A syntax for linear logic*

Philip Wadler (`wadler@dcs.glasgow.ac.uk`)

Department of Computing Science, University of Glasgow, G12 8QQ, Scotland

Abstract. There is a standard syntax for Girard's linear logic, due to Abramsky, and a standard semantics, due to Seely. Alas, the former is incoherent with the latter: different derivations of the same syntax may be assigned different semantics. This paper reviews the standard syntax and semantics, and discusses the problem that arises and a standard approach to its solution. A new solution is proposed, based on ideas taken from Girard's Logic of Unity. The new syntax is based on pattern matching, allowing for concise expression of programs.

1 Introduction

Somewhere inside linear logic, there is a programming language struggling to get out. We wish to define an analogue of lambda calculus to solve the following equation:

$$\frac{\textit{lambda calculus}}{\textit{intuitionistic logic}} = \frac{?}{\textit{linear logic}}$$

What does this language look like?

One would think the answer should be straightforward by now. There is the linear logic of Girard [Gir87], there is the syntax of Abramsky [Abr90], and there is the semantics of Seely [See89]. Each of these has become a standard.

Abramsky was inspired by the earlier work of Lafont [Laf88] and Holmström [Hol88], and in turn inspired related systems by Chirimar, Gunter, and Riecke [CGR92], Lincoln and Mitchell [LM92], Mackie [Mac91], Troelstra [Tro92], and Wadler [Wad90, Wad91].

Seely provided a categorical model, that subsumes other models such as coherence spaces [Gir87], event spaces [Pra91], games [LS91], and the Geometry of Interaction [AJ92].

Unfortunately, Abramsky's syntax is *incoherent* with Seely's semantics: different derivations of the same term may yield different semantics. The basic problem is that Promotion does not commute with substitution. All of the above syntaxes suffer from a similar problem in one form or another, meaning that it is difficult to assign them a meaning in any of the above models. (While the above rightly credits Abramsky's influence, it would be wrong to burden him with too much blame. His syntax is coherent with the operational model he uses.)

---

* In Brookes *et al.* editors, *Mathematical Foundations of Programming Semantics*, New Orleans, April 1993, Springer Verlag LNCS 802.

This difficulty was spotted previously by myself [Wad92]. Other researchers have not only observed the problem, but also proposed a solution in the form of a syntax that 'boxes' the Promotion rule, in much the same way that boxes are used in proof nets. Notable in this regard is the work of Benton, Bierman, de Paiva, and Hyland [BBdPH92], which provides a thorough introduction to natural deduction and sequent versions of linear logic, their categorical semantics, and the associated proof theory.

This paper presents a new syntax for linear logic that resolves the Promotion problem. The new syntax follows naturally from the idea of using patterns in sequents to represent destructors. It is closely related to Girard's Logic of Unity, LU (though without the polarities) [Gir91]. Indeed, the syntax presented here is based on a suggestion from Jean-Yves Girard, who pointed out to me that the problems I had noted with the standard syntax are resolved in the syntax of LU. The syntax also bears a passing resemblance to Moggi's calculus for monads [Mog89].

The syntax has been expressed in a way such that Dereliction and Promotion are made explicit, but Contraction and Weakening are left implicit. Even though linear logic is a 'resource conscious' logic, it seems adequate to be conscious of Dereliction and Promotion alone. The semantics introduces sufficient coherence properties so that the precise order in which Contraction and Weakening is applied is irrelevant. Such details may safely be omitted from the programme, yielding a more economic mode of expression. For those who truly desire to control all the details, a variant syntax that makes Contraction and Weakening explicit is given at the end.

Another approach to giving a syntax for linear logic based on LU appears in more recent work [Wad93]. That paper presents a more tutorial introduction: it is based on natural deduction rather than sequent calculus, so it takes less advantage of pattern matching, and it stresses the syntactic aspects of proof reduction while ignoring the semantics.

The remainder of this paper is organised as follows. Section 2 presents Abramsky's syntax. Section 3 presents Seely's semantics. Section 4 presents the new syntax. Section 5 compares the new syntax with Girard's Logic of Unity. Section 6 sketches some variations on the new syntax.

2 Old syntax

For simplicity, we restrict ourself to the connectives $\otimes$ (tensor product), $\multimap$ (linear implication), & (product), and ! (of course). A *type* (or proposition) is built from these connectives and base types.

$$A, B, C ::= X \mid (A \otimes B) \mid (A \multimap B) \mid (A \,\&\, B) \mid !A$$

Let A, B, C range over types, and X range over base types.

For each of these types, there are *terms* to construct and destruct values of that type.

$$\begin{aligned} t, u ::= {} & x \mid (t, u) \mid (\text{let } (x, y) = t \text{ in } u) \mid (\lambda x.\, t) \mid (t\ u) \mid \\ & \langle t, u \rangle \mid (\text{let } \langle x, \_ \rangle = t \text{ in } u) \mid (\text{let } \langle \_, y \rangle = t \text{ in } u) \mid \\ & !t \mid (\text{let } !x = t \text{ in } u) \mid (\text{let } (x@y) = t \text{ in } u) \mid (\text{let } \_ = t \text{ in } u) \end{aligned}$$

Let t, u range over terms, and f, x, y, z range over variables. The use here of 'let $(x, y) = t$ in u' in comparison with Abramsky's 'let t be $x \otimes y$ in u' merely reflects a preference for the traditional notation, not any significant difference.

An *assumption* has the form $x_1 : A_1, \ldots, x_n : A_n$ where all the variables are distinct, and $n \geq 0$. Let Γ and Δ range over assumptions. Write Γ, Δ for the catenation of two assumptions; whenever this appears it is assumed that the variables of Γ and Δ are disjoint. Finally, a *judgement* has the form $\Gamma \vdash t : A$.

The rules for this version of linear logic are shown in Figure 1. Each rule has zero or more hypotheses above the horizontal line, and a conclusion below. There is one rule for each term form, with the exception of the two rules Exchange and Cut. The Exchange rule expresses that the order of assumptions is irrelevant. The Cut rule uses the notation $u[t/x]$ to stand for the term derived from u by substituting t for all occurrences of x.

The rules are given in sequent calculus style, so constructors are represented by rules (such as $\otimes$-R) where the connective appears in the consequent of the conclusion (to the right of $\vdash$), and destructors are represented by rules (such as $\otimes$-L) where the connective appears in the antaceedent of the conclusion (to the left of $\vdash$). Promotion constructs a term with 'of course' type: it is a !-R rule. Dereliction uses a variable with 'of course' type once, Contraction duplicates it, and Weakening discards it: we refer to these collectively as !-L rules.

The $\multimap$-L rule only allows one to apply a variable to a term. Readers may be more familiar with the application rule of Natural Deduction, which allows one to apply a term to a term.

$$\multimap\text{-E}\ \frac{\Gamma \vdash t : (A \multimap B) \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash (t\ u) : B}$$

This rule is derived as follows.

$$\dfrac{\Gamma \vdash t : (A \multimap B) \qquad \dfrac{\Delta \vdash u : A \qquad \dfrac{}{y : B \vdash y : B}\,\text{Id}}{\Delta, f : (A \multimap B) \vdash (f\ u) : B}\,\multimap\text{-L}}{\Gamma, \Delta \vdash (t\ u) : B}\,\text{Cut}$$

Note the central role played here by Cut. Sequent and natural deduction versions of linear calculus are presented and shown equivalent by Lincoln and Mitchell [LM92]. Various mixtures of the two systems have been used by various researchers [BBdPH92, CGR92, Wad90, Wad91].

$$\text{Id}\ \frac{}{x : A \vdash x : A} \qquad \text{Exchange}\ \frac{\Gamma,\ x : A,\ y : B, \Delta \vdash t : C}{\Gamma,\ y : B,\ x : A, \Delta \vdash t : C}$$

$$\text{Cut}\ \frac{\Gamma \vdash t : A \qquad x : A,\ \Delta \vdash u : B}{\Gamma,\ \Delta \vdash u[t/x] : B}$$

$$\otimes\text{-R}\ \frac{\Gamma \vdash t : A \qquad \Delta \vdash u : B}{\Gamma,\ \Delta \vdash (t, u) : (A \otimes B)} \qquad \otimes\text{-L}\ \frac{\Gamma,\ x : A,\ y : B \vdash t : C}{\Gamma,\ z : (A \otimes B) \vdash (\text{let } (x, y) = z \text{ in } t) : C}$$

$$\multimap\text{-R}\ \frac{\Gamma,\ x : A \vdash t : B}{\Gamma \vdash (\lambda x.\, t) : (A \multimap B)} \qquad \multimap\text{-L}\ \frac{\Gamma \vdash t : A \qquad y : B,\ \Delta \vdash u : C}{\Gamma,\ f : (A \multimap B),\ \Delta \vdash u[(f\, t)/y] : C}$$

$$\&\text{-R}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : (A \,\&\, B)}$$

$$\&\text{-L}\ \frac{\Gamma,\ x : A \vdash t : C}{\Gamma,\ z : (A \,\&\, B) \vdash (\text{let } \langle x, \_ \rangle = z \text{ in } t) : C} \qquad \frac{\Gamma,\ y : B \vdash t : C}{\Gamma,\ z : (A \,\&\, B) \vdash (\text{let } \langle \_, y \rangle = z \text{ in } t) : C}$$

$$\text{Promotion}\ \frac{x_1 : !A_1,\ \ldots,\ x_n : !A_n \vdash t : B}{x_1 : !A_1,\ \ldots,\ x_n : !A_n \vdash !t : !B} \qquad \text{Dereliction}\ \frac{\Gamma,\ x : A \vdash t : B}{\Gamma,\ z : !A \vdash (\text{let } !x = z \text{ in } t) : B}$$

$$\text{Contraction}\ \frac{\Gamma,\ x : !A,\ y : !A \vdash t : B}{\Gamma,\ z : !A \vdash (\text{let } (x@y) = z \text{ in } t) : B} \qquad \text{Weakening}\ \frac{\Gamma \vdash t : B}{\Gamma,\ z : !A \vdash (\text{let } \_ = z \text{ in } t) : B}$$

Fig. 1. Old syntax

Here are a few example judgements.

$$\begin{array}{l}
\vdash (\lambda x.\, \lambda y.\, \text{let } _ = y \text{ in } x) : A \multimap !B \multimap A \\
\vdash (\lambda r.\, \lambda s.\, \lambda x.\, \text{let } !f = r \text{ in let } !g = s \text{ in let } (y@z) = x \text{ in } f\ y\ !(g\ z)) : \\
\qquad !(!A \multimap !B \multimap C) \multimap !(!A \multimap B) \multimap !A \multimap C \\
\vdash (\lambda x.\, \text{let } (y, z) = x \text{ in} \\
\quad !\langle \text{let } !r = y \text{ in let } _ = z \text{ in } r, \text{let } !s = z \text{ in let } _ = x \text{ in } s \rangle) : \\
\qquad (!A \otimes !B) \multimap !(A \,\&\, B)
\end{array}$$

Because of the Cut rule, an unnerving property of this system is that terms do *not* uniquely encode derivations. For example, the judgement

$$z : !!A \vdash !(\text{let } !x = z \text{ in } x) : !!A$$

has the derivation

$$(*) \qquad \cfrac{\cfrac{\cfrac{}{x : !A \vdash x : !A}\ \text{Id}}{z : !!A \vdash (\text{let } !x = z \text{ in } x) : !A}\ \text{Dereliction}}{z : !!A \vdash !(\text{let } !x = z \text{ in } x) : !!A}\ \text{Promotion}$$

and also the derivation

$$(**)\quad \dfrac{\dfrac{\dfrac{}{x : !A \vdash x : !A}\,\mathrm{Id}}{z : !!A \vdash (\mathrm{let}\ !x = z\ \mathrm{in}\ x) : !A}\,\mathrm{Dereliction} \qquad \dfrac{\dfrac{}{y : !A \vdash y : !A}\,\mathrm{Id}}{y : !A \vdash !y : !!A}\,\mathrm{Promotion}}{z : !!A \vdash !(\mathrm{let}\ !x = z\ \mathrm{in}\ x) : !!A.}\,\mathrm{Cut}$$

At first this may seem vaguely disturbing. We shall see shortly that it is profoundly disturbing, because each of these derivations is attached to a *different* semantics.

3 Semantics

This section presents Seely's model of linear logic, restricted to the case of intuitionistic linear logic. Seely's model is normally thought of as deriving from $*$-autonomous categories, but the dualising object $*$ is only required to model classical linear logic.

Anticipating that objects will model types and assumptions, and that arrows will model terms, let A, B, C and Γ, Δ range over objects, and t, u, v range over arrows.

A model of intuitionistic linear logic is provided by a category with the following structure.

- It is symmetric monoidal closed, with unit object $\mathit{1}$, tensor $\otimes$, and internal hom $\multimap$. The transpose of $t : \Gamma \otimes A \to B$ is $\mathit{curry}(t) : \Gamma \to (A \multimap B)$, and the counit is $\mathit{apply} : (A \multimap B) \otimes A \to B$.
- It possesses finite products, with terminal $\top$ and product $\&$. The unique arrow to the terminal is $\langle\rangle : \Gamma \to \top$, the mediating morphism of $t : \Gamma \to A$ and $u : \Gamma \to B$ is $\langle t, u\rangle : \Gamma \to A \& B$, and the projections are $\mathit{fst} : A \& B \to A$ and $\mathit{snd} : A \& B \to B$.
- It possesses a comonad $!$. The Kleisli operator of $t : !A \to B$ is $\mathit{kleisli}(t) : !A \to !B$, and the counit is $\mathit{counit} : !A \to A$.
- There are isomorphisms $\mathit{1} \simeq !\top$ and $!A \otimes !B \simeq !(A \& B)$. These induce a comonoid structure on each object $!A$ that is natural in A, given by

$$!A \xrightarrow{\mathit{discard}} \mathit{1} = !A \xrightarrow{!\langle\rangle} !\top \simeq \mathit{1},$$
$$!A \xrightarrow{\mathit{duplicate}} !A \otimes !A = !A \xrightarrow{!\langle id, id\rangle} !(A \& A) \simeq !A \otimes !A.$$

A categorical model is obtained by associating with each base type an object in our category, inducing a map from types to objects. Write A for both a type and its corresponding object. Each assumption $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ possesses a corresponding object $\Gamma = A_1 \otimes \cdots \otimes A_n$; the empty assumption corresponds to the unit object $\mathit{1}$.

Each judgement $\Gamma \vdash t : A$ corresponds to an arrow $t : \Gamma \to A$. Figure 2 shows how each derivation induces an arrow which is its semantics.

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

$$\text{Id}\ \frac{}{A \xrightarrow{id} A} \qquad \text{Exchange}\ \frac{\Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{t} C}{\Gamma \otimes B \otimes A \otimes \Delta \simeq \Gamma \otimes A \otimes B \otimes \Delta \xrightarrow{t} C}$$

$$\text{Cut}\ \frac{\Gamma \xrightarrow{t} A \qquad A \otimes \Delta \xrightarrow{u} B}{\Gamma \otimes \Delta \xrightarrow{t \otimes id} A \otimes \Delta \xrightarrow{u} B}$$

$$\otimes\text{-R}\ \frac{\Gamma \xrightarrow{t} A \qquad \Delta \xrightarrow{u} B}{\Gamma \otimes \Delta \xrightarrow{t \otimes u} A \otimes B} \qquad \otimes\text{-L}\ \frac{\Gamma \otimes A \otimes B \xrightarrow{t} C}{\Gamma \otimes (A \otimes B) \simeq \Gamma \otimes A \otimes B \xrightarrow{t} C}$$

$$\multimap\text{-R}\ \frac{\Gamma \otimes A \xrightarrow{t} B}{\Gamma \xrightarrow{curry(t)} (A \multimap B)}$$

$$\multimap\text{-L}\ \frac{\Gamma \xrightarrow{t} A \qquad B \otimes \Delta \xrightarrow{u} C}{\Gamma \otimes (A \multimap B) \otimes \Delta \xrightarrow{t \otimes id \otimes id} A \otimes (A \multimap B) \otimes \Delta \simeq (A \multimap B) \otimes A \otimes \Delta \xrightarrow{apply \otimes id} B \otimes \Delta \xrightarrow{u} C}$$

$$\&\text{-R}\ \frac{\Gamma \xrightarrow{t} A \qquad \Gamma \xrightarrow{u} B}{\Gamma \xrightarrow{\langle t,u \rangle} (A \mathbin{\&} B)}$$

$$\&\text{-L}\ \frac{\Gamma \otimes A \xrightarrow{t} C}{\Gamma \otimes (A \mathbin{\&} B) \xrightarrow{id \otimes fst} \Gamma \otimes A \xrightarrow{t} C} \qquad \frac{\Gamma \otimes B \xrightarrow{t} C}{\Gamma \otimes (A \mathbin{\&} B) \xrightarrow{id \otimes snd} \Gamma \otimes B \xrightarrow{t} C}$$

$$\text{Promotion}\ \frac{!A_1 \otimes \cdots \otimes !A_n \simeq !(A_1 \mathbin{\&} \cdots \mathbin{\&} A_n) \xrightarrow{t} B}{!A_1 \otimes \cdots \otimes !A_n \simeq !(A_1 \mathbin{\&} \cdots \mathbin{\&} A_n) \xrightarrow{kleisli(t)} !B}$$

$$\text{Dereliction}\ \frac{\Gamma \otimes A \xrightarrow{t} B}{\Gamma \otimes !A \xrightarrow{id \otimes counit} \Gamma \otimes A \xrightarrow{t} B}$$

$$\text{Contraction}\ \frac{\Gamma \otimes !A \otimes !A \xrightarrow{t} B}{\Gamma \otimes !A \xrightarrow{id \otimes duplicate} \Gamma \otimes (!A \otimes !A) \simeq \Gamma \otimes !A \otimes !A \xrightarrow{t} B}$$

$$\text{Weakening}\ \frac{\Gamma \xrightarrow{t} B}{\Gamma \otimes !A \xrightarrow{id \otimes discard} \Gamma \otimes I \simeq \Gamma \xrightarrow{t} B}$$

Fig. 2. Semantics

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 or Cut rules, since uses of all other rules are encoded in the term. Coherence is guaranteed for Exchange by the fact that $\otimes$ is symmetric monoidal.

Unfortunately, the Cut rule does indeed introduce incoherence, when used in conjunction with Promotion. The derivation $(*)$ given previously induces the semantics

$$(*)\qquad \dfrac{\dfrac{\dfrac{}{!A \xrightarrow{\mathit{id}} !A}\,\mathrm{Id}}{!!A \xrightarrow{\mathit{counit}} !A}\,\mathrm{Dereliction}}{!!A \xrightarrow{\mathit{kleisli}(\mathit{counit})} !!A.}\,\mathrm{Promotion}$$

The derivation $(**)$ given previously induces the semantics

$$(**)\qquad \dfrac{\dfrac{\dfrac{}{!A \xrightarrow{\mathit{id}} !A}\,\mathrm{Id}}{!!A \xrightarrow{\mathit{counit}} !A}\,\mathrm{Dereliction} \qquad \dfrac{\dfrac{}{!A \xrightarrow{\mathit{id}} !A}\,\mathrm{Id}}{!A \xrightarrow{\mathit{kleisli}(\mathit{id})} !!A}\,\mathrm{Promotion}}{!!A \xrightarrow{\mathit{counit}} !A \xrightarrow{\mathit{kleisli}(\mathit{id})} !!A.}\,\mathrm{Cut}$$

These are *not* necessarily equal. The arrow for $(*)$ is necessarily the identity, but the arrow for $(**)$ is not. We thus have the following.

> **Counterexample**. The syntax of Figure 1 is not coherent with the semantics of Figure 2.

This problem arises only with the Promotion rule.

> **Theorem**. The syntax of Figure 1 is coherent with the semantics of Figure 2 if Promotion is not used. If a term does not contain ! as a constructor, then all derivations of it will have the same semantics, even if they use Cut.

The proof is by examination of overlapping rules.

All of the variations of Abramsky's syntax cited above suffer from this problem in one form or another. In a natural deduction system, this problem reveals itself in a failure of the Substitution Lemma: substitution does not commute with Promotion [Wad92]. The same difficulty is at the root of problems that Lincoln and Mitchell [LM92] and Chirimar, Gunter, and Riecke [CGR92] encountered with Subject Reduction theorems, forcing them to be restricted in various ways.

One way to fix the problems is to restrict the class of categorical models. In an earlier paper [Wad92], it was shown that substitution commutes with Promotion if and only if the categorical model satisfies $\mathit{counit}; \mathit{kleisli}(\mathit{id}) = \mathit{id}$. This is not very satisfactory, as none of the models cited at the beginning of this paper satisfy this restriction. Nonetheless, similar restrictions appears in the work of O'Hearn [O'He91] and Filinski [Fil92], and this may explain why.

Another fix is to revise the syntax of Promotion, so that it records explicitly what substitutions have occured. This suggestion has been made by Benton, Bierman, de Paiva, and Hyland [BBdPH92] and by Reddy [Red91]. The syntax of promotion is changed so that the term $!t$ is replaced by $![u_1/x_1, \ldots, u_n/x_n]t$, where $x_1, \ldots, x_n$ are all the free variables of t. Here the square brackets are concrete syntax; this concrete syntax is chosen to resemble the meta-syntax for substitution, since the roles are similar. The revised Promotion rule is as follows.

$$\text{Promotion}' \; \frac{x_1 : !A_1, \; \ldots, \; x_n : !A_n \vdash t : B}{z_1 : !A_1, \; \ldots, \; z_n : !A_n \vdash ![z_1/x_1, \ldots, z_n/x_n]t : B}$$

After promotion, the free variables of the term are $z_1, \ldots, z_n$, and any substitutions for these variables will be explicit in the term. By acting as a barrier to substitution, the new syntax performs much the same role that boxing does in proof nets [Gir87]. It is possible to show that this 'boxed' syntax is coherent: all derivations of a term have the same semantics.

Returning to our example, the first derivation becomes

$$(*) \qquad \cfrac{\cfrac{\cfrac{}{x : !A \vdash x : !A}\,\text{Id}}{y : !!A \vdash (\text{let } !x = y \text{ in } x) : !A}\,\text{Dereliction}}{z : !!A \vdash ![z/y](\text{let } !x = y \text{ in } x) : !!A}\,\text{Promotion}'$$

and the second becomes

$$(**) \qquad \cfrac{\cfrac{\cfrac{}{x : !A \vdash x : !A}\,\text{Id}}{z : !!A \vdash (\text{let } !x = z \text{ in } x) : !A}\,\text{Dereliction} \qquad \cfrac{\cfrac{}{y : !A \vdash y : !A}\,\text{Id}}{w : !A \vdash ![w/y]y : !!A}\,\text{Promotion}'}{z : !!A \vdash ![(\text{let } !x = z \text{ in } x)/y]y : !!A.}\,\text{Cut}$$

Now the terms are different, so it is not a problem that they are assigned different semantics.

The key idea here is that there is a barrier around Promotion indicating what substitutions occur. The next section will reveal a different syntax that erects a similar barrier.

4 New syntax

The new syntax makes three significant changes. First, it introduces a notion of *pattern*. Whereas previously assumptions paired variables with types, now they will pair patterns with types. Second, the various instances of 'let' that appeared previously, associated with the $\otimes$-L, &-L, and !-L rules, are now all consolidated into a single 'let'. Third, there is no explicit indication of Contraction or Weakening in the terms. (This third change is convenient but not essential, and we will see how to undo it in the next section.)

For each type, there is now a *term* to construct values of that type, and a *pattern* to destruct values of that type. The exception is $\multimap$, which has terms for both construction and destruction. There is also a 'let' term.

$$\begin{aligned} p, q &::= x \mid (p, q) \mid \langle p, \_\rangle \mid \langle \_, q\rangle \mid !x \\ t, u &::= x \mid (t, u) \mid (\lambda p.\ t) \mid (t\ u) \mid \langle t, u\rangle \mid !t \mid (\text{let } p = t \text{ in } u) \end{aligned}$$

Let p, q range over patterns, t, u range over terms, and f, x, y, z range over variables. Note that patterns for the types $\otimes$ and & may be nested, but patterns for the type ! may not. We will see below that this system guarantees coherent semantics, but that if nested ! patterns were allowed then coherence would again be lost.

An *assumption* now has the form $p_1 : A_1, \ldots, p_n : A_n$ where $n \geq 0$ and no variable appears more than once in all of the patterns combined. Again, let Γ, Δ range over assumptions, and judgements have the form $\Gamma \vdash t : A$.

The rules for this version of linear logic are shown in Figure 3. With the exception of the new rule Let, there is a one-to-one correspondence between rules in the old syntax and rules in the new syntax. The $\otimes$-L, &-L, and !-L rules now all introduce patterns rather than 'let' terms. The introduction of 'let' terms has been factored out into a separate Let rule. The three !-L rules all introduce the same pattern, so there is no explicit indication of Contraction or Weakening. The appearance of ! patterns in Contraction helps to explain the restriction to variables, since this makes the substitution associated with Contraction easier to express. Promotion is changed so that in addition to requiring that all types in the assumption begin with a !, all patterns in the assumption must also do so.

This last change is the critical step – the ! patterns will act as a barrier to substitution, just as the 'boxed' syntax at the end of the last section did. What was written $![u_1/x_1, \ldots, u_n/x_n]t$ in the boxed syntax is here written

$$\text{let } !y_1 = u_1 \text{ in } \cdots \text{ let } !y_n = u_n \text{ in } t[!y_1/x_1, \ldots, !y_n/x_n].$$

Note that $![u_i/x_i]t$ is concrete syntax, whereas $t[!y_i/x_i]$ is meta-syntax for substitution. Although here the new syntax appears less compact than the boxed syntax, in practice the new syntax will often be more compact because of pattern matching, and because Contraction and Weakening are not explicitly indicated.

The Let rule has no logical content, as erasing the terms from the hypothesis or the conclusion gives the same logical judgement, $\Gamma, A \vdash B$. Indeed, the Let rule can be simply considered a convenient abbreviation, as it can be derived from the $\multimap$ rules and Cut.

$$\dfrac{\dfrac{\Gamma, p : A \vdash u : B}{\Gamma \vdash (\lambda p.\ u) : (A \multimap B)}\ \multimap\text{-R} \qquad \dfrac{\dfrac{}{x : A \vdash x : A}\ \text{Id} \qquad \dfrac{}{y : B \vdash y : B}\ \text{Id}}{f : (A \multimap B),\ x : A \vdash (f\ x) : B}\ \multimap\text{-L}}{\Gamma,\ x : A \vdash ((\lambda p.\ u)\ x) : B}\ \text{Cut}$$

Thus, we can take $(\text{let } p = x \text{ in } u)$ as an abbreviation for $((\lambda p.\ u)\ x)$.

$$\text{Id}\ \frac{}{x : A \vdash x : A} \qquad \text{Exchange}\ \frac{\Gamma,\ p : A,\ q : B, \Delta \vdash t : C}{\Gamma,\ q : B,\ p : A, \Delta \vdash t : C}$$

$$\text{Cut}\ \frac{\Gamma \vdash t : A \qquad x : A,\ \Delta \vdash u : B}{\Gamma,\ \Delta \vdash u[t/x] : B} \qquad \text{Let}\ \frac{\Gamma,\ p : A \vdash u : B}{\Gamma,\ x : A \vdash (\text{let } p = x \text{ in } u) : B}$$

$$\otimes\text{-R}\ \frac{\Gamma \vdash t : A \qquad \Delta \vdash u : B}{\Gamma,\ \Delta \vdash (t, u) : (A \otimes B)} \qquad \otimes\text{-L}\ \frac{\Gamma,\ p : A,\ q : B \vdash t : C}{\Gamma,\ (p, q) : (A \otimes B) \vdash t : C}$$

$$\multimap\text{-R}\ \frac{\Gamma,\ p : A \vdash t : B}{\Gamma \vdash (\lambda p.\ t) : (A \multimap B)} \qquad \multimap\text{-L}\ \frac{\Gamma \vdash t : A \qquad y : B,\ \Delta \vdash u : C}{\Gamma,\ f : (A \multimap B),\ \Delta \vdash u[(f\ t)/y]) : C}$$

$$\&\text{-R}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : (A \mathbin{\&} B)}$$

$$\&\text{-L}\ \frac{\Gamma,\ p : A \vdash t : C}{\Gamma,\ \langle p, \_ \rangle : (A \mathbin{\&} B) \vdash t : C} \qquad \frac{\Gamma,\ q : B \vdash t : C}{\Gamma,\ \langle \_, q \rangle : (A \mathbin{\&} B) \vdash t : C}$$

$$\text{Promotion}\ \frac{!x_1 : !A_1,\ \ldots,\ !x_n : !A_n \vdash t : B}{!x_1 : !A_1,\ \ldots,\ !x_n : !A_n \vdash !t : !B} \qquad \text{Dereliction}\ \frac{\Gamma,\ z : A \vdash t : B}{\Gamma,\ !z : !A \vdash t : B}$$

$$\text{Contraction}\ \frac{\Gamma,\ !x : !A,\ !y : !A \vdash t : B}{\Gamma,\ !z : !A \vdash t[z/x, z/y] : B} \qquad \text{Weakening}\ \frac{\Gamma \vdash t : B}{\Gamma,\ !z : !A \vdash t : B}$$

Fig. 3. New syntax

The rules in Figure 2 for assigning a semantics to the derivation of a term still apply. The Let rule assigns the judgement in the conclusion the same semantics as the judgement in the hypothesis.

> **Theorem**. The syntax of Figure 3 is coherent with the semantics of Figure 2.

The proof is by examining the possible overlaps between rules.

Here are the example judgements of Section 2 revisited.

$$\vdash (\lambda x.\ \lambda !y.\ x) : A \multimap !B \multimap A$$
$$\vdash (\lambda !f.\ \lambda !g.\ \lambda !x.\ f\,!x\,!(g\,!x)) : !(!A \multimap !B \multimap C) \multimap !(!A \multimap B) \multimap !A \multimap C$$
$$\vdash (\lambda(!r, !s).\ !\langle r, s \rangle) : (!A \otimes !B) \multimap !(A \mathbin{\&} B)$$

The new syntax is considerably more compact.

Returning to our main example, the first derivation becomes

$$(*) \qquad \cfrac{\cfrac{\cfrac{}{z : !A \vdash z : !A}\ \text{Id}}{!z : !!A \vdash z : !A}\ \text{Dereliction}}{!z : !!A \vdash !z : !!A.}\ \text{Promotion}$$

The second derivation is no longer valid. The Promotion rule no longer applies, because it contains patterns not in the proper form. In order to obtain the same semantics as previously, the derivation must be rewritten. The old use of the Id rule, which yielded $x : !A \vdash x : !A$, is replaced with a use of Id, Dereliction, and Promotion, which yields $!y : !A \vdash !y : !A$. Both derivations have the same semantics (the identity arrow), but further promotion is only possible for the latter.

$$(**)\qquad \dfrac{\dfrac{\dfrac{}{z : !A \vdash z : !A}\,\text{Id}}{!z : !!A \vdash z : !A}\,\text{Dereliction} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,\text{Id}}{!x : !A \vdash x : A}\,\text{Dereliction}}{!x : !A \vdash !x : !A}\,\text{Promotion}}{!x : !A \vdash !!x : !!A}\,\text{Promotion}}{w : !A \vdash \text{let } !x = w \text{ in } !!x : !!A}\,\text{Let}}{!z : !!A \vdash (\text{let } !x = z \text{ in } !!x) : !!A.}\,\text{Cut}$$

The new $(*)$ and $(**)$ have the same semantics as the old. As with the boxed semantics, we now have distinct terms yielding distinct semantics. Every old derivation carries into a new derivation with the same semantics; the only change needed may be to replace some uses of Id with Id, Dereliction, and Promotion, as above; and to add some uses of Let.

If nested ! patterns were allowed, the coherence property would again be lost. Consider the (illegal) judgement $!!x : !!A \vdash !x : !A$. There are two different proof trees that yield this judgement. The first applies rules in the order Id, Derelict, Promote, Derelict and has semantics $\mathit{counit}; \mathit{kleisli}(\mathit{counit})$, which simplifies to counit. The second applies rules in the order Id, Derelict, Derelict, Promote and has semantics $\mathit{kleisli}(\mathit{counit}; \mathit{counit})$, which does *not* simplify to counit. Hence the restriction that ! patterns cannot be nested. There is no similar problem for $\otimes$ or & patterns.

Since there are no longer explicit terms for Contraction and Weakening, these must be checked for coherence. Coherence here is guaranteed by the fact that *discard* and *duplicate* form a comonoid: duplicating and then discarding is the same as the identity; two duplications in different orders have the same meaning, and so on. The situation is very similar to that for Exchange, and indeed there appears to be no more reason for textually indicating each use of Contraction or Weakening than there is for indicating each use of Exchange.

The new syntax satisfies a pleasing number of equivalences. In the case where the 'let' is simply binding a variable, it can be replaced by substitution. Further, whenever a constructor meets a corresponding destructor, it can be substituted out. Finally, 'let' satisfies a pair of familiar laws. All these points are summarised in the following.

Theorem. The following equations hold for the syntax of Figure 3 with the semantics of Figure 2.

$$
\begin{array}{lrcl}
(1) & (\text{let } x = t \text{ in } u) & = & u[t/x] \\
(2) & (\text{let } (p, q) = (t, u) \text{ in } v) & = & (\text{let } p = t \text{ in } (\text{let } q = u \text{ in } v)) \\
(3) & ((\lambda p.\, u)\, t) & = & (\text{let } p = t \text{ in } u) \\
(4) & (\text{let } \langle p, \_\rangle = \langle t, u\rangle \text{ in } v) & = & (\text{let } p = t \text{ in } v) \\
(5) & (\text{let } \langle \_, q\rangle = \langle t, u\rangle \text{ in } v) & = & (\text{let } q = u \text{ in } v) \\
(6) & (\text{let } !x = !t \text{ in } u) & = & u[t/x] \\
(7) & (\text{let } p = t \text{ in } p) & = & t \\
(8) & (\text{let } q = (\text{let } p = t \text{ in } u) \text{ in } v) & = & (\text{let } p = t \text{ in } (\text{let } q = u \text{ in } v))
\end{array}
$$

These laws assume no collision of bound variables; e.g., in law (2), the free variables of u must not be bound in p.

Law (1) is immediate from coherence. Laws (2)–(6) and (8) follow immediately from the categorical semantics. Law (7) is proved by induction on the pattern.

Here are equations (6)–(8) again, with the last two instantiated to the special case of ! patterns.

$$
\begin{array}{rcl}
(\text{let } !x = !t \text{ in } u) & = & u[t/x] \\
(\text{let } !x = t \text{ in } !x) & = & t \\
(\text{let } !y = (\text{let } !x = t \text{ in } u) \text{ in } v) & = & (\text{let } !x = t \text{ in } (\text{let } !y = u \text{ in } v))
\end{array}
$$

These are reminiscent of the three equations satisfied by Moggi's calculus for monads [Mog89]. For our syntax the first equation depends on the right counit law for comonads and the second equation depends on the left counit law for comonads; while for Moggi's calculus the first equation depends on the left unit law for monads, and the second equation depends on the right unit law for monads. However, the analogy goes awry with the third equation. Moggi's last equation depends on the associative law for monads, while our last equation has nothing to do with the associative law for comonads. (However, the associative laws for comonads is important in verifying the coherence of the new syntax.)

5 Logic of Unity

The system described here is closely related to Girard's Logic of Unity (LU) [Gir91]. Indeed, it was inspired by it: the trick that avoids coherence problems was stolen from LU. To clarify the relation, this section present an appropriately simplified version of LU. Major differences from Girard's LU are that this version is restricted to the intuitionistic fragment, and there are no polarities.

In this variant of LU, there are two sorts of assumptions, linear and intuitionistic. Linear assumptions pair patterns with types, so they have the form $p_1 : A_1, \ldots p_n : A_n$, while intuitionistic assumptions pair variables with types, so they have the form $x_1 : A_1, \ldots x_n : A_n$. Linear assumptions may not be contracted or weakened, while intuitionistic assumptions may. The Contraction rule is much more neatly expressed in terms of variables because it involves substitution, which partly explains the restriction to variables in intuitionistic assumptions. Let Γ, Δ range over linear assumptions, and Φ, Ψ range over intuitionistic assumptions. A judgement has the form $\Gamma;\ \Phi \vdash t : A$, where the linear and intuitionistic assumptions are separated by a semicolon.

The rules for this variant of LU are shown in Figure 4. There is a close correspondence with our new syntax of Figure 3, here called LL for short. The previous Id rule is split into two rules, Id and Id-Int, the first dealing with a linear assumption and the second dealing with an intuitionistic one. Similarly, the previous Exchange rule is split into Exchange and Exchange-Int. The logical rules for $\otimes$ and $\multimap$ deal with linear assumptions. Promotion and Dereliction are logical rules of ! and deal with the relation between the two sorts of assumptions, while Contraction and Weakening have metamorphosed from logical rules of ! to structural rules dealing with intuitionistic assumptions.

$$\text{Id}\ \frac{}{x : A;\ \vdash x : A} \qquad \text{Id-Int}\ \frac{}{;\ x : A \vdash x : A}$$

$$\text{Exchange}\ \frac{\Gamma,\ p : A,\ q : B,\ \Delta;\ \Phi \vdash t : C}{\Gamma,\ q : B,\ p : A,\ \Delta;\ \Phi \vdash t : C} \qquad \text{Exchange-Int}\ \frac{\Gamma;\ \Phi,\ x : A,\ y : B,\ \Psi \vdash t : C}{\Gamma;\ \Phi,\ y : B,\ x : A,\ \Psi \vdash t : C}$$

$$\text{Cut}\ \frac{\Gamma;\ \Phi \vdash t : A \qquad x : A,\ \Delta;\ \Psi \vdash u : B}{\Gamma,\ \Delta;\ \Phi, \Psi \vdash u[t/x] : B} \qquad \text{Let}\ \frac{\Gamma,\ p : A;\ \Phi \vdash u : B}{\Gamma,\ x : A;\ \Phi \vdash (\text{let}\ p = x\ \text{in}\ u) : B}$$

$$\otimes\text{-R}\ \frac{\Gamma;\ \Phi \vdash t : A \qquad \Delta;\ \Psi \vdash u : B}{\Gamma,\ \Delta;\ \Phi, \Psi \vdash (t, u) : (A \otimes B)} \qquad \otimes\text{-L}\ \frac{\Gamma,\ p : A,\ q : B;\ \Phi \vdash t : C}{\Gamma,\ (p, q) : (A \otimes B);\ \Phi \vdash t : C}$$

$$\multimap\text{-R}\ \frac{\Gamma,\ p : A;\ \Phi \vdash t : B}{\Gamma;\ \Phi \vdash (\lambda p.\ t) : (A \multimap B)} \qquad \multimap\text{-L}\ \frac{\Gamma;\ \Phi \vdash t : A \qquad y : B,\ \Delta;\ \Psi \vdash u : C}{\Gamma,\ f : (A \multimap B),\ \Delta;\ \Phi, \Psi \vdash u[(f\ t)/y]) : C}$$

$$\&\text{-R}\ \frac{\Gamma;\ \Phi \vdash t : A \qquad \Gamma;\ \Phi \vdash u : B}{\Gamma;\ \Phi \vdash \langle t, u \rangle : (A \,\&\, B)}$$

$$\&\text{-L}\ \frac{\Gamma,\ p : A;\ \Phi \vdash t : C}{\Gamma,\ \langle p, \_ \rangle : (A \,\&\, B);\ \Phi \vdash t : C} \qquad \frac{\Gamma,\ q : B;\ \Phi \vdash t : C}{\Gamma,\ \langle \_, q \rangle : (A \,\&\, B);\ \Phi \vdash t : C}$$

$$\text{Promotion}\ \frac{;\ \Phi \vdash t : B}{;\ \Phi \vdash !t : !B} \qquad \text{Dereliction}\ \frac{\Gamma,\ !z : !A;\ \Phi \vdash t : B}{\Gamma;\ z : A, \Phi \vdash t : B}$$

$$\text{Contraction}\ \frac{\Gamma;\ \Phi,\ x : A,\ y : A \vdash t : B}{\Gamma;\ \Phi,\ z : A \vdash t[z/x, z/y] : B} \qquad \text{Weakening}\ \frac{\Gamma;\ \Phi \vdash t : B}{\Gamma;\ \Phi,\ z : A \vdash t : B}$$

Fig. 4. A version of the Logic of Unity

It is possible to translate LU into LL. A judgement of the form $\Gamma;\ \Phi \vdash t : A$ in LU corresponds to a judgement $\Gamma, !\Phi \vdash t : A$ in LL, where if Φ is

$x_1 : A_1, \ldots, x_n : A_n$ then $!\Phi$ is $!x_1 : !A_1, \ldots, !x_n : !A_n$.

Each rule in LU corresponds to the rule of the same name in LL, with two spectacular exceptions. Id-Int in LU translates to a combination of Id and Dereliction in LL.

$$\text{Id-Int}\ \frac{}{;\ x : A \vdash x : A} \quad \mapsto \quad \cfrac{\cfrac{}{x : A \vdash x : A}\,\text{Id}}{!x : !A \vdash x : A}\,\text{Dereliction}$$

On the other hand, both the hypothesis and conclusion of the Dereliction rule of LU translate to the same judgement of LL.

$$\text{Dereliction}\ \frac{\Gamma, !z : !A;\ \Phi \vdash t : B}{\Gamma;\ z : A, \Phi \vdash t : B} \quad \mapsto \quad \Gamma, !z : !A, !\Phi \vdash t : B$$

Thus Id-Int in LU corresponds to Dereliction in LL, while Dereliction in LU corresponds to nothing at all!

The translation induces the obvious semantics: the semantics of a judgement in LU is the the same as the semantics of the corresponding judgement in LL. Analogues of the theorems of Section 4 hold.

There are a number of rules which one would expect of LU, which can be derived from the rules given here. The most important of these is Cut-Int.

$$\text{Cut-Int}\ \frac{;\ \Phi \vdash t : A \qquad \Delta;\ x : A, \Psi \vdash u : B}{\Delta;\ \Phi, \Psi \vdash (\text{let } !x = !t \text{ in } u) : B}$$

This rule is derived as follows.

$$\cfrac{\cfrac{;\ \Phi \vdash t : A}{;\ \Phi \vdash !t : !A}\,\text{Promotion} \qquad \cfrac{\cfrac{\Delta;\ x : A, \Psi \vdash u : B}{\Delta, !x : !A;\ \Psi \vdash u : B}\,\text{Dereliction}}{\Delta, y : !A;\ \Psi \vdash \text{let } !x = y \text{ in } u : B}\,\text{Let}}{\Delta;\ \Phi, \Psi \vdash (\text{let } !x = !t \text{ in } u) : B}$$

Observe that the semantics of (let $!x = !t$ in u) is identical to the semantics of $u[t/x]$, which may offer further scope for simplification.

6 Variations

Many programmers are unfamiliar with the $\multimap$-L rule of the sequent calculus, and may find the $\multimap$-E rule of natural deduction more natural. On the other hand, the use of sequent calculus seems to naturally capture the pattern matching in the $\otimes$ and & rules, so there may be some value in exploring a hybrid of the two systems. One variation would simply replace the $\multimap$-L rule by $\multimap$-E. This might be easier for programmers to follow, though important logical properties such as cut-elimination would be lost.

The work presented here extends straightforwardly to handle sums.

$$\oplus\text{-I}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash (\mathrm{inl}\, t) : (A \oplus B)} \qquad \frac{\Gamma \vdash u : B}{\Gamma \vdash (\mathrm{inr}\, u) : (A \oplus B)}$$

$$\oplus\text{-E}\ \frac{\Gamma \vdash z : (A \oplus B) \qquad \Delta,\, p : A \vdash t : C \qquad \Delta,\, q : B \vdash u : C}{\Gamma,\, \Delta \vdash (\mathrm{case}\ z\ \mathrm{of}\, \{\mathrm{inl}\, p \to t; \mathrm{inr}\, q \to u\}) : C}$$

Here we have I and E rules rather than R and L rules. These rules do not exploit the power of pattern matching as thoroughly as one might hope. For instance, patterns of the form $(\mathrm{inl}\, p)$ and $(\mathrm{inr}\, q)$ cannot appear nested inside other patterns. An open question is whether there is a different approach that allows for such nested patterns. One path in this direction is indicated by the work of Breazu-Tannen, Kesner, and Puel [BTKP93].

Another variation is to include patterns to indicate Contraction and Weakening. The grammar of patterns is divided into *patterns* and *of-course patterns*, the former being a superset of the latter.

$$\begin{aligned} p, q &::= x \mid (p, q) \mid \langle p, \_ \rangle \mid \langle \_, q \rangle \mid o \\ o, r &::= (o@r) \mid \_ \mid !x \end{aligned}$$

Let p, q range over patterns, and o, r range over of-course patterns. The new rules are as follows.

$$\text{Promotion}\ \frac{o_1 : !A_1, \ldots, o_n : !A_n \vdash t : B}{o_1 : !A_1, \ldots, o_n : !A_n \vdash !t : !B} \qquad \text{Dereliction}\ \frac{\Gamma,\, z : A \vdash t : B}{\Gamma,\, !z : !A \vdash t : B}$$

$$\text{Contraction}\ \frac{\Gamma,\, o : A,\, r : A \vdash t : B}{\Gamma,\, (o@r) : A \vdash t : B} \qquad \text{Weakening}\ \frac{\Gamma \vdash t : B}{\Gamma,\, \_ : !A \vdash t : B}$$

Dereliction, Contraction, and Weakening introduce the three different sorts of of-course pattern, while Promotion allows any of-course pattern. This variation is included simply to illustrate that the approach used here does not preclude the use of specific patterns to indicate Contraction and Weakening. However, in practice there does not seem to be much value in including such detailed information.

Acknowledgements. I am grateful to Jean-Yves Girard, Samson Abramsky, Robert Seely, Martin Hyland, Valeria de Paiva, and Uday Reddy for their insights into linear logic. One of the MFPS referees made detailed and valuable suggestions for improvement.

References

[Abr90] S. Abramsky, Computational interpretations of linear logic. Presented at *Workshop on Mathematical Foundations of Programming Language Semantics*, 1990. To appear in *Theoretical Computer Science*.

[AJ92] S. Abramsky and R. Jagadeesan, New foundations for the geometry of interaction. In *7'th Symposium on Logic in Computer Science*, IEEE Press, Santa Cruz, California, June 1992.

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

[BBdPH92] N. Benton, G. Bierman, V. de Paiva, and M. Hyland, Type assignment for intuitionistic linear logic. Technical report 262, Computing Laboratory, University of Cambridge, August 1992.

[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.)

[BTKP93] V. Breazu-Tannen, D. Kesner, L. Puel, A typed pattern calculus. In *8'th Symposium on Logic in Computer Science*, Montreal, June 1993.

[CGR92] J. Chirimar, C. A. Gunter, and J. G. Riecke. Linear ML. In *Symposium on Lisp and Functional Programming*, ACM Press, San Francisco, June 1992.

[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.

[Gir91] J.-Y. Girard, On the unity of logic. Manuscript, 1991.

[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.

[LM92] P. Lincoln and J. Mitchell, Operational aspects of linear lambda calculus. In *7'th Symposium on Logic in Computer Science*, IEEE Press, Santa Cruz, California, June 1992.

[LS91] Y. Lafont and T. Streicher. Game semantics for linear logic. In *6'th Symposium on Logic in Computer Science*, IEEE Press, Amsterdam, July 1991.

[Mac91] I. Mackie, Lilac: a functional programming language based on linear logic. Master's Thesis, Imperial College London, 1991.

[Mog89] E. Moggi, Computational lambda-calculus and monads. In *4'th Symposium on Logic in Computer Science*, IEEE Press, Asilomar, California, June 1989.

[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.

[Pra91] V. Pratt, Event spaces and their linear logic. In *AMAST '91: Algebraic Methodology And Software Technology*, Iowa City, Springer Verlag LNCS, 1992.

[Red91] U. Reddy, Acceptors as Values. Manuscript, December 1991.

[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.

[Tro92] A. S. Troelstra, *Lectures on Linear Logic*. CSLI Lecture Notes, 1992.

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

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

[Wad92] P. Wadler, There's no substitute for linear logic. Presented at *Workshop on Mathematical Foundations of Programming Language Semantics*, Oxford, April 1992.

[Wad93] P. Wadler, A taste of linear logic. In *Mathematical Foundations of Computer Science*, Gdansk, Poland, LNCS, Springer Verlag, August 1993.