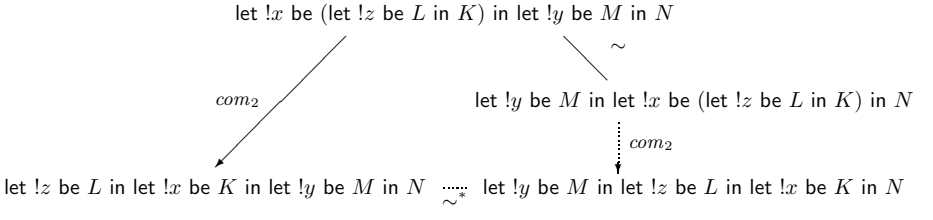


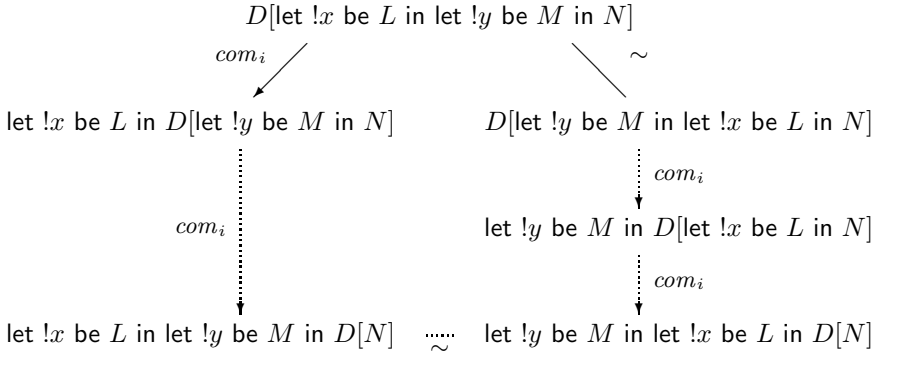






The remaining two cases are less trivial:





□

Now we can state the fruit of the last three sections, thanks to Theorem 1.

Theorem 5 (Church-Rosser modulo $\sim^*$). *$\succ$ is Church-Rosser modulo $\sim^*$; if $M = N$, there exist M', N' such that $M \succ^* M'$, $N \succ^* N'$ and $M' \sim^* N'$.* □

8 Translation from the Computational Lambda Calculus

8.1 The Simply Typed Computational Lambda Calculus λ_c

The simply typed computational lambda calculus λ_c (its untyped version was introduced by Moggi [17]) has the same syntax as the simply typed lambda calculus plus the let-binding

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ N : \tau}$$

It is a *call-by-value* calculus however, and its rewriting / equational theory is valid for reasoning about programs in the call-by-value programming languages like ML and Scheme. λ_c features the following reduction rules:

$(\beta.v)$	$(\lambda x^\sigma.M)\,V$	$\succ M[V/x]$	
$(\eta.v)$	$\lambda x^\sigma.V\,x$	$\succ V$	$(x \notin \mathrm{FV}(V))$
$(\beta.let)$	$\mathsf{let}\ x^\sigma\ \mathsf{be}\ V\ \mathsf{in}\ M$	$\succ M[V/x]$	
$(\eta.let)$	$\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ x$	$\succ M$	
$(assoc)$	$\mathsf{let}\ y^\tau\ \mathsf{be}\ (\mathsf{let}\ x^\sigma\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N$	$\succ \mathsf{let}\ x^\sigma\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ y^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N$	
$(let.1)$	$P\,M$	$\succ \mathsf{let}\ x^\sigma\ \mathsf{be}\ P\ \mathsf{in}\ x\,M$	$(P : \sigma)$
$(let.2)$	$V\,Q$	$\succ \mathsf{let}\ y^\sigma\ \mathsf{be}\ Q\ \mathsf{in}\ V\,y$	$(Q : \sigma)$

where V, W range over values (variables and lambda abstractions) while P, Q over non-values (applications and let expressions).

8.2 The Kernel Computational Lambda Calculus λ_{c*}

Interestingly, the reductions in the λ_c-calculus can be simulated within a smaller sublanguage λ_{c*} called *kernel computational lambda calculus* [18]. In λ_{c*}, applications $M\,N$ are restricted to those of values $V\,W$, and we no longer have reduction rules $(let.1)$ and $(let.2)$. Its reduction rules are given as follows.

$(\beta.v)$	$(\lambda x^\sigma.M)\,V$	$\succ M[V/x]$	
$(\eta.v)$	$\lambda x^\sigma.V\,x$	$\succ V$	$(x \notin \mathrm{FV}(V))$
$(\beta.let)$	$\mathsf{let}\ x^\sigma\ \mathsf{be}\ V\ \mathsf{in}\ M$	$\succ M[V/x]$	
$(\eta.let)$	$\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ x$	$\succ M$	
$(assoc)$	$\mathsf{let}\ y^\tau\ \mathsf{be}\ (\mathsf{let}\ x^\sigma\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N$	$\succ \mathsf{let}\ x^\sigma\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ y^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N$	

Here is a reduction-preserving inclusion $(-)^*$ from λ_c into λ_{c*}:

$$\begin{array}{ll}
x^* & \equiv x \\
(\lambda x^\sigma.M)^* & \equiv \lambda x^\sigma.M^* \\
(P\,M)^* & \equiv \mathsf{let}\ x\ \mathsf{be}\ P^*\ \mathsf{in}\ (\lambda y.(y\,M)^*)\,x \\
(V\,Q)^* & \equiv \mathsf{let}\ y\ \mathsf{be}\ Q^*\ \mathsf{in}\ (\lambda x.V^*\,x)\,y \\
(V\,W)^* & \equiv V^*\,W^* \\
(\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ N)^* & \equiv \mathsf{let}\ x^\sigma\ \mathsf{be}\ M^*\ \mathsf{in}\ N^*
\end{array}$$

Lemma 4. *If $\Gamma \vdash M : \sigma$ is derivable in λ_c, so is $\Gamma \vdash M^* : \sigma$ in λ_{c*}.* □

Lemma 5. $M^*[V^*/x] \equiv (M[V/x])^*$. □

Proposition 6. *If $M \succ_1 N$ in λ_c, then $M^* \succ_1 N^*$ in λ_{c*}.*

Proof (sketch). The key cases are

$$\begin{array}{lcl} (P\,M)^* & \equiv & \mathsf{let}\ x\ \mathsf{be}\ P^*\ \mathsf{in}\ (\lambda y.(y\,M)^*)\,x \\ & \overset{\beta.v}{\succ} & \mathsf{let}\ x\ \mathsf{be}\ P^*\ \mathsf{in}\ (x\,M)^* \\ & \equiv & (\mathsf{let}\ x\ \mathsf{be}\ P\ \mathsf{in}\ x\,M)^* \end{array}$$

$$\begin{array}{lcl} (V\,Q)^* & \equiv & \mathsf{let}\ y\ \mathsf{be}\ Q^*\ \mathsf{in}\ (\lambda x.V^*\,x)\,y \\ & \overset{\beta.v \text{ or } \eta.v}{\succ} & \mathsf{let}\ y\ \mathsf{be}\ Q^*\ \mathsf{in}\ V^*\,y \\ & \equiv & (\mathsf{let}\ y\ \mathsf{be}\ Q\ \mathsf{in}\ V\,y)^* \end{array}$$

□

Corollary 2. *λ_{c*} is strongly normalizing if and only if λ_c is strongly normalizing.* □

Remark 4. This embedding $(-)^*$ is inspired from the translation $*_1 : \lambda_c \to \lambda_{c*}$ given by Sabry and Wadler [18], but not quite the same. For $*_1$, the translations of $P\,M$ and $V\,Q$ are simply

$$(P\,M)^* \equiv \mathsf{let}\ x\ \mathsf{be}\ P^*\ \mathsf{in}\ (x\,M)^* \qquad (V\,Q)^* \equiv \mathsf{let}\ y\ \mathsf{be}\ Q^*\ \mathsf{in}\ V^*\,y$$

while our embedding introduces additional redices so that the reduction steps are strictly preserved.

8.3 Embedding λ_{c*} into the Linear Lambda Calculus

Now it is fairly easy to give a reduction-preserving translation $(-)^\diamond$ from λ_{c*} into the linear lambda calculus (the "call-by-value Girard translation"): let $b^\diamond = b$, $(\sigma_1 \to \sigma_2)^\diamond = !\sigma_1^\diamond \multimap !\sigma_2^\diamond$ and

$$\begin{array}{ll} x^\dagger & \equiv x \\ (\lambda x^\sigma.M)^\dagger & \equiv \lambda y^{!\sigma^\diamond}.\mathsf{let}\ !x^{\sigma^\diamond}\ \mathsf{be}\ y\ \mathsf{in}\ M^\diamond \\ & \\ V^\diamond & \equiv\ !V^\dagger \\ (V\,W)^\diamond & \equiv V^\dagger\,W^\diamond \\ (\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ N)^\diamond & \equiv \mathsf{let}\ !x^{\sigma^\diamond}\ \mathsf{be}\ M^\diamond\ \mathsf{in}\ N^\diamond \end{array}$$

Lemma 6 (type soundness). *If $\Gamma \vdash M : \sigma$ is derivable in λ_{c*}, so is $\Gamma^\diamond\ ;\ \emptyset \vdash M^\diamond : !\sigma^\diamond$ in the linear lambda calculus [18,9].* □

Lemma 7 (substitution lemma). $M^\diamond[V^\dagger/x] \equiv (M[V/x])^\diamond$. □

Proposition 7 (preservation of reduction). *If $M \succ N$ in λ_{c*}, then $M^\diamond \succ^+ N^\diamond$ in the linear lambda calculus.* □

Corollary 3 (strong normalization). *λ_c is strongly normalizing.* □

We note that a different proof of this result via the reducibility argument has been given by Lindley and Stark [14].

Remark 5. For reasoning about *commutative effects* like non-termination and non-determinism, it makes sense to add the *commutativity axiom*

$$com \quad \mathsf{let}\ x\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ y\ \mathsf{be}\ M\ \mathsf{in}\ N\ =\ \mathsf{let}\ y\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ x\ \mathsf{be}\ L\ \mathsf{in}\ N$$

We conjecture that our translation also preserves reduction modulo the equivalence relation generated by this *com*.

9 Concluding Remarks

We have given a rather simple-minded rewriting system on the linear lambda calculus which enjoys strong normalization and Church-Rosser property modulo trivial commuting conversion. We hope that this gives a reasonably understandable and feasible tool for reasoning about equivalence of terms in the linear lambda calculus. We shall conclude this paper by a few additional remarks.

9.1 Call-by-Name, Call-by-Value, and the Linear Lambda Calculus

This work can be considered as a refinement of some of the results in [15] where reduction-preserving translations between the (simply typed) call-by-name, call-by-value, call-by-need and linear lambda calculi were discussed. In *ibid.*, weaker non-extensional theories without η-rules were considered. In contrast, here we have studied the semantically complete theories (DILL-based linear lambda calculus and the computational lambda calculus, as well as the simply typed $\beta\eta$-lambda calculus) and the translations into the linear lambda calculus. We conjecture that the CPS translation from the computational lambda calculus into the linear lambda calculus [9] also enjoys good property with respect to the reduction theories.

9.2 Other Connectives

It is natural to ask if this approach would work well for other logical connectives in DILL, i.e., tensor $\otimes$ and unit I. While the tensor does not seem to cause any significant trouble, the unit is really problematic. For example we have $\mathsf{let}\ *\ \mathsf{be}\ M\ \mathsf{in}\ N = \mathsf{let}\ *\ \mathsf{be}\ N\ \mathsf{in}\ M$ and $M \otimes N = N \otimes M$ for any $M, N : I$. For overcoming this problem with unit, perhaps we need to use the η-expansions as considered by Ghani [6].

Acknowledgements. This work was partially supported by the Japanese Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) 17700013.

References

1. Barber, A. (1997) *Linear Type Theories, Semantics and Action Calculi*. PhD Thesis ECS-LFCS-97-371, University of Edinburgh.
2. Barber, A. and Plotkin, G. (1997) Dual intuitionistic linear logic. Manuscript. An earlier version available as Technical Report ECS-LFCS-96-347, LFCS, University of Edinburgh.
3. Barr, M. (1991) $*$-autonomous categories and linear logic. *Math. Struct. Comp. Sci.* **1**, 159–178.
4. Bierman, G.M. (1995) What is a categorical model of intuitionistic linear logic? In *Proc. Typed Lambda Calculi and Applications*, Springer Lecture Notes in Comput. Sci. **902**, pp. 78–93.
5. Crole, R. (1993) *Categories for Types*. Cambridge University Press.
6. Ghani, N. (1996) Adjoint rewriting and the !-type constructor. Manuscript.
7. Girard, J.-Y. (1987) Linear logic. *Theoret. Comp. Sci.* **50**, 1–102.
8. Hasegawa, M. (2000) Girard translation and logical predicates. *J. Funct. Programming* **10**(1), 77–89.
9. Hasegawa, M. (2002) Linearly used effects: monadic and CPS transformations into the linear lambda calculus. In *Proc. 6th Functional and Logic Programming*, Springer Lecture Notes in Comput. Sci. **2441**, pp. 67–182.
10. Hasegawa, M. (2004) Semantics of linear continuation-passing in call-by-name. In *Proc. 7th Functional and Logic Programming*, Springer Lecture Notes in Comput. Sci. **2998**, pp. 229–243.
11. Hasegawa, M. (2005) Classical linear logic of implications. *Math. Struct. Comput. Sci.* **15**(2), 323–342.
12. Huet, G. (1980) Confluent reductions: abstract properties and applications to term rewriting systems. *J. ACM* **27**(4), 797–821.
13. Lambek, J. and Scott, P. (1986) *Introduction to Higher-order Categorical Logic*. Cambridge University Press.
14. Lindley, S. and Stark, I. (2005) Reducibility and $\top\top$-lifting for computation types. In *Proc. Typed Lambda Calculi and Applications*, Springer Lecture Notes in Comput. Sci. **3461**, pp. 262–277.
15. Maraist, J., Odersky, M., Turner, D.N. and Wadler, P. (1995) Call-by-name, call-by-value, call-by-need and the linear lambda calculus. In *Proc. 11th Mathematical Foundations of Programming Semantics*, *Electr. Notes Theor. Comput. Sci.* **1**, pp. 370–392.
16. Mellies, P.-A. (2003) Categorical models of linear logic revisited. To appear in *Theoret. Comp. Sci.*
17. Moggi, E. (1989) Computational lambda-calculus and monads. In *Proc. 4th Annual Symposium on Logic in Computer Science*, pp. 14–23; a different version available as Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.
18. Sabry, A. and Wadler, P. (1997) A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, **19**(6), 916–941.
19. Seely, R.A.G. (1989) Linear logic, $*$-autonomous categories and cofree coalgebras. In *Categories in Computer Science*, *AMS Contemp. Math.* **92**, pp. 371–389.
20. Terese (2003) *Term Rewriting Systems*. Cambridge University Press.