There's no substitute for linear logic

Philip Wadler
University of Glasgow*

Abstract

Surprisingly, there is not a good fit between a syntax for linear logic in the style of Abramsky, and a semantics in the style of Seely. Notably, the Substitution Lemma is valid if and only if $!A$ and $!!A$ are isomorphic in a canonical way. An alternative syntax is proposed, that has striking parallels to Moggi's language for monads. In the old syntax, some terms look like the identity that should not, and vice versa; the new syntax eliminates this awkwardness.

1 Introduction

This paper has two purposes: to show that linear logic has no substitute, and to propose one.

The first part presents a standard syntax and semantics for linear logic, and notes some resulting difficulties. The linear logic is that of Girard [Gir87]. The syntax is based on lambda terms, following in the footsteps of Abramsky [Abr90]: the four rules associated with the 'of course' type, Weakening, Contraction, Dereliction, and Promotion, are each represented by a separate term form. The semantics is based on category theory, following in the footsteps of Seely [See89]: Weakening and Contraction are modelled by a *comonoid*, while Dereliction and Promotion are modelled by a *comonad*.

Surprisingly, when you combine a syntax like Abramsky's with a semantics like Seely's, various problems arise. For one thing, there is a term that *looks* as if it denotes the identity, though it does *not*; and a term that does *not* look as if it denotes the identity, though it *does*. For another, the Substitution Lemma does not hold!

The Substitution Lemma is essential: it guarantees that $((\lambda x.\ b)\ a)$ and $b[a/x]$ have the same meaning. Fortunately, there is a simple fix. A necessary and sufficient condition for The Substitution Lemma to holds is that $!A$ and $!!A$ be isomorphic in a canonical way. This explains the occurrence of similar restrictions in the work of O'Hearn [O'He91] and Filinski [Fil92].

*Author's address: Department of Computing Science, University of Glasgow, G12 8QQ, Scotland. Electronic mail: wadler@dcs.glasgow.ac.uk. Phone: +44 41 330 4966. Fax: +44 41 330 4913.

In *8'th International Workshop on the Mathematical Foundations of Programming Semantics*, Oxford, April 1992.

The second part considers alternate syntaxes for linear logic. There is a simpler alternative syntax for Weakening and Contraction, namely, no syntax at all: just like Exchange, there is no need to indicate these rules explicitly in a term form.

There is also an alternate syntax for Dereliction and Promotion. The new syntax has striking parallels to Moggi's language of monads [Mog89]. Roughly speaking,

$$\textit{monads} : \textit{Moggi} :: \textit{comonads} : \textit{Girard}.$$

The constructs are not exactly dual: Moggi's language requires a monad with a tensorial strength, whereas Girard's language requires a comonad that maps product into tensor product.

The new syntax is a better match to the semantics: a term *looks* like its semantics should be the identity only when its semantics *is* the identity. Intriguingly, similar syntax has arisen in other areas: a program transformation method due to Wadler [Wad88], and a model of graph reduction due to Peyton Jones and Salkild [PS89]. This suggests new possible applications of linear logic, which we are avidly pursuing.

1.1 Background and organisation. The syntactic approach of Abramsky has a long history, including as predecessors Lafont [Laf88] and Holmström [Hol88], and as successors Chirimar, Gurnter, and Riecke [CGR91]. The categorical approach of Seely also has a long history, with a notable successor being the work of Filinski [Fil92], which reconciles the categoric approach with operational intuition.

Linear logic has been applied to control updates in a functional language by by Guzmán and Hudak [GH90] and Wadler [Wad90, Wad91], and in an imperative language by O'Hearn [O'He91].

The remainder of this paper is organised as follows. Section 2 gives the syntax and semantics of a fragment of intuitionistic linear logic containing only 'of course' and function types. Section 3 describes the Substitution Lemma, and gives a necessary and sufficient condition for its validity. Section 4 gives an alternate syntax for Weakening and Contraction. Section 5 gives an alternate syntax for Dereliction and Promotion, analogous to Moggi's syntax for monads.

1.2 Acknowledgements. Andrew Pitts first observed that there may be a syntax for comonads analogous to Moggi's syntax for monads, during a lively public discussion at the LMS meeting on Category Theory in Computer Science, held in Durham in July 1991. It is surprising that linear logic was not mentioned in that discussion, since it retrospect it appears to be the key to comparing the monad and comonad approaches.

For comments on this paper, I am grateful to Samson Abramsky, Andrzej Filinski, Yves Lafont, Eugenio Moggi, Peter O'Hearn, and Robert Seely.

2 Intuitionistic linear logic

This section presents a fragment of intuitionistic linear logic, which for simplicity contains only 'of course' and function types. The logic is presented in Figure 1. We deal first with

Id

$$\frac{}{x : A \vdash x : A} \qquad \frac{}{A \xrightarrow{id} A}$$

Exchange

$$\frac{\Gamma, x : A, y : B, \Delta \vdash c : C}{\Gamma, y : B, x : A, \Delta \vdash c : C} \qquad \frac{\Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{c} C}{\Gamma \otimes B \otimes A \otimes \Delta \simeq \Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{c} C}$$

Weakening

$$\frac{\Delta \vdash e : !A \quad \Gamma \vdash b : B}{\Gamma, \Delta \vdash (\text{let } (\_) = e \text{ in } b) : B} \qquad \frac{\Delta \xrightarrow{e} !A \quad \Gamma \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes e} \Gamma \otimes !A \xrightarrow{id \otimes discard} \Gamma \otimes I \simeq \Gamma \xrightarrow{b} B}$$

Contraction

$$\frac{\Delta \vdash e : !A \quad \Gamma, x : !A, y : !A \vdash b : B}{\Gamma, \Delta \vdash (\text{let } (x@y) = e \text{ in } b) : B} \qquad \frac{\Delta \xrightarrow{e} !A \quad \Gamma \otimes !A \otimes !A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes e} \Gamma \otimes !A \xrightarrow{id \otimes duplicate} \Gamma \otimes !A \otimes !A \xrightarrow{b} B}$$

Dereliction

$$\frac{\Delta \vdash e : !A \quad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash (\text{let } (!x) = e \text{ in } b) : B} \qquad \frac{\Delta \xrightarrow{e} !A \quad \Gamma \otimes A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes e} \Gamma \otimes !A \xrightarrow{id \otimes counit} \Gamma \otimes A \xrightarrow{b} B}$$

Promotion

$$\frac{!\Gamma \vdash a : A}{!\Gamma \vdash (!a) : !A} \qquad \frac{!\Gamma \xrightarrow{a} A}{!\Gamma \xrightarrow{kleisli(a)} !A}$$

$\multimap$-Elimination

$$\frac{\Gamma \vdash f : (A \multimap B) \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash (f\ a) : B} \qquad \frac{\Gamma \xrightarrow{f} (A \multimap B) \quad \Delta \xrightarrow{a} A}{\Gamma \otimes \Delta \xrightarrow{f \otimes a} (A \multimap B) \otimes A \xrightarrow{apply} B}$$

$\multimap$-Introduction

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash (\lambda x : A.\ b) : (A \multimap B)} \qquad \frac{\Gamma \otimes A \xrightarrow{b} B}{\Gamma \xrightarrow{curry(b)} (A \multimap B)}$$

Figure 1: Intuitionistic linear logic

the syntax of typing judgements, which appears on the left of the figure, and then with the categorical semantics, which appears on the right.

2.1 Syntax. The inference rules are expressed using types, terms, assumptions, and judgements.

A type has one of the forms:

X	type variable,
$(!A)$	'of course' type,
$(A \multimap B)$	function type.

Let X, Y, Z range over type variables and A, B, C range over types.

A term has one of the forms:

x	individual variable,
$(\text{let } (\_) = e \text{ in } b)$	Weakening,
$(\text{let } (x@y) = e \text{ in } b)$	Contraction,
$(\text{let } (!x) = e \text{ in } b)$	Dereliction,
$(!a)$	Promotion,
$(f\ \ a)$	$\multimap$-Elimination,
$(\lambda x : A.\ b)$	$\multimap$-Introduction.

Let x, y, z range over individual variables and a, b, c, e, f range over terms.

An assumption has the form $x_1 : A_1, \ldots, x_n : A_n$, where $x_1, \ldots, x_n$ are distinct individual variables. Let Γ, Δ range over assumptions. Write $!\Gamma$ for an assumption of the form $x_1 : !A_1, \ldots, x_n : !A_n$

A judgement has the form $\Gamma \vdash a : A$.

Figure 1 gives rules for identifiers (Id and Exchange), 'of course' type (Weakening, Contraction, Dereliction, and Promotion), and function type ($\multimap$-Elimination and Introduction). Weakening discards a variable of 'of-course' type; Contraction duplicates it; and Dereliction uses it once: these are !-Elimination rules. Promotion raises a term to 'of course' type: it is a !-Introduction rule.

2.2 Semantics. A categorical model of intuitionistic linear logic is specified by the following data.

1. A closed symmetric monoidal category $\mathbf{C}$ with unit object I, tensor $\otimes$, and internal hom $\multimap$: the transpose of $f : \Gamma \otimes A \to B$ is $\mathit{curry}(f) : \Gamma \to (A \multimap B)$, and the counit is $\mathit{apply} : (A \multimap B) \otimes A \to B$.

2. A comonad ! on $\mathbf{C}$: the Kleisli operator of $f : !A \to B$ is $\mathit{kleisli}(f) : !A \to !B$, and the counit is $\mathit{counit} : !A \to A$.

3. Finite products in $\mathbf{C}$ with terminal 1 and product $\times$, with natural isomorphisms $I \simeq !1$ and $(!A) \otimes (!B) \simeq !(A \times B)$.

This induces a comonoid structure on each object $!A$ in $\mathbf{C}$ that is natural in A. It is given by

$$\begin{array}{rcl} !A \xrightarrow{discard} I & = & !A \xrightarrow{!terminal} !1 \simeq I, \\ !A \xrightarrow{duplicate} !A \otimes !A & = & !A \xrightarrow{!diagonal} !(A \times A) \simeq (!A) \otimes (!A), \end{array}$$

where $terminal : A \to 1$ is the unique map to the terminal, and $diagonal : A \to A \times A$ is the diagonal map.

The above definition is derived from Seely [See89]. Usually, Seely's definition is thought of as extending Barr's notion of a $*$-autonomous category [Bar79], but we have no need of a dualising object since we do not deal with negation.

A categorical model is obtained by associating with each type variable an object in $\mathbf{C}$, inducing a map from types to objects. Write A for both a type and its corresponding object. Each context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ possesses a corresponding object $\Gamma = A_1 \otimes \cdots \otimes A_n$; the empty context corresponds to the object I. Finally, each judgement $\Gamma \vdash a : A$ corresponds to an arrow $a : \Gamma \to A$ in $\mathbf{C}$.

The right-hand sides of Figure 1 assigns a semantics to each derivation of a judgement. The only tricky rule is Promotion, which implictly depends on the fact that a context $!\Gamma = x_1 : !A_1, \ldots, x_n : !A_n$ corresponds to two isomorphic objects,

$$!A_1 \otimes \cdots \otimes !A_n \simeq !(A_1 \times \cdots \times A_n).$$

The reading on the left is necessary to match the form of the semantics, while the reading on the right is necessary to apply the Kleisli operator. When the context is empty, the isomorphism $I \simeq !1$ applies.

2.3 Coherence. Since a given judgement may have more than one derivation, we must verify that all possible derivations of a judgement assign it the same semantics. This property is called *coherence*, and its importance was noted by Breazu-Tannen, Coquand, Gunter and Scedrov [BCGS91]. In our case, two derivations of a judgement can differ only in their use of the Exchange rule, since uses of all other rules are encoded in the term, and the fact that $\otimes$ is symmetric and monoidal is sufficient to guarantee coherence.

2.4 Lambda calculus. There is a standard encoding, due to Girard [Gir87], of the usual lambda calculus into our linear language:

$$\begin{array}{lcl} \overline{X} & = & X, \\ \overline{(A \to B)} & = & (!\overline{A}) \multimap \overline{B}, \\ \overline{x} & = & (\text{let } (!z) = x \text{ in } z), \\ \overline{(f\ a)} & = & (\overline{f}\ (!\overline{a})), \\ \overline{(\lambda x : A.\ b)} & = & (\lambda x : !\overline{A}.\ \overline{b}). \end{array}$$

Each variable $x : A$ is replaced by a variable $x : !\overline{A}$, each occurrence of a variable Derelicted, and each argument of an application is Promoted. A fresh variable z is introduced for each occurrence of a variable in the original expression; this serves no purpose except to act as a placeholder for Dereliction.

This mapping induces a standard semantics of λ-calculus in a comonad. For instance, in the category of CPOs one may take $!A$ to be lifting, $\otimes$ to be smash product, and $\multimap$ to be strict functions. Dereliction can be read as evaluation, and Promotion can be read as building a closure. This semantics was observed, independent of the connection to linear logic, by Asperti and Curien.

2.5 Examples. Here are five examples of typing judgements and the corresponding semantics:

$$
\begin{array}{lll}
(1) & x : !A \vdash (!x) : !!A & !A \xrightarrow{kleisli(id)} !!A, \\
(2) & y : !!A \vdash (\text{let } (!z) = y \text{ in } z) : !A & !!A \xrightarrow{counit} !A, \\
(3) & x : !A \vdash (\text{let } (!z) = (!x) \text{ in } z) : !A & !A \xrightarrow{kleisli(id)} !!A \xrightarrow{counit} !A, \\
(4) & y : !!A \vdash (\text{let } (!z) = y \text{ in } (!z)) : !!A & !!A \xrightarrow{counit} !A \xrightarrow{kleisli(id)} !!A, \\
(5) & y : !!A \vdash (!(\text{let } (!z) = y \text{ in } z)) : !!A & !!A \xrightarrow{kleisli(counit)} !!A.
\end{array}
$$

The semantics of terms (3) and (5) are both necessarily the identity, but this is not the case for term (4). This is disturbing, since term (4) looks like it ought to be an identity, while term (5) does not.

3 Substitution and the Cut rule

If logic and computing are seen in each others' company, we might accuse the Cut rule of having played the pander. Cut elimination in logic corresponds to the central notion of reduction in computation.

Here is one way to formulate the Cut rule in natural deduction, introducing a new term syntax, (let $x = a$ in b):

$$
\frac{\Delta \vdash a : A \qquad \Gamma, x : A \vdash b : B}{\Gamma, \Delta \vdash (\text{let } x = a \text{ in } b) : B}
\qquad
\frac{\Delta \xrightarrow{a} A \qquad \Gamma \otimes A \xrightarrow{b} B}{\Gamma \otimes \Delta \xrightarrow{id \otimes a} \Gamma \otimes A \xrightarrow{b} B.}
$$

In order for Γ, Δ to be a sensible composition, a and b must contain no free variables in common, though they may both contain variables with the same "of course" type that are later contracted.

Corresponding to Cut elimination, we would expect that (let $x = a$ in b) could be replaced by $b[a/x]$, the term that results from b by substituting a for each occurrence of x. This is justified if we have that

$$\Gamma \otimes \Delta \xrightarrow{[\![b[a/x]]\!]} B \quad = \quad \Gamma \otimes \Delta \xrightarrow{id \otimes [\![a]\!]} A \otimes \Delta \xrightarrow{[\![b]\!]} B,$$

where now we write $[\![a]\!]$, $[\![b]\!]$, and $[\![b[a/x]]\!]$ for the arrows that give the semantics of the terms a, b, and $b[a/x]$, respectively. This is called the *Substitution Lemma.*

One can think of (let $x = a$ in b) as an abbreviation for the term $((\lambda x : A.\ b)\ a)$, which has the same semantics. The Substitution Lemma is equivalent to the β rule of lambda calculus.

Say that the Substitution Lemma is *applicable* when both the hypotheses and conclusion of the following are valid:

$$\frac{\begin{array}{l}\Delta \vdash a : A \\ \Gamma,\ x : A \vdash b : B\end{array}}{\Gamma,\ \Delta \vdash b[a/x] : B.}$$

This is not always the case. A counter-example:

$$\frac{\begin{array}{l}f : (A \multimap !B),\ x : A \vdash (f\ x) : !B \\ g : !(!B \multimap C),\ y : !B \vdash (!(\text{let } (!h) = g \text{ in } (h\ y))) : !C\end{array}}{g : !(!B \multimap C),\ f : (A \multimap !B),\ x : A \vdash (!(\text{let } (!h) = g \text{ in } (h\ (f\ x)))) : !C.}$$

The hypotheses are valid but the conclusion is not. Promotion does not apply because the assumption list in the last line does not have the form $!\Gamma$ for any Γ.

Inapplicable instances of the Substitution Lemma, like that above, can only arise from substituting into Promotion a term with free variables that do not all have "of course" types. The Substitution Lemma would always apply if our logic were restricted so that when $\Gamma \vdash a : A$, if $A = !A'$ for some A' then $\Gamma = !\Gamma'$ for some Γ'. It is an interesting question to consider what restrictions would guarantee this property. However, for the time being we restrict ourselves to the simpler question of validity for applicable instances of substitution.

Here is a surprise: *even when restricted to applicable instances, the Substitution Lemma does not always hold.* The two terms

$$(!(\text{let } (!z) = y \text{ in } z)) \quad \text{and} \quad (\text{let } x = (\text{let } (!z) = y \text{ in } z) \text{ in } (!x))$$

have, respectively, the semantics

$$!!A \xrightarrow{kleisli(counit)} !!A \quad \text{and} \quad !!A \xrightarrow{counit} !A \xrightarrow{kleisli(id)} !!A.$$

These are *not* equal: the morphism on the left is guaranteed to be the identity, while the one on the right is not. Here the semantics of the terms has been simplified by eliminating various trivial isomorphisms such as $!!A \otimes I \simeq !!A$, but the problem holds regardless.

The above shows it is necessary that $counit; kleisli(id) = id$ if the Substitution Lemma is to hold, and hence that $!A$ and $!!A$ be isomorphic. Some widely-used models of linear logic, such as coherence spaces, violate this condition.

Not only is this condition necessary, it is also sufficient.

3.1 Proposition. If $counit; kleisli(id) = id$ then for all $h : !A \to !B$ and all $f : !B \to C$ we have $h; kleisli(f) = kleisli(h; f)$, and conversely.

Proof: We have

$$\begin{array}{rl} & h; kleisli(f) \\ = & h; kleisli(f); counit; kleisli(id) \\ = & h; f; kleisli(id) \\ = & kleisli(h; f); counit; kleisli(id) \\ = & kleisli(h; f). \end{array}$$

For the converse, take $h = \mathit{counit}$ and $f = \mathit{id}$. $\square$

A corrollary of the above is that if $\mathit{counit}; \mathit{kleisli}(\mathit{id}) = \mathit{id}$ then any arrow $h : !A \to !B$ can be rewritten in the form $\mathit{kleisli}(h; \mathit{counit}) : !A \to !B$, and hence the Kleisli category is isomorphic to a full subcategory.

3.2 Proposition. Assuming $\mathit{counit}; \mathit{kleisli}(\mathit{id}) = \mathit{id}$, the Substitution Lemma is valid when applicable: that is, if $b[a/x]$ is well formed, then $[\![b[a/x]]\!] = (\mathit{id} \otimes [\![a]\!]); [\![b]\!]$.

Proof: By induction on the derivation of $\Gamma,\, x : A \vdash b : B$.

Most of the cases are straightforward. We consider only the most interesting case, that where $b = (!b')$ for some b':

$$\frac{!\Delta \vdash a : !A \qquad !\Gamma,\, x : !A \vdash (!b') : (!B)}{!\Gamma,\, !\Delta \vdash (!b'[a/x]) : (!B)}$$

The assumption of the first judgement must have the form $!\Delta$ in order for $(!b'[a/x])$ to be well-formed. Then we have

$$\begin{array}{cl} & (\mathit{id} \otimes [\![a]\!]); [\![(!b')]\!] \\ = & \text{semantics of Promotion} \\ & (\mathit{id} \otimes [\![a]\!]); \mathit{kleisli}([\![b']\!]) \\ = & \text{previous proposition} \\ & \mathit{kleisli}((\mathit{id} \otimes [\![a]\!]); [\![b']\!]) \\ = & \text{induction hypothesis} \\ & \mathit{kleisli}([\![b'[a/x]]\!]) \\ = & \text{semantics of Promotion} \\ & [\![(!b'[a/x])]\!] \end{array}$$

as required. $\square$

Often the Cut rule is presented with the syntax $b[a/x]$ in place of (let $x = a$ in b); for instance, this is done by Abramsky [Abr90] and O'Hearn [O'He91]. Such a presentation is sensible only if the Substitution Lemma applies. It appears that Abramsky's operational semantics does verify the Substitution Lemma, although his paper contains no statement to this effect. O'Hearn and also Filinski [Fil92] take $!A$ and $!!A$ to be isomorphic, although they don't note that this is required to make substitution valid.

The encoding of lambda calculus into linear logic, as discussed in Section 2, is better behaved with regard to substitution. The Substitution Lemma is always applicable, because all types in the assumption list are 'of course' types; and is always valid, because function arguments are always the result of Promotion.

4 Weakening and contraction

This section proposes an alternative syntax for Weakening and Contraction, namely, no syntax at all. Replace the syntactic construct (let $(\_) = z$ in b) with b and the syntactic

Weakening

$$\frac{\Gamma \vdash b : B}{\Gamma, z : !A \vdash b : B} \qquad \frac{\Gamma \xrightarrow{b} B}{\Gamma \otimes !A \xrightarrow{id \otimes discard} \Gamma \otimes 1 \simeq \Gamma \xrightarrow{b} B}$$

Contraction

$$\frac{\Gamma, x : !A,\ y : !A \vdash b : B}{\Gamma, z : !A \vdash b[z/x, z/y] : B} \qquad \frac{\Gamma \otimes !A \otimes !A \xrightarrow{b} B}{\Gamma \otimes !A \xrightarrow{id \otimes duplicate} \Gamma \otimes !A \otimes !A \xrightarrow{b} B}$$

Dereliction

$$\frac{\Gamma \vdash e : !A}{\Gamma \vdash (\langle e \rangle) : A} \qquad \frac{\Gamma \xrightarrow{e} !A}{\Gamma \xrightarrow{e} !A \xrightarrow{counit} A}$$

Promotion

$$\frac{!\Delta \vdash a : A \qquad \Gamma, x : !A \vdash b : B}{\Gamma, !\Delta \vdash (\text{let } x \Leftarrow a \text{ in } b) : B} \qquad \frac{!\Delta \xrightarrow{a} A \qquad \Gamma \otimes !A \xrightarrow{b} B}{\Gamma \otimes !\Delta \xrightarrow{id \otimes kleisli(a)} \Gamma \otimes !A \xrightarrow{b} B}$$

Figure 2: Alternative structural rules

construct (let $(x@y) = z$ in b) with $b[z/x, z/y]$. The new rules are shown in Figure 2. This figure also shows new rules for Promotion and Dereliction, which are discussed in the next section.

4.1 Coherence. It is necessary to confirm that the new system is coherent. This is guaranteed by the fact that *discard* and *duplicate* form a comonoid. The two distinct terms in the old syntax

$$(\text{let } (x@y) = z \text{ in } (\text{let } (\_) = y \text{ in } b)) \quad \text{and} \quad b[z/x]$$

are both written $b[z/x]$ in the new syntax, and both have the same semantics since

$$duplicate; (id \otimes discard); left = id,$$

where $left : A \otimes I \to A$ is the canonical isomorphism. Similarly, the two distinct terms in the old syntax

$$\begin{array}{l}(\text{let } (x@w) = u \text{ in } (\text{let } (y@z) = w \text{ in } b)) \quad \text{and} \\ (\text{let } (v@z) = u \text{ in } (\text{let } (y@z) = v \text{ in } b))\end{array}$$

are both written $b[u/x, u/y, u/z]$ in the new syntax, and both have the same semantics since

$$duplicate; (duplicate \otimes id); assoc = duplicate; (id \otimes duplicate),$$

where $assoc : (A \otimes B) \otimes C \rightarrow A \otimes (B \otimes C)$ is the canonical isomorphism.

4.2 Example. In the old syntax, the K and W combinators are defined by writing

$$\begin{array}{lcl} K & = & (\lambda x.\,(\lambda y.\,(\mathrm{let}\ (\_) = y\ \mathrm{in}\ x))), \\ W & = & (\lambda f.\,(\lambda z.\,(\mathrm{let}\ (x @ y) = z\ \mathrm{in}\ ((f\ x)\ y)))). \end{array}$$

In the new syntax, one simply writes

$$\begin{array}{lcl} K & = & (\lambda x.\,(\lambda y.\,x)), \\ W & = & (\lambda f.\,(\lambda z.\,((f\ z)\ z))). \end{array}$$

5 Deriliction and Promotion

This section proposes an alternative syntax for Dereliction and Promotion. In the old system, Dereliction is a binding form, while in the new system it is Promotion that is a binding form. The new rules are shown in Figure 2. The new syntax of Deriliction is $\langle a \rangle$ and the new syntax of Promotion is $(\mathrm{let}\ x \Leftarrow a\ \mathrm{in}\ b)$; observe the use of $\Leftarrow$ rather than $=$ in the syntax.

The new syntax may be defined in terms of the old as follows:

$$\begin{array}{lcl} \langle a \rangle & = & (\mathrm{let}\ (!x) = a\ \mathrm{in}\ x), \\ (\mathrm{let}\ x \Leftarrow a\ \mathrm{in}\ b) & = & (\mathrm{let}\ x = (!a)\ \mathrm{in}\ b). \end{array}$$

Conversely, the old syntax may be defined in terms of the new:

$$\begin{array}{lcl} (\mathrm{let}\ (!x) = a\ \mathrm{in}\ b) & = & (\mathrm{let}\ x = \langle a \rangle\ \mathrm{in}\ b), \\ (!a) & = & (\mathrm{let}\ x \Leftarrow a\ \mathrm{in}\ x). \end{array}$$

The syntax verifies the following identities, which are similar to those described by Moggi [Mog89]:

$$\begin{array}{lcl} (\mathrm{let}\ x \Leftarrow a\ \mathrm{in}\ \langle x \rangle) & = & a, \\ (\mathrm{let}\ x \Leftarrow \langle a \rangle\ \mathrm{in}\ b) & = & (\mathrm{let}\ x = a\ \mathrm{in}\ b), \\ (\mathrm{let}\ x \Leftarrow a\ \mathrm{in}\ (\mathrm{let}\ y \Leftarrow b\ \mathrm{in}\ c)) & = & (\mathrm{let}\ y \Leftarrow (\mathrm{let}\ x \Leftarrow a\ \mathrm{in}\ b)\ \mathrm{in}\ c). \end{array}$$

In the middle equation, both sides equal $b[a/x]$ if the Substitution Lemma holds. In the last equation, x may not appear free in c.

This syntax is subject to the problems described in Section 3, and also admits of the same solution: if $\mathit{counit}; \mathit{kleisli}(\mathit{id}) = \mathit{id}$ then the Substitution Lemma is valid when applicable.

5.1 Lambda calculus. In the new syntax, the standard encoding of λ-calculus into linear logic is given by

$$\begin{array}{lcl} \overline{x} & = & \langle x \rangle, \\ \overline{(f\ a)} & = & (\mathrm{let}\ z \Leftarrow \overline{a}\ \mathrm{in}\ (\overline{f}\ z)), \\ \overline{(\lambda x : A.\,b)} & = & (\lambda x : !\overline{A}.\,\overline{b}). \end{array}$$

A fresh variable z is introduced for each application in the original expression. Unlike with the previous encoding, the variable serves a useful purpose. Consider $(\lambda x : A.\,b)$. The old encoding yields $((\lambda x : !\overline{A}.\ \overline{b})\ (!\overline{a}))$, and applying substitution yields $\overline{b}[(!\overline{a})/x]$, which may duplicate the subexpression $\overline{a}$. The new encoding yields (let $z \Leftarrow \overline{a}$ in $((\lambda x : !\overline{A}.\ \overline{b})\ z)$) and applying substitution yields (let $z \Leftarrow \overline{a}$ in $\overline{b}[z/x]$), which does not duplicate $\overline{a}$. Hence the new encoding captures some of the operational intuition of graph reduction: there is no loss of sharing.

5.2 Examples. The examples of Section 2 are now rendered as follows.

$$
\begin{array}{lll}
(1) & x : !A \vdash (\text{let } z \Leftarrow x \text{ in } z) : !!A & !A \xrightarrow{kleisli(id)} !!A, \\
(2) & y : !!A \vdash \langle y \rangle : !A & !!A \xrightarrow{counit} !A, \\
(3) & x : !A \vdash (\text{let } y \Leftarrow x \text{ in } \langle y \rangle) : !A & !A \xrightarrow{kleisli(id)} !!A \xrightarrow{counit} !A, \\
(4) & y : !!A \vdash (\text{let } x = \langle y \rangle \text{ in } (\text{let } z \Leftarrow x \text{ in } z)) : !!A & !!A \xrightarrow{counit} !A \xrightarrow{kleisli(id)} !!A, \\
(5) & y : !!A \vdash (\text{let } z \Leftarrow \langle y \rangle \text{ in } z) : !!A & !!A \xrightarrow{kleisli(counit)} !!A.
\end{array}
$$

Recall that (3) and (5) are both the identity, while (4) is not in general the identity, but will be if the Substitution Lemma holds. In the old syntax, (3) and (4) looked like the identity function, but (5) did not. The new syntax is a better match to the semantics: (3) and (5) look like identities (by the laws of 'let' above) and (4) looks like an identity if you believe in the Substitution Lemma.

Girard's standard encoding of lambda calculus into linear logic takes the lambda calculus term $(f\ \ a)$ into $(f\ \ (!a))$ in the old syntax, and (let $x \Leftarrow a$ in $(f\ \ x)$) in the new syntax. Whereas β-reduction of $(f\ \ (!a))$ may cause the term $(!a)$ to be duplicated, no such problem arises with $(f\ \ x)$. Thus, the new syntax captures an important aspect of sharing. This feature is central to the similar syntax used by Peyton Jones and Salkild, discussed below.

5.3 Applications. One interest of the new syntax is that in practice the 'let' construct has already been used in a similar way.

In work on program transformation, Wadler [Wad88] has used the form (let $x \Leftarrow a$ in b) to indicate that term a is 'unblazed' and thus not subject to further transformation, and the similarity between 'unblazed' types and 'of course' types has already been noted in [Wad90].

In work on compiling functional languages, Peyton Jones and Salkild [PS89] have used the form (let $x \Leftarrow a$ in b) to indicate that x should be bound to a closure representing a, and the similarity of Promotion to the construction of closures has already been noted by Lafont, Holmström, and Abramsky, among others.

This suggests some exciting possible applications of linear logic. Early results are positive. Joint work of Hughes, Launchbury, Peyton Jones, and Wadler on the compilation model of [PS89] suggests that type reconstruction based on a variant of linear logic can discover some places where closures need not be overwritten. We are avidly pursuing this and other areas of application.

References

[Abr90] S. Abramsky, Computational interpretations of linear logic. Preprint, Imperial College, London.

[Bar79] M. Barr, $*$-*Autonomous Categories*. Lecture Notes in Mathematics 752, Springer Verlag, 1979.

[BCGS91] V. Breazu-Tannen, T. Coquand, C. A. Gunter, and A. Scedrov, Inheritance as explicit coercion. *Information and Compututation*, 93:172–221, 1991. (An earlier version appeared in *Symposium on Logic in Computer Science*, IEEE Press, Asilomar, California, June 1989.)

[CGR91] J. Chirimar, C. A. Gunter, and J. G. Riecke. Linear ML. Preprint, Department of Computer and Information Science, University of Pennsylvania, 1991.

[Fil92] A. Filinski, Linear continuations. In *Symposium on Principles of Programming Languages*, ACM Press, Albuquerque, New Mexico, January 1992.

[Gir87] J.-Y. Girard, Linear logic. *Theoretical Computer Science*, **50**:1–102, 1987.

[GH90] J. Guzmán and P. Hudak, Single-threaded polymorphic lambda calculus. In *Symposium on Logic in Computer Science*, IEEE Press, Philadelphia, Pa., June 1990.

[Hol88] S. Holmström, A linear functional language. Draft paper, Chalmers University of Technology, 1988.

[Laf88] Y. Lafont, The linear abstract machine. *Theoretical Computer Science*, **59**:157–180, 1988.

[Mog89] E. Moggi, Computational lambda-calculus and monads. In *4'th IEEE Symposium on Logic in Computer Science*, Asilomar, California, June 1989. (A longer version is available as a technical report from the University of Edinburgh.)

[O'He91] P. W. O'Hearn, Linear logic and interference control. In *Conference on Category Theory and Computer Science*, Paris, September 1991. LNCS, Springer Verlag.

[PS89] S. L. Peyton Jones and J. Salkild, The spineless tagless G-machine. In *4'th ACM Conference on Functional Programming Languages and Computer Architecture*, London, September 1989.

[See89] R. A. G. Seely, Linear logic, $*$-autonomous categories, and cofree coalgebras. In *Categories in Computer Science and Logic*, June 1989. AMS Contemporary Mathematics 92.

[Wad88] P. Wadler, Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73: 231–248, 1990. (Special issue of selected papers from 2'nd European Symposium on Programming, 1988.)

[Wad90] P. Wadler, Linear types can change the world! In *IFIP TC 2 Working Conference on Programming Concepts and Methods*, Sea of Galilee, Israel, April 1990. Published as M. Broy and C. Jones, editors, *Programming Concepts and Methods*, North Holland, 1990.

[Wad91] P. Wadler, Is there a use for linear logic? In *Conference on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, ACM Press, New Haven, Connecticut, June 1991.