Separating Weakening and Contraction in a Linear Lambda Calculus

John Maraist

Advanced Computing Research Centre
School of Computer and Information Science, University of South Australia
Email, *maraist@cis.unisa.edu.au*; URL, *www.cis.unisa.edu.au/~cisjpm*

Abstract. We present a separated-linear lambda calculus of resource consumption based on a refinement of linear logic which allows separate control of weakening and contraction. The calculus satisfies subject reduction and confluence, and inherits previous results on the relationship of Girard's two translations from minimal intuitionistic logic to linear logic with call-by-name and call-by-value. We construct a hybrid translation from Girard's two which is sound and complete for mapping types and reduction sequences from call-by-need into separated-linear λ. This treatment of call-by-need is more satisfying than in previous work, allowing a contrasting of all three reduction strategies in the manner (for example) that the CPS translations allow for call-by-name and call-by-value.

HOW can we explain the differences between parameter-passing styles? With the continuation-passing style (CPS) transforms [24, 25], one makes the flow of control explicit. Each parameter-passing style is associated with a different CPS transform, which reveals the control-flow decisions implicit in each mechanism. In fact these implicit control decisions explain the difference between the mechanisms quite well, and evaluation of the transformed terms actually becomes independent of any particular parameter-passing mechanism. In this paper, we will also compare different calling mechanisms by mapping them into a common system, but rather than focusing on the flow of control as with the CPS transforms, we will contrast the mechanisms in terms of structural typing operations, which reveal how programs use resources, namely their parameters. In particular, we will use *linear* systems, which allow structural operations only in conjunction with a new modal connective. Moreover, rather than considering linear systems with a single intuitionistic mode, as in previous linear calculi, we will construct as a target of the translations a *separated-linear* lambda calculus motivated by Jacobs' model theory [17], where the two key structural operations of *weakening* and *contraction* are enabled by distinct modal connectives, rather than the same single connective.

This presentation assumes familiarity with the lambda calculus and its simple types [5], intuitionistic logic [27] and the basics of linear logic [13].

Introduction

Typed lambda calculi generally exhibit a Curry-Howard correspondence [12, 15]: a close relationship between their type systems and formal logical frameworks, usually some variety of minimal intuitionistic logic. In such logics, structural inference rules play an important, if often overlooked, role: weakening discards unused assumptions, while contraction duplicates assumptions for multiple uses. Moreover, these rules are the *only* facility for duplicating and discarding assumptions. So at the level of the calculi, the corresponding typing rules are the mechanisms by which we introduce the copying or discarding of terms. A term is discarded when it is substituted for a variable which appears only on the left side of a typing judgement, which occurs only as the result of weakening. Similarly, a term is duplicated when it is substituted for a variable which appears more than once on the right side of a typing judgement, which occurs only through contraction.

In previous work [22], we explored translations into a system based on the linear logic of Girard [13]. In linear logic the ability to weaken or contract is expressed in a formula by a new connective !. Linear implication requires the precedent to be used exactly once in the proof of the consequent, and is written with a new connective $\multimap$. The intuitionistic subset of linear logic is of particular interest to us here, so henceforward when we write "linear logic" we will mean this fragment, and will refer to the entire system of Girard as "full linear logic." Girard described two translations of intuitionistic logic into linear logic, which center around the question of the appropriate image in linear logic for the intuitionistic implication proposition $A \to B$: under his *standard* translation $\circ$, we map intuitionistic implication to $(!A^\circ) \multimap (B^\circ)$, while under his *steadfast* translation $*$ we map to $!(A^* \multimap B^*)$. Girard introduced the transformations to relate formulas in the logics; the extensions of the translations to encompass term reduction became clear later. Both translations preserve typability of terms: The standard translation is sound and complete for mapping general reduction sequences from call-by-name, and the steadfast translation is sound for mapping general reduction sequences from call-by-value [18, 20, 22].

The steadfast translation can also address call-by-need [2, 3, 21]. Call-by-need is a more suitable basis for the analysis of lazy functional languages than call-by-name. Although call-by-need uses a different notion of reduction than call-by-name, their observational equivalence theories are the same for convergence both to constants and to function-based closures. Unlike call-by-name — but like call-by-value and actual implementations of lazy functional languages such as Haskell — call-by-need also has the property that reduction will only duplicate terms which have been reduced to values.

Call-by-need cannot be straightforwardly mapped into the linear lambda calculus. The usual transformation would not be sound for reduction because call-by-need allows any term to be discarded once it is clear that the term will not be used, while in the transformation into the linear calculus, only values are marked as discardable. Hence we previously used an *affine* calculus as the target of the translation from call-by-need [22]. In affine systems, only copying is restricted; any term may be discarded. This arrangement has two major drawbacks: first, like call-by-value, the translation of call-by-need reduction sequences to affine reduction sequences is not complete. This restriction undermines the value of the translation as a tool for reasoning about programs, since reduction of the transformed term may suggest transformations not valid in the original system. A second disadvantage is the fact that we must use separate systems as the target of the translations. Much of the insight into call-by-name and call-by-value provided by the CPS or Girard translations arises from the fact that the distinct calculi are mapped into a single system; the separate target for call-by-need muddles the insights. In fact this problem also arises when translating call-by-need into CPS; Okasaki *et al.* [23] use a target CPS calculus enriched with explicit stores.

In this paper we address these problems through the separate control of weakening and contraction. We present a separated-linear lambda calculus which has *affine* and *relevance* connectives $!^a$ and $!^r$ for enabling weakening and contraction, respectively, rather than a single ! operator enabling both together. The adaptation of Girard's two transformations into this system is simple, and not surprisingly the adapted transformations enjoy the same properties as the respective transformations into the original nonseparated, or *unitary*, linear system. What is more insightful is the treatment the separated system allows of call-by-need.

Call-by-name and call-by-value calculi translate as well as they do into the linear calculus in part because the opportunities for discarding and duplication in these systems are coincident. In call-by-name, discarding and duplication is allowed for — but only for — any function argument; in call-by-value, for any value. Call-by-need, on the other hand, allows any function argument to be discarded, but any value to be duplicated: in the separated-linear calculus we can separately describe enabling the structural operations as suggested by this intuition. We need not resort to an affine system, since the separated calculus includes affine-style reduction. Our hybrid translation $\circledast$ from call-by-need into the separated calculus takes intuitionistic implication $A \to B$ to the separated-linear type $!^r((!^a A^\circledast) \multimap B^\circledast)$, and is sound and complete for all typing judgements and reduction sequences.

Related work. Several approaches to the integration of concepts from linear logic and the original Girard translations into the lambda calculus have been presented, including (but not exhaustively) comments in Girard's original paper on linear logic [13]; the work of Abramsky [1], Barber [4], Benton, Bierman, de Paiva and Hyland [6–9], Braüner [10], della Rocca and Roversi [26], and Wadler [28, 29]; and our previous work [20, 22], which is the basis for the present work. All of these approaches have used a single mode for enabling both weakening and contraction.

In the three *substructural* lambda calculi λI [11], λA and λL [16], any use, explicit or implicit, of (respectively) weakening, contraction or both are simply banned. These systems are not sufficient here: while we do want to make the use of such rules explicit, we do not want to prohibit them altogether. Instead, we will take as target calculi systems whose type systems are related to logics where the use of the structural rules must be explicitly enabled. Based on his earlier work on models for these three substructural calculi [16], Jacobs identified a category of models for a variant of linear logic with separate exponential connectives for each of weakening and contraction; the model includes affine and relevance logic as special cases [17]. Jacobs limited the scope of his study to the model theory, giving only a brief sketch of a possible syntax for the logic and not considering its relation to intuitionistic systems. It would be fair to characterise the calculus we present here as a syntax for an intuitionistic fragment of Jacobs' system. To our knowledge there have been no other previous attempts to allow individual control over weakening and contraction, and none to construct the translation we call $\circledast$ for call-by-need.

Outline. This paper is organised as follows. We begin with a review of the relevant background. We consider three calculi which make no restrictions on the use of weakening and contraction in Section 1.1: the call-by-name, call-by-value and call-by-need systems. We are building on previous work on calculi with a single mode for enabling structural operations, and specifically on the linear and affine lambda calculi which we review in Section 1.2. The core of the present work is Section 2, where we introduce the separated-linear calculus and the three translations into it. We conclude in Section 3 with a discussion of various applications and extensions.

Syntactic domains

Types	A, B	$::= Z \mid A \to B$
Values	V	$::= x \mid \lambda x.\, M$
Terms	L, M, N	$::= V \mid M\ N$

Typing judgements

Axiom

$$\frac{}{\Gamma, x : A \vdash x : A}\ \mathrm{Id}$$

Term structure rules

$$\frac{\Gamma,\, x : A \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to B} \to\text{-I} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B} \to\text{-E}$$

Call-by-name reduction

$$(\beta \to) \quad (\lambda x.\, M)\ N \xrightarrow[\mathsf{Name}]{} M[x := N]$$

Call-by-value reduction

$$(\beta \to_v) \quad (\lambda x.\, M)\ V \xrightarrow[\mathsf{Val}]{} M[x := V]$$

Fig. 1. The call-by-name and call-by-value calculi.

1 Background

1.1 Intuitionistic calculi

Figure 1 presents the details of the call-by-name and call-by-value systems [11, 24], and Figure 2 presents the details of the call-by-need calculus [2, 3, 20, 21]. Here and subsequently, we let L, M, N range over terms with a subset of values over which V, W range. We assume the existence of certain base types, ranged over by Z, which we leave unspecified for now. Types, over which A, B range, are either the base type or a function type. Values are plain variables and lambda abstractions. In Name and Val, terms can be a value or an application; in Need terms may also take the form of a let-binding, which describes the sharing in a term. As usual, weakening and contraction do not appear explicitly in the typing rules, but rather implicitly inside of other rules. Weakening occurs in the Id rule whenever Γ is non-empty, since any assumptions in Γ are clearly not used on the right-hand side. Contraction can occur in the $\to$ -E rule: when a variable is used on both sides of the application, we have an implicit contraction.

We omit the name of reduction relations from under symbols when no confusion will result. All reduction is taken to be compatibly closed under arbitrary contexts, with $\to$ denoting single steps and $\twoheadrightarrow$ for zero or more steps. The properties of these systems have been investigated previously:

Proposition 1. *Reduction in* Name *satisfies subject reduction and is confluent [5, 11].*

Proposition 2. *Reduction in* Val *satisfies subject reduction and is confluent [24].*

Proposition 3. *Reduction in* Need *satisfies subject reduction and is confluent [2, 3, 20, 21].*

Syntactic domains As for Name and Val, plus:

$$\text{Terms} \qquad L, M, N ::= \cdots \mid \text{let } x = M \text{ in } N$$

Typing judgements As for Name and Val, plus:

$$\frac{\Gamma \vdash M : A \qquad \Delta,\, x : A \vdash N : B}{\Gamma,\, \Delta \vdash \text{let } x = M \text{ in } N : B}\ \text{Let}$$

Reduction

(I)	$(\lambda x.\, M)\, N$	$\xrightarrow[\mathsf{Need}]{}$ let $x = N$ in M	
(V)	let $x = V$ in M	$\xrightarrow[\mathsf{Need}]{}$ $M[x := V]$	
(C)	(let $x = L$ in M) N	$\xrightarrow[\mathsf{Need}]{}$ let $x = L$ in $(M\ N)$	
(A)	let $y =$ (let $x = L$ in M) in N	$\xrightarrow[\mathsf{Need}]{}$ let $x = L$ in (let $y = M$ in N)	
(G)	let $x = M$ in N	$\xrightarrow[\mathsf{Need}]{}$ N,	if x is not free in N

Fig. 2. The call-by-need lambda calculus.

1.2 Unitary linear calculi

Figure 3 presents the (unitary) linear lambda calculus Lin. Types here are base types, linear function types or modal types; members of the latter should reduce to an expression prefixed with the ! operator. Terms, aside from the usual variables, abstractions and applications, can be ! prefixed, or be an eliminator to access the body of such a prefixed subterm. This system is discussed in detail by Wadler [29] and in our previous work [22] although we have used a simpler formulation of the typing rules in the manner of Barber's DILL system [4], and adapted Jacobs' notation for typing assumptions [16]: We distinguish linear typing assumptions, on which weakening and contraction may not occur, from intuitionistic assumptions, which allow both, by marking groups of assumptions with $\blacktriangleright_l$ and $\blacktriangleright_i$, respectively. The elegance of such a duality of assumptions has been noted for full linear logic in general by Girard [14], and for the intuitionistic fragment in particular by Barber [4]. Then the implicit occurrences of weakening and contraction are allowed only for assumptions tagged $\blacktriangleright_i$: in the axioms no unused $\blacktriangleright_l$-tagged assumptions are allowed, while in the Elimination rules separate groups of assumptions binding different variables must be used in the two judgements above the bar.

Proposition 4. *Reduction in* Lin *satisfies subject reduction and is confluent [20, 22].*

Figure 3 also presents the standard and steadfast Girard translations for (respectively) call-by-name and call-by-value.

Proposition 5. *The translation* $\circ$ *from* Name *to* Lin *preserves substitution, typing judgements and reduction sequences [20, 22]:*

(i) $(M[x := N])^\circ \equiv M^\circ[x := N^\circ]$.
(ii) $\Gamma \vdash_{\mathsf{Name}} M : A$ *if and only if* $\blacktriangleright_i \Gamma^\circ, \blacktriangleright_l - \vdash_{\mathsf{Lin}} M^\circ : A^\circ$.
(iii) $M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow N$ *if and only if* $M^\circ \xrightarrow[\mathsf{Lin}]{} \!\!\!\!\!\rightarrow N^\circ$.

Syntactic domains

Types	A, B	$::= Z \mid\ !A \mid A \multimap B$
Terms	L, M, N	$::= x \mid\ !M \mid \text{let}\ !x = M\ \text{in}\ N \mid \lambda x.\, M \mid M\ N$

Typing judgements

Axioms

$$\frac{}{{}^{\blacktriangleright_i}\Phi, x : A, {}^{\blacktriangleright_l}- \vdash x : A}\ \text{Id}_i \qquad \frac{}{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}x : A \vdash x : A}\ \text{Id}_l$$

Term structure rules

$$\frac{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}- \vdash M : A}{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}- \vdash\ !M :\ !A}\ \text{!-I} \qquad \frac{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Gamma \vdash M :\ !A \qquad {}^{\blacktriangleright_i}\Phi,\ x : A, {}^{\blacktriangleright_l}\Delta \vdash N : B}{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Gamma,\ \Delta \vdash \text{let}\ !x = M\ \text{in}\ N : B}\ \text{!-E}$$

$$\frac{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Gamma,\ x : A \vdash M : B}{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Gamma \vdash \lambda x.\, M : A \multimap B}\ \multimap\text{-I}$$

$$\frac{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Gamma \vdash M : A \multimap B \qquad {}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Delta \vdash N : A}{{}^{\blacktriangleright_i}\Phi, {}^{\blacktriangleright_l}\Gamma,\ \Delta \vdash M\ N : B}\ \multimap\text{-E}$$

Reduction

$(\beta\text{-}\!\multimap)$	$(\lambda x.\, M)\ N$	$\xrightarrow[\mathsf{Lin}]{} M[x := N]$
$(\beta!)$	$\text{let}\ !x =\ !M\ \text{in}\ N$	$\xrightarrow[\mathsf{Lin}]{} N[x := M]$
$(!\!\multimap)$	$(\text{let}\ !x = L\ \text{in}\ M)\ N$	$\xrightarrow[\mathsf{Lin}]{} \text{let}\ !x = L\ \text{in}\ (M\ N)$
$(!!)$	$\text{let}\ !y = (\text{let}\ !x = L\ \text{in}\ M)\ \text{in}\ N$	$\xrightarrow[\mathsf{Lin}]{} \text{let}\ !x = L\ \text{in}\ (\text{let}\ !y = M\ \text{in}\ N)$

Standard translation

Of types

$$Z^\circ \equiv Z$$
$$(A \to B)^\circ \equiv (!A^\circ) \multimap B^\circ$$

Of terms

$$x^\circ \equiv x$$
$$(\lambda x.\, M)^\circ \equiv \lambda y.\, \text{let}\ !x = y\ \text{in}\ M^\circ$$
$$(M\ N)^\circ \equiv M^\circ\ !N^\circ$$

Of typing contexts

$$(\Gamma, x : A)^\circ \equiv \Gamma^\circ, x : A^\circ$$

Steadfast translation

Of types

$$A^* \equiv\ !A^+$$
$$Z^+ \equiv Z$$
$$(A \to B)^+ \equiv (A^* \multimap B^*)$$

Of terms

$$V^* \equiv\ !V^+$$
$$(M\ N)^* \equiv (\text{let}\ !z = M^*\ \text{in}\ z)\ N^*$$
$$(\text{let}\ x = M\ \text{in}\ N)^* \equiv \text{let}\ !x = M^*\ \text{in}\ N^*$$
$$x^+ \equiv x$$
$$(\lambda x.\, M)^+ \equiv \lambda y.\, \text{let}\ !x = y\ \text{in}\ M^*$$

Of typing contexts

$$(\Gamma, x : A)^* \equiv \Gamma^*, x : A^+$$

Fig. 3. The linear lambda calculus.

Proposition 6. *The translation $*$ from* Val *to* Lin *preserves substitution of values, typing judgements and reduction sequences [20, 22].*

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.
(ii) $\Gamma \vdash_{\mathsf{Val}} M : A$ *if and only if* $\blacktriangleright_i \Gamma^*, \blacktriangleright_l - \vdash_{\mathsf{Lin}} M^* : A^*$*, and*
$\Gamma \vdash_{\mathsf{Val}} V : A$ *if and only if* $\blacktriangleright_i \Gamma^*, \blacktriangleright_l - \vdash_{\mathsf{Lin}} V^+ : A^+$.
(iii) If $M \underset{\mathsf{Val}}{\twoheadrightarrow} N$ *then* $M^* \underset{\mathsf{Lin}}{\twoheadrightarrow} N^*$.

As a counterexample to completeness in Proposition 6.(iii) we take $M \equiv I\ P\ M$ and $N \equiv P\ M$ where I is the identity function $I \equiv \lambda x.\, x$ and P is any non-value. Then clearly $M \not\twoheadrightarrow N$ in Val, but we do have in Lin:

$$
\begin{array}{rcl}
(I\ P\ M)^* & \equiv & (\text{let } !w = (\text{let } !z = !I^+ \text{ in } z)\ P^* \text{ in } w)\ M^* \\
& \xrightarrow[(\beta !)]{} & (\text{let } !w = (\lambda y.\, \text{let } !x = y \text{ in } !x)\ P^* \text{ in } w)\ M^* \\
& \xrightarrow[(\beta \multimap)]{} & (\text{let } !w = (\text{let } !x = P^* \text{ in } !x) \text{ in } w)\ M^* \\
& \xrightarrow[(!!)]{} & (\text{let } !x = P^* \text{ in let } !w = !x \text{ in } w)\ M^* \\
& \xrightarrow[(\beta !)]{} & (\text{let } !x = P^* \text{ in } x)\ M^* \\
& \equiv & (P\ M)^*
\end{array}
$$

So clearly completeness does not hold. The problem of finding an extension of Val for which the translation is complete remains open, and we do not discuss the problem further.

Figure 4 presents the affine lambda calculus Aff. This system is just like Lin except that we allow weakening anywhere, and regulate only contraction with the modal operator. This change manifests itself in two places. In the type system, we no longer have linear assumptions, but rather affine assumptions marked $\blacktriangleright_a$ which allow weakening in the axioms, but not contraction in the Elimination rules. Secondly, we have a new reduction rule (!W), which allows any subterm bound to a variable introduced by implicit weakening in an axiom — that is, one which does not appear free in the body of an eliminator — to be simply discarded without further evaluation, even without an exposed prefix.

Proposition 7. *Reduction in* Aff *satisfies subject reduction and is confluent [20, 22].*

The shift from linear to affine lambda as the target of the steadfast translation corresponds to a shift from a strict to a lazy evaluation semantics in the source calculus. In other words, we map not from call-by-value into affine lambda, but rather from call-by-need.

Proposition 8. *The translation $*$ from* Need *to* Aff *preserves substitution of values, typing judgements and reduction sequences [20, 22].*

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.
(ii) $\Gamma \vdash_{\mathsf{Need}} M : A$ *if and only if* $\blacktriangleright_i \Gamma^*, \blacktriangleright_a - \vdash_{\mathsf{Aff}} M^* : A^*$*, and*
$\Gamma \vdash_{\mathsf{Need}} V : A$ *if and only if* $\blacktriangleright_i \Gamma^*, \blacktriangleright_a - \vdash_{\mathsf{Aff}} V^+ : A^+$.
(iii) If $M \underset{\mathsf{Need}}{\twoheadrightarrow} N$ *then* $M^* \underset{\mathsf{Aff}}{\twoheadrightarrow} N^*$.

Unfortunately, we can make no simple statement about the translation of call-by-need into Lin instead of Aff, since we could not expect to soundly translate both (V) and (G) steps. This discrepancy motivates the finer control obtained by separating weakening from contraction.

Syntactic domains As for Lin.

Typing judgements As for Lin, but changing $\blacktriangleright_l$ to $\blacktriangleright_a$, and with new axioms:

$$\frac{}{\blacktriangleright_i \Phi, x : A, \blacktriangleright_a \Gamma \vdash x : A}\ \mathrm{Id}_i \qquad \frac{}{\blacktriangleright_i \Phi, \blacktriangleright_a \Gamma, x : A \vdash x : A}\ \mathrm{Id}_l$$

Reduction As for Lin, plus the following:

$$(!\mathrm{W}) \qquad \text{let } !x = M \text{ in } N \to N, \qquad \text{if } x \text{ is not free in } N$$

Fig. 4. The affine lambda calculus.

2 The separated-linear lambda calculus

Figure 5 presents the types, terms and main reduction rules of the separated-linear lambda calculus SLin. A type is a base type, a linear function, or one of two modal types. As in Name and Val, basic terms include variables, function abstractions and applications. We have a number of new term forms, however, for the introduction and elimination of the new modal operators. The prefixes $!^a$ and $!^r$ denote terms which one may discard or duplicate, respectively; we refer to each prefix as the *dual* of the other. The corresponding $!^a$- and $!^r$-eliminators strip prefixes from expressions for discarding, or for copying into another term. However, the remaining eliminator has no corresponding prefix, and as such is a bit mysterious until one considers the structure of the typing rules.

Typing judgements Lin divide typing assumptions into two groups, *linear* and *intuitionistic* assumptions. Here, since we want to separately enable weakening and contraction, we will have a total of four groups: in addition to the linear and intuitionistic assumptions marked $\blacktriangleright_l$ and $\blacktriangleright_i$, we have affine assumptions marked $\blacktriangleright_a$ as in Aff, plus *relevant* assumptions marked $\blacktriangleright_r$ whose assumptions admit contraction but not weakening. We let $\mathcal{E}$ and $\mathcal{F}$ range over groups of marked typing assumptions. We refer to a free variable typed by an assumption marked $\blacktriangleright_l$ as a *linear variable*, and similarly for relevant variables and so forth.

We can now motivate the "extra" eliminator structure: we must be able to eliminate variables with each of the four markings, and this eliminator corresponds to the intuitionistic assumptions. We have no intuitionistic prefix because such an amalgamation of the $!^a$ and $!^r$ prefixes is exactly what we are trying to avoid with this calculus. We wish to enable weakening and contraction separately from each other, and so an intuitionistic term is one marked as both weakenable and contractable: that is, of either form $!^a!^r M$ or $!^r!^a M$; the $!^i$-eliminator does not distinguish between the two.

The typing rules are straightforward. We have four axioms for each of the four markings of assumptions, where weakening may occur only for assumption marked $\blacktriangleright_a$ or $\blacktriangleright_i$. As usual, lambda abstractions form linear functions, and so eliminate variables bound in an assumption marked $\blacktriangleright_l$. Introduction of a modal prefix is allowed only if a term contains no free variables on which the operation enabled by the prefix is not allowed. For example, the $!^a$-I rule requires that we have only assumptions marked $\blacktriangleright_i$ and $\blacktriangleright_a$ which admit weakening, and none marked $\blacktriangleright_l$ or $\blacktriangleright_r$ which forbid it. A term under a $!^a$ prefix should be discardable, but if a linear or relevant variable were discarded we

Syntactic domains

$$
\begin{array}{lrl}
\text{Types} & A, B & ::= Z \mid A \multimap B \mid !^a A \mid !^r A \\
\text{Terms} & L, M, N & ::= x \mid \lambda x.\, M \mid M\ N \\
 & & \mid\ \text{let } !^a x = M \text{ in } N \mid !^a M \\
 & & \mid\ \text{let } !^r x = M \text{ in } N \mid !^r M \\
 & & \mid\ \text{let } !^i x = M \text{ in } N
\end{array}
$$

Typing rules

Axioms

$$\frac{}{\blacktriangleright^i \Gamma, x : A, \blacktriangleright^a \Delta \vdash x : A}\ \text{Id}_i \qquad \frac{}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta, x : A \vdash x : A}\ \text{Id}_w$$

$$\frac{}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta, \blacktriangleright^r x : A \vdash x : A}\ \text{Id}_c \qquad \frac{}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta, \blacktriangleright^l x : A \vdash x : A}\ \text{Id}_l$$

Term structure rules

$$\frac{\blacktriangleright^i \Gamma, \blacktriangleright^r \Phi \vdash M : A}{\blacktriangleright^i \Gamma, \blacktriangleright^r \Phi \vdash M : !^r A}\ !^r\text{-I} \qquad \frac{\blacktriangleright^i \Gamma, \blacktriangleright^a \Phi \vdash M : A}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Phi \vdash M : !^a A}\ !^a\text{-I} \qquad \frac{\mathcal{E}, \blacktriangleright^l x : A \vdash M : B}{\mathcal{E} \vdash \lambda x.\, M : A \multimap B}\ \multimap\text{-I}$$

$$\frac{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0 \vdash M :!^r A \qquad \blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_1, \blacktriangleright^r \Phi, x : A, \blacktriangleright^l \Psi_1 \vdash N : B}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0, \Psi_1 \vdash \text{let } !^r x = M \text{ in } N : B}\ !^r\text{-E}$$

$$\frac{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0 \vdash M :!^a A \qquad \blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_1, x : A, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_1 \vdash N : B}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0, \Psi_1 \vdash \text{let } !^a x = M \text{ in } N : B}\ !^a\text{-E}$$

$$\frac{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0 \vdash M : !^a !^r A \qquad \blacktriangleright^i \Gamma, x : A, \blacktriangleright^a \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_1 \vdash N : B}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0, \Psi_1 \vdash \text{let } !^i x = M \text{ in } N : B}\ !^i\text{-E}_1$$

$$\frac{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0 \vdash M : !^r !^a A \qquad \blacktriangleright^i \Gamma, x : A, \blacktriangleright^a \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_1 \vdash N : B}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0, \Psi_1 \vdash \text{let } !^i x = M \text{ in } N : B}\ !^i\text{-E}_2$$

$$\frac{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0 \vdash M : A \multimap B \qquad \blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_1 \vdash N : A}{\blacktriangleright^i \Gamma, \blacktriangleright^a \Delta_0, \Delta_1, \blacktriangleright^r \Phi, \blacktriangleright^l \Psi_0, \Psi_1 \vdash M\ N : B}\ \multimap\text{-E}$$

Fig. 5. Syntax and typing rules for the separated-linear lambda calculus.

would violate the typing assumptions. The Elimination rules allow only assumptions marked $\blacktriangleright^r$ or $\blacktriangleright^i$ to occur more than once above the bar.

The reduction rules for SLin are given in Figure 6. The main reduction rules for SLin are the five β rules and the weakening rule. As usual, we have a β rule whenever an elimination rule occurs immediately below the corresponding introduction rule; we have two such rules for the $!^i$-eliminator since the two prefixes to be eliminated may arrive in either order. The weakening rule allows us to take advantage of the fact that a term prefixed by $!^a$ may be discarded immediately by a $!^i$-eliminator if it is not to be used, without necessarily finding a $!^r$ prefix.

The remaining reduction rules are commuting rules for managing the additional structure. The rôle of the $(!\multimap)$ and $(!!)$ rules are the same as for the rules of the same

β-reduction rules

$$
\begin{array}{lll}
(\beta\multimap) & (\lambda x.\, M)\, N & \xrightarrow[\mathsf{SLin}]{} M[x := N] \\
(\beta!^r) & \text{let } !^r x = !^r M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M] \\
(\beta!^a) & \text{let } !^a x = !^a M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M] \\
(\beta!^i_{cw}) & \text{let } !^i x = !^a !^r M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M] \\
(\beta!^i_{wc}) & \text{let } !^i x = !^r !^a M \text{ in } N & \xrightarrow[\mathsf{SLin}]{} N[x := M]
\end{array}
$$

Weakening rule

$$
(\mathrm{W}!^i) \qquad \text{let } !^i x = !^a M \text{ in } N \xrightarrow[\mathsf{SLin}]{} N,\ x \notin \mathrm{fv}(N)
$$

Commuting conversion rules

$$
\begin{array}{ll}
(!^{\square}\multimap) & (\text{let } !^{\square} x = L \text{ in } M)\, N \xrightarrow[\mathsf{SLin}]{} \text{let } !^{\square} x = L \text{ in } (M\, N) \\
(!^{\square}!^{\triangle}) & \text{let } !^{\square} x = (\text{let } !^{\triangle} y = L \text{ in } M) \text{ in } N \\
 & \qquad \xrightarrow[\mathsf{SLin}]{} \text{let } !^{\triangle} y = L \text{ in } (\text{let } !^{\square} x = M \text{ in } N) \\
 & \text{where } !^{\square} \text{ and } !^{\triangle} \text{ each range over } !^a, !^r, !^i. \\
(!^i !^{\square} !^a) & \text{let } !^i x = !^{\square}(\text{let } !^i y = !^a L \text{ in } M) \text{ in } N \\
 & \qquad \xrightarrow[\mathsf{SLin}]{} \text{let } !^i y = !^a L \text{ in let } !^i x = !^{\square} M \text{ in } N \\
 & \text{where } !^{\square} \text{ ranges over } !^a \text{ and } !^r.
\end{array}
$$

Fig. 6. Reduction in the separated-linear lambda calculus.

name in Lin. In the left-hand sides of the former, a modal elimination rule can block a $(\beta\multimap)$ step by hiding an abstraction in the left subterm; in the left-hand sides of the latter, a modal elimination rule can block a $(\beta!)$ step by hiding a correctly prefixed term. In both cases, the rule expands the scope of the variable bound by the blocking eliminator to allow any blocked reduction at that point. The $(!!!)$ rules extend the $(!!)$ rules to the cases where an amalgamated eliminator "sees" one of the two modal prefixes required for a reduction step, but under that prefix another binding can hide the second prefix. Again, the $(!!!)$ rules simply expand the scope of the inner binding and remove it from a position where it may no longer block the reduction of the outer binding.

2.1 Syntactic properties

The first two substitution lemmas follow from easy structural inductions; subject reduction, by induction on the length of the typing trees for β reduction, plus a simple rearrangement of the trees for the other rules.

Lemma 9. *Typing proofs in* SLin *are well-behaved under substitution.*

1. *Let* $\mathcal{E}, \blacktriangleright_l x : B \vdash M : A$ *and* $\mathcal{F} \vdash N : B$. *Then* $\mathcal{E}, \mathcal{F} \vdash M[x := N] : A$.
2. *Let* $\mathcal{E}, \blacktriangleright_r x : B \vdash M : A$ *and* $\blacktriangleright_i \Gamma, \blacktriangleright_r \Delta \vdash N : B$.
   Then $\mathcal{E}, \blacktriangleright_i \Gamma, \blacktriangleright_r \Delta \vdash M[x := N] : A$.
3. *Let* $\mathcal{E}, \blacktriangleright_a x : B \vdash M : A$ *and* $\blacktriangleright_i \Gamma, \blacktriangleright_a \Delta \vdash N : B$.
   Then $\mathcal{E}, \blacktriangleright_i \Gamma, \blacktriangleright_a \Delta \vdash M[x := N] : A$.
4. *Let* $\mathcal{E}, \blacktriangleright_i x : B \vdash M : A$ *and* $\blacktriangleright_i \Gamma \vdash N : B$. *Then* $\mathcal{E}, \blacktriangleright_i \Gamma \vdash M[x := N] : A$.

Lemma 10. *Let* $L, M, N \in \mathsf{SLin}$ *where* $M \xrightarrow[\mathsf{SLin}]{} N$.
Then $L[x := M] \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\!\rightarrow L[x := N]$ *and* $M[x := L] \xrightarrow[\mathsf{SLin}]{} N[x := L]$.

Proposition 11. SLin *satisfies the subject reduction property.*

The confluence result of this section requires more effort; specifically a notion of *marked* reduction *à la* Barendregt [5]. Marking redexes which are let-bindings presents unique difficulties which we first addressed in the context of the call-by-need system [20, 21]; the same techniques apply naturally to SLin [19, 20].

Proposition 12. SLin *is confluent.*

2.2 Translations into SLin

Figure 7 presents the adaptations of the Girard translations to the separated calculus. The translations behave exactly as described in previous work [13, 22]: the standard translation allows any function's argument to be duplicated or discarded, while the steadfast translation allows any value to be duplicated or discarded.

Separated standard translation

Of types

$$(A \to B)^\circ \equiv (!^r!^a A^\circ) \multimap B^\circ$$
$$Z^\circ \equiv Z$$

Of terms

$$x^\circ \equiv x$$
$$(\lambda x.\, M)^\circ \equiv \lambda y.\, \text{let } !^i x = y \text{ in } M^\circ$$
$$(M\ N)^\circ \equiv M^\circ\ (!^r!^a N^\circ)$$

Of typing environments

$$(\Gamma, x : A)^\circ \equiv \Gamma^\circ, x : A^\circ$$

Separated steadfast translation

Of types

$$A^* \equiv !^r!^a A^+$$
$$(A \to B)^+ \equiv (A^*) \multimap (B^*)$$
$$Z^+ \equiv Z$$

Of terms

$$V^* \equiv !^r!^a V^+$$
$$(M\ N)^* \equiv (\text{let } !^i z = M^* \text{ in } z)\ N^*$$
$$(\text{let } x = M \text{ in } N)^* \equiv \text{let } !^i x = M^* \text{ in } N^*$$
$$x^+ \equiv x$$
$$(\lambda x.\, M)^+ \equiv \lambda y.\, \text{let } !^i x = y \text{ in } M^*$$

Of typing environments

$$(\Gamma, x : A)^* \equiv \Gamma^*, x : A^+$$

Fig. 7. The separated Girard translations into SLin.

Theorem 13. *Let* $M, N \in \mathsf{Name}$.

(i) $(M[x := N])^\circ \equiv M^\circ[x := N^\circ]$.
(ii) $\Gamma \vdash_{\mathsf{Name}} M : A$ *if and only if* $\Gamma^\circ \vdash_{\mathsf{SLin}} M^\circ : A^\circ$.
(iii) $M \xrightarrow[\mathsf{Name}]{} \!\!\!\!\!\rightarrow N$ *if and only if* $M^\circ \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\!\rightarrow N^\circ$.

Theorem 14. *Let* $M, N \in \mathsf{Val}$.

(i) $(M[x := V])^* \equiv M^*[x := V^+]$.
(ii) $\Gamma \vdash_{\mathsf{Val}} M : A$ *if and only if* $\blacktriangleright_i \Gamma^* \vdash_{\mathsf{SLin}} M^* : A^*$*, and* $\Gamma \vdash_{\mathsf{Val}} V : A$ *if and only if* $\blacktriangleright_i \Gamma^* \vdash_{\mathsf{SLin}} V^+ : A^+$.
(iii) If $M \xrightarrow[\mathsf{Val}]{} \!\!\!\!\rightarrow N$ *then* $M^* \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\rightarrow N^*$.

The proofs of these two results as well as the counterexample to completeness in Clause (iii) of Theorem 14 are essentially the same as with the unitary target [22].

Finally, Figure 8 describes the hybrid translation motivated in the introduction. We translate all values to be copyable but not necessarily discardable, and all function arguments to be discardable, but not necessarily copyable. So a call-by-need function mapping the type A to the type B is translated to a copyable linear function from the image of A made explicitly discardable, into the image of B: that is, into the type $!^r((!^a A^{\circledast}) \multimap B^{\circledast})$.

Hybrid translation

Of types

$$A^{\circledast} \equiv\, !^r A^{\oplus}$$
$$(A \to B)^{\oplus} \equiv (!^a A^{\circledast}) \multimap B^{\circledast}$$
$$Z^{\oplus} \equiv Z$$

Of typing environments

$$(\Gamma, x : A)^{\circledast} \;\equiv\; \Gamma^{\circledast}, x : A^{\oplus}$$

Of terms

$$V^{\circledast} \equiv\, !^r V^{\oplus}$$
$$(M\ N)^{\circledast} \equiv (\text{let } !^r z = M^{\circledast} \text{ in } z)\ (!^a N^{\circledast})$$
$$(\text{let } x = M \text{ in } N)^{\circledast} \equiv \text{let } !^i x = !^a M^{\circledast} \text{ in } N^{\circledast}$$
$$x^{\oplus} \equiv x$$
$$(\lambda x.\, M)^{\oplus} \equiv \lambda y.\, \text{let } !^i x = y \text{ in } M^{\circledast}$$

Fig. 8. The hybrid translation from Need into SLin.

To show soundness of $\circledast$ we consider various cases of call-by-need typing and reduction sequences; the analysis is tedious but straightforward. For completeness we construct grammars of terms corresponding to the reduction- and evaluation-closures of the $\circledast$-images of Need-terms, and a right-inverse $\natural$ of $\circledast$, shown in Figure 9. This inverse is sound for mapping typing proofs of mapped terms as well as their reduction sequences, back into Need, which implies completeness of $\circledast$. We enumerate the various cases in the full technical report [19].

Lemma 15. *Let* S, T, U *and so forth be as in Figure 9.*

- $(S[x := U])^{\natural} \equiv S^{\natural}[x := U^{\bowtie}]$, $(P[x := U])^{\bowtie} \equiv P^{\bowtie}[x := U^{\bowtie}]$ *and* $(D[y][x := U])^{\bowtie} \equiv (D^{\bowtie}[y])[x := U^{\bowtie}]$.
- *If* $\mathcal{E} \vdash_{\mathsf{SLin}} S : A$ *then* $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} S^{\natural} : A^{\natural}$*;*
  if $\mathcal{E} \vdash_{\mathsf{SLin}} P : A$ *then* $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} P^{\bowtie} : A^{\bowtie}$*;*
  if $\mathcal{E} \vdash_{\mathsf{SLin}} U : \bar{A}$ *then* $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} U^{\bowtie} : \bar{A}^{\bowtie}$*; and*
  if $\mathcal{E} \vdash_{\mathsf{SLin}} D[x] : \bar{A}$ *then* $\mathcal{E}^{\natural} \vdash_{\mathsf{Need}} D[x]^{\bowtie} : \bar{A}^{\bowtie}$.
- *If* $S \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\rightarrow T$ *then* $S^{\natural} \xrightarrow[\mathsf{Need}]{} \!\!\!\!\rightarrow T^{\natural}$.

Theorem 16. *Let* $M, N \in \mathsf{Need}$.

Reduction-closed images of Need types and terms

$$
\begin{array}{llll}
\text{Types} & A, B & ::= & !^r \bar{A} \\
 & \bar{A}, \bar{B} & ::= & (!^a A) \multimap B \mid Z \\
\text{Terms} & S, T & ::= & !^r U \mid P\ (!^a S) \mid \text{let } !^i x = !^a S \text{ in } T \\
 & & \mid & \text{let } !^r x = S \text{ in } (D[x]\ !^a T)\ ,\ x \notin \text{fv}(D, T) \\
\text{Values} & U & ::= & x \mid \lambda y.\, \text{let } !^i x = y \text{ in } S \\
 & P & ::= & U \mid \text{let } !^i x = !^a S \text{ in } P \mid \text{let } !^r x = S \text{ in } D[x]\ ,\ \ x \notin \text{fv}(D) \\
 & D & ::= & [\,] \mid D\ (!^a S)
\end{array}
$$

The inverse translation

Of types

$$
\begin{aligned}
(!^r \bar{A})^\natural &\equiv \bar{A}^{\bowtie} \\
((!^a A) \multimap B)^{\bowtie} &\equiv A^\natural \to B^\natural \\
Z^{\bowtie} &\equiv Z
\end{aligned}
$$

Of typing environments

$$
(\Gamma, x : \bar{A})^\natural \equiv \Gamma^\natural, x : \bar{A}^{\bowtie}
$$

Of terms

$$
\begin{aligned}
(!^r U)^\natural &\equiv U^\natural \\
(P\ (!^a S))^\natural &\equiv P^{\bowtie}\ S^\natural \\
(\text{let } !^i x = S \text{ in } T)^\natural &\equiv \text{let } x = S^\natural \text{ in } T^\natural \\
(\text{let } !^r x = S \text{ in } (D[x])\ T)^\natural &\equiv (D^{\bowtie}[S^\natural])\ T^\natural \\
x^{\bowtie} &\equiv x \\
(\lambda y.\, \text{let } !^i x = y \text{ in } S)^{\bowtie} &\equiv \lambda x.\, S^\natural \\
(\text{let } !^r x = S \text{ in } D[x])^{\bowtie} &\equiv D^{\bowtie}[S^\natural] \\
(\text{let } !^i x = S \text{ in } P)^{\bowtie} &\equiv \text{let } !^i x = S^\natural \text{ in } P^{\bowtie} \\
[\,]^{\bowtie} &\equiv [\,] \\
(D\ (!^a S))^{\bowtie} &\equiv D^{\bowtie}\ S^\natural
\end{aligned}
$$

Fig. 9. The inverse hybrid translation from SLin to Need.

(i) $(M[x := V])^{\circledast} \equiv M^{\circledast}[x := V^{\oplus}]$.
(ii) $\Gamma \vdash_{\mathsf{Need}} M : A$ *if and only if* $^{\blacktriangleright_i} \Gamma^{\circledast} \vdash_{\mathsf{SLin}} M^{\circledast} : A^{\circledast}$*, and* $\Gamma \vdash_{\mathsf{Need}} V : A$ *if and only if* $^{\blacktriangleright_i} \Gamma^{\circledast} \vdash_{\mathsf{SLin}} V^{\oplus} : A^{\oplus}$.
(iii) $M \xrightarrow[\mathsf{Need}]{} \!\!\!\!\!\rightarrow N$ *if and only if* $M^{\circledast} \xrightarrow[\mathsf{SLin}]{} \!\!\!\!\!\rightarrow N^{\circledast}$.

Why is $\circledast$ complete for general reduction from Need into SLin while $*$ from Val into Lin or from Need into Aff, which produce similar reduction-closed images of terms, are not? The counterexample for $*$ exploits the fact that we cannot distinguish an eliminator introduced by the translation for massaging the left-hand side of an application from an eliminator which is the translation of a let-binding or an abstraction. Under $\circledast$ we do not have this confusion: the former is translated as a $!^r$-eliminator while the latter becomes a $!^i$-eliminator, which can be distinguished in the inverse translation.

3 Conclusion

The separated-linear lambda calculus uses a more refined control of the structural operations on types and terms than previous systems based on linear logic. The additional control allows call-by-need to be incorporated in a much more satisfying manner within the scope of the Girard translations, by a new transformation, into the same system as call-by-name and call-by-value, and in a way which is both sound and complete for reduction. We end this report with a sketch of three extensions of the separated calculus.

Standard reduction. Notions of reduction are frequently equipped with standard reduction strategies: a standard strategy $\mapsto$ for a reduction relation $\rightarrow$ and a subset of *answer* terms (ranged over by $\mathcal{A}$) which has three properties:

1. For all $\mathcal{A}$, $\mathcal{A} \not\mapsto$.
2. If $M \mapsto M_1$ and $M \mapsto M_2$, then $M_1 \equiv M_2$.
3. If $M \twoheadrightarrow \mathcal{A}$ then there exists some answer $\mathcal{A}_0$ such that $M \mapsto\!\!\!\!\rightarrow \mathcal{A}_0$.

Standard strategies for the intuitionistic calculi are well-known [2, 3, 5, 12, 21, 24]. In fact the linear, affine and separated systems also have standard strategies [19, 20], although space prevents us from discussing them here. Each of the translations, including the steadfast translation, are sound and complete for mapping these standard reduction sequences.

Larger logical issues. We have implicitly raised but not answered the question of what the decomposition of ! into $!^a$ and $!^r$ does to the original system of linear logic. A complete answer to that question is clearly beyond the scope of this paper, where we have subordinated the logic to the reduction theory. Nonetheless, we do believe that a single-sided sequent calculus for a separated-linear logic should be straightforward, although we have not yet attempted to verify its relation to the models of Girard [13] or Jacobs [16].

Product, sum and base types. As a basis for programming languages one would usually consider a larger calculus, in particular one with product and sum types. An extension to the standard translation to map both sums and products from a suitable extension of call-by-name into the linear lambda calculus extended with the multiplicative and additive sum connectives (& and $\oplus$, in Girard's formulation) is straightforward, if rather tedious with notation. However, the extension of the steadfast translation is less clear. While the extension for products is merely quite awkward, it is not at all clear how to achieve the extension for sums. One can extend the hybrid translation for sum types in the manner of either original translation. For the standard translation, the results are straightforward, while an attempt in the style of the steadfast translation inherits the difficulties of the original [20]. Basic types of constant values and functions over basic types can be included in the separated-linear system and in the various translations in the same manner as in our previous work on the unitary systems [20, 22].

Acknowledgements. I'd like to thank Martin Odersky, Bruce Thomas, David N. Turner, Philip Wadler, Martin Wehr and a number of anonymous referees for their comments through the development of this work.

References

1. S. Abramsky, Computational interpretations of linear logic, *Th. Comp. Sci.* **111** (1993) 3–57.
2. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky and P. Wadler, A call-by-need lambda calculus, in: *Conf. Rec. POPL'95: 22nd ACM Symp. on Principles of Programming Languages* (ACM Press, San Francisco, California, 1995) 233–246.

3. Z. M. Ariola and M. Felleisen, The call-by-need lambda calculus, *J. Func. Prog.* **7**:3 (1997).
4. A. Barber, Dual intuitionistic linear logic (Technical Report ECS-LFCS-96-347, Laboratory for Foundations of Computer Science, University of Edinburgh, 1996).
5. H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Computer Science* (North-Holland Publishing Company, 1984).
6. P. N. Benton, G. Bierman, V. de Paiva and M. Hyland, Type assignment for intuitionistic linear logic (Technical Report 262, Computing Laboratory, University of Cambridge, 1992).
7. P. N. Benton, Strong normalisation for the linear term calculus, *J. Func. Prog.* **5**:1 (1995) 65–80.
8. G. Bierman, What is a categorical model of intuitionistic linear logic?, in: *Proc. Second Int. Conf. on Typed Lambda Calculus* (Springer-Verlag, Lecture Notes in Computer Science (LNCS) 902, Edinburgh, Scotland, 1995).
9. G. Bierman, *On Intuitionistic Linear Logic* (Doctoral Thesis, Computer Laboratory, University of Cambridge, December 1993; Technical Report 346, 1994).
10. T. Braüner, *An Axiomatic Approach to Adequacy* (Ph.D. thesis, Department of Computer Science, University of Aarhus, 1996, Technical Report BRICS-DS-96-4).
11. A. Church, *The Calculi of Lambda Conversion* (Princeton University Press, 1941).
12. H. B. Curry and R. Feys, *Combinatory Logic*, vol. 1 (North-Holland, Amsterdam, 1958).
13. J.-Y. Girard, Linear logic, *Th. Comp. Sci.* **50** (1987) 1–102.
14. J.-Y. Girard, On the unity of logic, *Annals of Pure and Applied Logic* **59** (1993) 201–217.
15. W. Howard, The formulae-as-types notion of construction, in: *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism* (Academic Press, 1980).
16. B. Jacobs, Semantics of lambda-I and of other substructure lambda calculi, in: *Typed Lambda Calculus and Applications* (Springer-Verlag LNCS 664, 1994) 195-208.
17. B. Jacobs, Semantics of weakening and contraction, *Annals of Pure and Applied Logic* **69** (1994) 73–106.
18. I. Mackie, *The Geometry of Implementation* (Ph.D. Thesis, Imperial College London, 1994).
19. J. Maraist, Separating weakening and contraction in a linear lambda calculus (unabridged) (Technical Report 25/96, Fakultät für Informatik, Universität Karlsruhe, 1996).
20. J. Maraist, *Comparing Reduction Strategies in Resource-Conscious Lambda Calculi* (Doctoral thesis, University of Karlsruhe, 1997).
21. J. Maraist, M. Odersky and P. Wadler, The call-by-need lambda calculus, *J. Func. Prog.*, to appear.
22. J. Maraist, M. Odersky, D. N. Turner and P. Wadler, Call-by-name, call-by-value, call-by-need and the linear lambda calculus, in: *Proc. MFPS'95: Eleventh Conf. on the Mathematical Foundations of Programming Semantics* (Elsevier Publishers, ENTCS 1, New Orleans, 1995; extended version to appear in TCS special issue on MFPS).
23. C. Okasaki, P. Lee and D. Tarditi, Call-by-need and continuation-passing style, *Lisp and Symbolic Computation* **7** (1994) 57–81.
24. G. D. Plotkin, Call-by-name, call-by-value and the λ calculus, *Th. Comp. Sci.* **1** (1975) 125–159.
25. J. C. Reynolds, The discoveries of continuations, *Lisp and Symb. Comp.* **6**:3/4 (1993) 233–248.
26. S. R. della Rocca and L. Roversi, Lambda calculus and intuitionistic linear logic (Manuscript, 1994).
27. M. Ryan and M. Sadler, Valuation systems and consequence relations, in: *Handbook of Logic in Computer Science* (Volume 1, Oxford University Press, 1992).
28. P. Wadler, Linear types can change the world!, in: M. Broy and C. Jones, eds., *Programming Concepts and Methods* (North Holland, Sea of Galilee, Israel, 1990).
29. P. Wadler, A syntax for linear logic, in: *Proc. MFPS'93: Ninth Int. Conf. on the Mathematical Foundations of Programming Semantics* (Springer Verlag, LNCS 802, New Orleans, 1993).