Abstract The understanding of continuation-passing style (CPS) translations, an historical source of denotational semantics for programming languages, benefits from notions brought by linear logic [Gir91], such as focalisation, polarities and the involutive negation. Here we aim to show how linear logic helps as well when continuations are delimited, i.e. return and can be composed, in the sense of Danvy and Filinski [DF89].

First we provide a polarised calculus with delimited control (first-class delimited continuations) which is, at the level of computation, a variant of Girard’s polarised classical logic LC [Gir91]. It contains variants of delimited control calculi which came as answers to the question “what order of evaluation can we consider with delimited control?” Thus our polarised calculus is one answer which is unifying to some degree.

Subsequently we decompose the calculus through polarised linear logic. The only difference with non-delimited continuations is the use of specific exponentials, that account for the specific semantics of the target of delimited CPS translation, as well as for annotations of type-and-effect systems.

As a by-product, we obtain an explanation of CPS translations through a factoring, each step of which accounts for distinct phenomena of CPS translations. Although the factoring also holds for non-delimited CPS translations, it did not appear in its entirety before.

Contents

Introduction 1

1 A polarised calculus for delimited control 3
   1.1 Understanding LLP 3
   1.2 CPS translation 6
   1.3 A type-and-effect system 6
   1.4 Call-by-value delimited control 7
   1.5 Call-by-name delimited control 7
   1.6 The decomposition of type-and-effect systems 8

2 LLP with annotated exponentials 9
   2.1 From LLP to λµv 9
   2.2 From LLP to LLP 9
   2.3 Translation into linear logic 11

3 Comparison with other works 11
   3.1 The polarised analysis of CPS translations 11
   3.2 Polarised linear logic and the study of double negation translations 12
   3.3 Polarities and delimited control 12
   3.4 The “subtractive” decomposition of delimited continuations 13

4 Perspectives 13

Introduction

Continuation-passing style (CPS) translations are a paradox in the theory of programming languages, because they show that the λ calculus, this purely functional, idealised paradigm of computation, can be diverted so as to seemingly contradict the purely functional paradigm:

1. CPS translations give firm grounds for the notion of evaluation order, by “hard-coding” the order in the structure of the resulting term (Plotkin [Plo75]).
2. They give semantics of non-functional operators like call/cc.
3. Using continuation-passing it is possible to manipulate the output of the computation, not only the input. This highlights the possible symmetry between the two notions (see e.g. Curien and Herbelin [CH00]), and thus further departs from the functional point of view.

Understanding which abstract structures are involved in these translations is fruitful: the call-by-value CPS translation for instance inspired no less than the use of strong monads to describe the semantics of effects in call-by-value programming languages ( Moggi [Mog99]).

Monads however are not sufficient to describe all the existing CPS semantics, as they do not account for the symmetry inherent to CPS: for instance, the call-by-name CPS translation of Lafont, Reus and Streicher [LRS93] is categorically dual to the call-by-value one (Selinger [Sel01]) — which means that if we see computation from an external point of view, the dynamics of expressions in the one coincide with the dynamics of evaluation contexts in the other. Therefore, while some CPS semantics account for monadic constructions, some others account for comonadic ones.

Polarisation

A single structure which accounts for both of these dual constructions yet exists in proof theory, under the form of Girard’s polarised classical logic (LC) [Gir91]. LC was invented to provide a construction of classical logic that would retain the symmetries of classical logic — at the cost of relaxing other properties such as the associativity of composition [LQdF05]. The most important symmetry kept is the involutivity of negation, by which ¬¬A is not only equivalent to A, but isomorphic to it.

Here we are concerned about understanding how notions such as polarities and the involutive negation of LC, once endowed with the proper computational interpretation, give a better view of what we can encode with CPS translations.

To do so, we show how CPS translations can be factored through LC, and isolate the three above aspects in successive steps.

1 Laboratoire PPS, Université Paris Diderot. April 20th, 2011.
Step 1) explains strictness (as in call-by-value) and laziness (or demand-driven-ness, as in call-by-name rather than call-by-need) of a λ calculus with control operators (such as Parigot's μ [Par92]) as the result of a choice, for the constructs of a language, of polarities of LC. Conversely this allows us to see LC through Curry-Howard as a programming language with (call/cc-related) effects that mixes demand-driven computations and data-driven computations (an interpretation that we introduced in [Mun09]).

Step 2) is the standard translation from LC to Laurent's so-called Polarised Linear Logic (LLP) [Lau92]. We show how at the level of the computational interpretation it relates the polarities of LC to the ones of LLP. While in LC expressions of both polarities may carry effects, in LLP they take a lower-level form: positives denote primitive elements of data (such as values), while negatives are defined by how they react to such data. These primitive ingredients yield demand-driven and data-driven computations in a construction that generalises in a symmetric fashion the monadic semantics of call-by-value — the latter only accounting for the positive part of LC.

Step 3) translates the intuitionistic-looking LLP into the lambda-calculus. It introduces specificities of CPS translations that seem orthogonal to their understanding: “administrative” redexes appear, and a gross simplification is introduced on the structure of the polarity-changing negation. This has the consequence of blurring, from the point of view of CPS, the distinction between terms and evaluation contexts that LLP and LC can describe.

But we are also concerned with generalising polarisation to broader forms of effects, and to this effect we give a similar decomposition for delimited CPS translations.

**Delimited CPS translations**

Danvy and Filinski introduced in [DF99] the shift (S) and reset ([:]) control operators in call-by-value λ calculus. The shift operator reifies the current evaluation context under the form of a continuation, and passes it to its argument, like the call/cc control operator. However, unlike call/cc, it only captures the context up to the nearest reset operator – this is why we speak of delimited continuations. Beside standard call-by-value reductions, it has the following reductions:

\[
[E(S(\lambda f.t))] \to [(\lambda f.t)(\lambda x.E(x))] \\
[V] \to V
\]

Amongst other proposals of calculi with composable first-class continuations, the one of Danvy and Filinski met practical and theoretical success.

To name a few, a result of Filinski [Fil94] showed the considerable expressiveness of these operators: any monadic effect that is expressible — that is, whose laws can be expressed as pure functional terms — is available in direct-style through the use of shift and reset. For instance, with the following:

\[
\begin{align*}
\text{read} & \equiv \lambda i. S(\lambda k. \lambda s.([k]()s)) \\
\text{write} & \equiv \lambda s. S(\lambda k. \lambda s.([k]()s))
\end{align*}
\]

[\tau] V computes like \( t \) in which a memory cell has been initialised to the value \( v \). (We adapted this example from [HG08].)

Delimited control operators have also been used to model linguistic phenomena (see Shan [Sha04] for an introduction). And in logic itself, these operators have been used (in a limited fashion) to internalise Markov's principle into intuitionistic logic (Herbelin [Her10]).

Danvy and Filinski describe a denotational semantics under the form of a CPS translation. In this translation, continuations may return and be composed. As a consequence, contrarily to usual CPS translations, the order of evaluation of the target matters, and Danvy and Filinski chose a target in call-by-value. This translation is well understood from the point of view of monads — and the particular type system Danvy and Filinski provide on top of their calculus itself can be described by “annotated” monads (Wadler [Wad94]).

This monadic analysis of the translation and the type system is paralleled by our polarised analysis; and, as far as the positive (call-by-value) part is concerned, our contribution resides in our analysis of the target of Danvy-Filinski’s CPS translation.

We found in the literature an analysis of this target that relies on a second CPS translation in order to account for its call-by-value semantics; and in this last translation it is important that continuations are used linearly (as remarked by Kameyama & Hasegawa for their completeness result [KH03]). Moreover, it is the composition of the two translations, or “double” CPS translation, that is studied in practice.

Here we see the target of Danvy-Filinski's (single) CPS translation through Girard's linear logic and the so-called “boring” translation [Gir87] based on the (!(R)) (\(R\)) (coding of implication). This translation indeed yields a semantics of call-by-value (Maraist et al. [MOT94]), and more precisely, corresponds to Moggli's monadic model [Mog89] where the strong monad is commutative (see Benton & Wadler [BW95]). Thus our axiomatisation of the target corresponds to Moggi's computational calculus together with the following equation that accounts for commutativity:

\[
\lambda x. b \in (let \ y = \mu o. (let \ x = \nu e. (let \ y = \mu o. e)) \in v) = _n \lambda x. b \in (let \ x = \nu e. (let \ y = \mu o. e)) \in v.
\]

We call this calculus \( \lambda_n \), because it turns out to be essentially an extension with pairs of Plotkin's call-by-value λ calculus, which already had implicit in its definition that the order of evaluation (left-to-right or right-to-left) of function application did not matter [Plo75]. With a clear definition of the target, we can focus our analysis on single rather than double CPS translations.

The cornerstone of our approach is that the “boring” semantics of call-by-value allows us to link up with the polarised approach of LLP and LC. This approach is based on the decomposition of the type of continuations into the exponential modality \( ! \) of LLP and the polarity-changing negation that comes from linear logic: \( \neg_{LLP} P = !(\mu \nu Q) \) (we write \( \neg_{LLP} \) the intuitionistic negation).

Delimited CPS translations replace negation by implication, which can be understood in the context of CPS translations as an annotated negation modality: \( \neg_{LP} P = \mu P \rightarrow Q \). Now, this annotated negation modality can be seen through the “boring” translation as an annotated exponential:

\[
\neg_{LP} P = \mu P \rightarrow Q
\]

(\( P \) is an exponential in the sense that its \( !P \) is a function that at least enjoys \( !P(A \otimes B) \rightarrow !P \otimes !P \otimes !P \), since \( A \otimes B \) distributes over \&. The idea that we can consider various kinds of exponentials in linear logic is common, see e.g. Danos, Joinet and Schellinx [DS93].)

Thus, we can imagine a variant of LLP (which we call LLP\(^{LP}\)) which is obtained from linear logic by replacing the usual exponential by the annotated modalities \( \neg_{LP} \) and \( \mu_{LP} \). On top of it we can build a variant of LC, whose computational interpretation is a calculus in which we can retrieve existing delimited control calculi.

We can see the fact that LLP lends itself to analogies with call-by-value — rather than call-by-name — λ calculus as a theoretical argument in favour of Danvy and Filinski's choice. Indeed, if the target of the CPS translation had been in call-by-name (as in Sabry's variant of the calculus [Sab96]), then we would have been unable to provide a polarised decomposition.
The interest of polarisation is that it allows us to account for CPS translations that are not directly related to the monadic approach, such as the semantics of call-by-name $\lambda\mu$ calculus and indeed of LC itself. We show that this is also the case with delimited control by decomposing a call-by-name variant of Danvy-Filinski’s calculus that has been introduced in Herbelin & Ghilezan [HG08].

**Method**

The key technical aspect of this work is the use of a syntax that gives an independent status to evaluation contexts, with their own variables and binders, in the lineage of Curien and Herbelin’s $\lambda\mu$ calculus [CH00], which can also be seen as a term assignment for sequent calculus. We introduced $L_{loc}$ in [Mun09] in order to refine the $\lambda\mu$ calculus to provide a computational interpretation of LC. We see such a notation as a tool for the mind, with which we strive to express the more properties of semantics with the fewest possible symbols.

Another key technical aspect is that we rely on a decomposition of shift and reset into operators that are simpler from the point of view of CPS translations: the $\mu$ operator of Parigot [Par92] together with a dynamically-scoped continuation variable written $\mathbb{P}$. This decomposition has been introduced with the $\lambda\mu\mathbb{P}$ calculus of Ariola, Herbelin and Sabry [AHS04] and Herbelin provided a syntax for it à la $\lambda\mu\mathbb{P}$ in [Her05].

**Technical contribution**

In Section 1, we produce a polarised calculus for delimited control that we call $L_{loc,\mathbb{P}}$. It is a symmetric (yet confluent) variant of $\lambda\mu\mathbb{P}$ that extends $L_{loc}$. We describe its sound CPS semantics into $\lambda^\phi$.

We show that this calculus contains both $\lambda\mu\mathbb{P}$, and its call-by-name variant $\lambda\mu\mathbb{P}_{\mu}$ [HG08], in the sense that the constructs of these languages can be defined in $L_{loc,\mathbb{P}}$ and that these definitions induce translations – 1) – through which the CPS translations factor.

We also study the typed versions of these calculi and introduce a type-and-effect system for $L_{loc,\mathbb{P}}$, extension of LC, with annotations of effects related to shifts of polarities. This type system decomposes the type-and-effect systems of $\lambda\mu\mathbb{P}$, and $\lambda\mu\mathbb{P}_{\mu}$.

In Section 2, we define a system that is essentially Laurent’s LLP [Lau02] in which the exponentials enjoy a richer structure. We call the system LLP$^{\lambda\phi}$ (we omit additives, of which we have no use for our goal). We show how the CPS translation of $L_{loc,\mathbb{P}}$ factors through LLP$^{\lambda\phi}$ in a manner that recalls the translation of LC into LLP: 2) & 3).

We explain the modalities $\mathbb{P}$ and $?_{\mathbb{P}}$ by describing a translation of LLP$^{\lambda\phi}$ into LL similar to the one described for LLP by Laurent [Lau02].

In Section 3 we compare our approach with extant works.

1 **A polarised calculus for delimited control**

We introduce in Fig. 1 a polarised calculus for delimited control ($L_{loc,\mathbb{P}}$). The $L_{loc}$ part accounts for polarised classical logic (as in [Mun09]), while the $\mathbb{P}$ part account for the delimited control operators of the $\lambda\mu\mathbb{P}$ calculus [AHS04].

We describe its CPS semantics and we show that CPS translations of the (call-by-value) $\lambda\mu\mathbb{P}_{\mu}$ calculus of Ariola, Herbelin & Sabry and the (call-by-name) $\lambda\mu\mathbb{P}_{\mu}$ calculus of Herbelin & Ghilezan factor in the following manner:

$$
\begin{align*}
\lambda\mu\mathbb{P}_v & \xrightarrow{\text{POS}} L_{loc,\mathbb{P}} \\
\lambda\mu\mathbb{P}_v & \xrightarrow{\text{NEG}} L_{loc,\mathbb{P}} \\
\end{align*}
$$

where the top arrow relies moreover on the derivability of implication in $L_{loc,\mathbb{P}}$ (modulo administrative reductions for the second diagram).

**Proposition 1** (Confluence). The reduction $\rightarrow$ is confluent.

**Proof.** The reduction $\rightarrow_{\mu}$ is defined by an orthogonal pattern rewrite system, which implies confluence for $\rightarrow$ (Nipkow [Nip91]).

1.1 **Understanding $L_{loc,\mathbb{P}}$**

1.1.1 **Generalities on system $L$**

Our syntax models computation as the rewriting of pairs $(t \parallel e)$, also written $c$ for commands. This rewriting is seen as the interaction between expressions (programs) $(t)$ and their context of evaluation $(e)$. Thus they are kind of abstract machines states, but abstract ones in the sense that the syntax features variables and binders.

The first example of such a machine is Krivine’s abstract machine. It has abstraction, application and a control operator as expressions, and stacks as evaluation contexts:

$$
\begin{align*}
\langle t u \parallel E \rangle & \rightarrow (t \parallel u \cdot E) \\
\langle \lambda x.t \parallel u \cdot E \rangle & \rightarrow (t[u/x] \parallel E) \\
\langle \mu a.c \parallel E \rangle & \rightarrow c[E/\alpha] \\
\end{align*}
$$

This machine describes a call-by-name reduction strategy. Curien and Herbelin [CH00] gave an equally elegant machine for computing in call-by-value: the $\hat{\lambda}\mu\mathbb{P}\hat{c}$ diagonal. It only requires to introduce a new kind of binders on the right.

$$
\begin{align*}
\langle t u \parallel E \rangle & \rightarrow (t \parallel u \cdot E) \\
\langle \lambda x.t \parallel u \cdot E \rangle & \rightarrow (u \parallel \mu x.(t \parallel E)) \\
\langle v \parallel \mu x.c \rangle & \rightarrow c[v/x] \\
\langle \mu a.c \parallel E \rangle & \rightarrow c[E/\alpha] \\
\end{align*}
$$

The $\langle \mu x.c \rangle$ binder must be understood as the “let $x$ be $\square$ in...” context, as the one of Moggi’s computational $\lambda$ calculus [Mog89]. The second line reads:

$$
\hat{E}[(\lambda x.t)u] \rightarrow \hat{E}[\text{let } x \text{ be } [u] \text{ in } t]
$$

if $\hat{E}$ is some term with a hole that corresponds to the evaluation context $E$.

One advantage of working with solid contexts is that the equational theory becomes simplified. For instance, the following equation of Moggi’s calculus for $x$ fresh in $v$:

$$
\text{let } x \text{ be } t \text{ in } (\text{let } y \text{ be } u \text{ in } v) \Rightarrow \text{let } y \text{ be } (\text{let } x \text{ be } t \text{ in } u) \text{ in } v
$$

becomes superfluous since it can be derived from other rules [CH00].

The important remark of Curien and Herbelin was that for the following two tentative reductions:

$$
\langle \mu a.c \parallel \mu x.c \rangle \rightarrow c'[\mu a.c/x]
$$

then the left one, giving priority to expressions, corresponds to call-by-value, denoting a strict reduction of the expression, while
the right one, giving priority to contexts, corresponds to call-
by-name (or the specific variant they studied), denoting a lazy reduction of the expression.\footnote{We believe that Danos, Joinet and Schellinx \cite{danojoinetschellinx} already had at least the intuition of this interpretation of the critical pair. However we could find no such formal claim in print.}

### 1.1.2 Call-by-name, call-by-value...

Wadler showed with the Dual Calculus \cite{wadler} (along with an efficient introduction to symmetric syntaxes for sequent calculus) how it is possible in this kind of syntax to define abstraction and application in terms of simpler connectives. As introductory examples we show how we retrieve similar definitions in \L felon.\footnote{\L felon is not subject to \alpha conversion.}

Note that the syntax in Fig. 1 allows for the two writings \( \langle t_\alpha \parallel t \rangle \) and \( \langle t \parallel t_\alpha \rangle \), which are considered equal in the same way as the \( \lambda \) calculus equates \( \lambda x.x \) and \( \lambda y.y \). Note also that reduction rules always give priority to positives \( t_\alpha \) over negatives \( t \), in the sense that \( \mu x.c \) only reduces when supplied a value \( V_\alpha \). This priority is further reflected in our convention that negative terms are values.

We will show how to derive both call-by-name and call-by-value abstractions and applications, and then we will explain what computational and logical meaning we give to an equality like \( \langle t_\alpha \parallel t \rangle \equiv \langle t \parallel t_\alpha \rangle \).

The call-by-value implication can be derived if we choose to write \( \langle t_\alpha \parallel t \rangle \) instead of \( \langle t \parallel t_\alpha \rangle \) (that is, if we read \( t_\alpha \) as expressions).

#### Example 2 (Call-by-value implication)

Given \( t_\alpha, u_\alpha \) negative terms of \L felon and \( \alpha \), a negative term, we define:

\[
\begin{align*}
\langle \lambda x.t \alpha \parallel t_\alpha \rangle & \equiv \langle \mu x.(\langle t \parallel t_\alpha \rangle) \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle t_\alpha \cdot u_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle \langle t_\alpha \parallel t \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle t_\alpha \cdot u_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle \langle t \parallel t_\alpha \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle t \parallel t_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle \langle t \parallel t_\alpha \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle t \parallel t_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle \langle t \parallel t_\alpha \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle t \parallel t_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle \langle t \parallel t_\alpha \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \
\end{align*}
\]

It is an easy exercise to check that these definitions allow us to retrieve the reduction rules described above for call-by-value.

Conversely, the call-by-name implication can be derived if we choose to write \( \langle t \parallel t_\alpha \parallel t \rangle \) instead of \( \langle t_\alpha \parallel t \rangle \) (that is, if we read \( t_\alpha \) as expressions).

#### Equations.

The following equations are contextual:

\[
\begin{align*}
\langle \eta_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle t \parallel t_\alpha \parallel t \rangle \\
\langle \eta_\alpha \parallel t \rangle & \equiv \mu \alpha.\langle t \parallel t_\alpha \parallel t \rangle \\
\langle \gamma \parallel t \rangle & \equiv \langle \langle t \parallel t_\alpha \parallel t \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle \gamma \parallel t \rangle & \equiv \langle \langle t \parallel t_\alpha \parallel t \rangle \parallel \mu x.(\langle t \parallel t_\alpha \rangle) \rangle \\
\langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle & \equiv \langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle \\
\langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle & \equiv \langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle \\
\langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle & \equiv \langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle \\
\langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle & \equiv \langle \eta \parallel t \parallel t_\alpha \parallel t \parallel t_\alpha \parallel t \rangle \\
\end{align*}
\]

The reader might be surprised — here and later — to see the call-by-name abstraction written \( \lambda \alpha t \) rather than \( \lambda x t \). This is due to the fact that we encode expressions with negative terms, and therefore our notation for negative variables (\( \lambda y.y \)) applies, while the positive co-variables are written \( \mu x.c \). Note also that \( \alpha \) is not subject to \( \alpha \) conversion.

Similarly to the call-by-value case, and modulo the exchange of notations between \( \alpha \)'s and \( x \)'s, one can easily check that these definitions allow us to retrieve the rules of reduction above for the Krivine machine (in particular stacks \( E \) become of the form \( V_\alpha \)).

### 1.1.3 ... and polarisation

At this point the reader might think that our syntax is just an artificial way of writing the two deterministic variants of Curien-Herbelin \( \mu \mu \) or Wadler's Dual Calculus with one syntax, based on a play on words (we leave the \( \mu \) part aside for the moment).

This is only an illusion caused by our choice of presentation, as we explain now.

As made clear by Wadler \cite{wadler}, the rules of negation in sequent calculus can be interpreted computationally as the constructors that exchange expressions and contexts — such constructors, written not, are unary, and they are such that if \( t \) is an expression, then \( \text{not}(t) \) is a context; and conversely.

In the Dual Calculus, \( \text{not}(t) \) is always a value even if \( t \) is not, and the constructors are given the following reduction rule:

\[
\text{not}(t) \parallel \text{not}(u) \rightarrow (u \parallel t)
\]
Thus if we go back to $L_{0\cup \top \&}$ we see that our equality:

$$\langle t_\bot \parallel t_\bot \rangle =_a \langle t_\bot \parallel t_\bot \rangle$$

amounts to no less than omitting the constructors for negation – in the same way as negation is implicit in the proof nets of linear logic, for instance. As a consequence, negation becomes involutive, since, when not rewritten, $\text{not}(\text{not}(t))$ is same as $t$.

Here we encounter the two “traps” mentioned by Girard [Gir91]. Trap #1: our $a$-equality above forces negation to be involutive; but it is false to think that forcing $\neg\neg A$ to be isomorphic to $A$ was sufficient — just taking call-by-value or call-by-name classical logic and imposing this isomorphism as an additional axiom would not work, since this results in equating all terms.

We made this negation implicit instead of explicit in $L_{0\cup \top \&}$, for one reason: the CPS translation of LC is sort of an unsound semantics for the polarity-changing negation, in the sense that it anyway forces the identification of $\langle t_\bot \parallel t_\bot \rangle$ with $\langle t_\bot \parallel t_\bot \rangle$ (see Section 2.1). Therefore our analysis would be no better informed if we had explicit rules for negation: these rules would simply be mapped to the identity transformation on CPS terms.

1.1.4 Polarised connectives

On the logical side, the want for symmetry and in particular an involutive negation is certainly of interest. But here, we are also interested in polarisation because it allows us to mix positive and negative expressions.

The calculus we present has constructors for the multiplicative connectives of polarised classical logic, which are written $\otimes$ (the positive conjunction) and $\&$ (the negative disjunction) which can be understood as a strict pair and a primitive and symmetric version of implication. Additives (the lazy pair & and the strict sum $\oplus$) as well as units are omitted since they are not useful for decomposing CPS translations.

$L_{0\cup \top \&}$ has a negation which we chose to leave implicit in the syntax, as we have already discussed. Also, it has coercions of polarities, or shifts, which can be implicit in the notation — for instance we can form strict pairs with negative terms. They can also be explicit using the “unary patterns” $[\cdot]$ and $\mu[\cdot]c^2$, associated to the connectives $\otimes$ and $\&$. Explicit shifts are typically used at the root of a term, such as in the definition of call-by-value $\lambda$ abstraction above.

We could simulate explicit shifts by implicit ones, but our particular design choice will provide us accuracy in the decomposition of CPS translations, by allowing us to treat particularly such cases where the shift is at the root. Now, if we go back to the definitions of $\lambda$ abstraction, we see that $\lambda c^2$ is indeed a primitive version of implication if we define the follow-

\[2\text{We would have preferred to write these constructs (·) and } \mu[·]c^2 \text{ since they really read best as unary versions of the pair (·, ·). For clarity reasons however, because single parentheses are commonly used for grouping, we chose the notation (·) instead, with no strong preference.} \]
ing:

\((\lambda x.t_\_ x) \equiv (\mu(x,y). (t_\_ y))\)

From such a construct, we see that it is possible to derive the call-by-value (resp. call-by-name) λ abstraction defined above by only introducing the appropriate polarity coercions. (We compare such a primitive implication to a direct-style variant of Levy’s Call-By-Push-Value paradigm [Lev99] which provides a similar decomposition of call-by-value and call-by-name implications in a non-symmetric setting.)

1.1.5 η restriction

On a more technical side, we note that positive terms enjoy what is called the η restriction in Danos et al. [DJS97], that is, any non-value is of the form \(\mu a \cdot \eta \mu(x). \mu y.(x,y) \langle e \rangle\), which is one way of expressing the focalisation of positive constructors. This corresponds also to the notion of administrative normal form, where (for instance) a pair of strict terms \(t \) and \(u\) (computed from left to right, say) is not given in the syntax but defined as:

\[(t,u) \equiv \text{let } x \text{ be } t \text{ in } let y \text{ be } u \text{ in } (x,y)\]

which in \(L_{\text{loc}}\) is represented by \(\mu a \cdot \eta \mu(x). \mu y.(x,y) \langle e \rangle\).

This means that our diagrams omit (if we wanted to be consistent with traditions) one additional trivial step responsible for the restriction to administrative normal forms, starting from a variant of \(L_{\text{loc}}\) without this η restriction, where focalisation is expressed instead through reduction rules like the following:

\[\langle (t,u) \rangle \rightarrow \langle t \rangle \mu x. (u \parallel \mu y.(x,y) \langle e \rangle) \]

if \((t,u)\) is not a value.

The corresponding trivial translation step replaces \((t,u)\) by its above definition, which reflects the arbitrary choice between left-to-right and right-to-left evaluation for the strict pair.

1.1.6 Delimited control operators

Following Ariola, Herbelin & Sabry [AHS04], we can derive shift and reset as follows:

\[S(t_\_ \_ x) \equiv \mu a \cdot \eta \mu(x). \mu y.(x,y) \langle e \rangle \parallel \langle \_ \_ \_ x \rangle\]

\[\text{[t_{\_ \_ x}]} \equiv \mu \_ \_ \_ \mu(x). \mu y.(x,y) \langle e \rangle \parallel \langle \_ \_ \_ x \rangle\]

To help understand the rules associated to \(\_ \_ \_ x\), let us introduce the following notation:

\[c[\_ \_ \_ x] = t_1 \ldots ; \_ \_ \_ x = t_{\_ \_ \_ x}\]

and let us write \(\sigma = [\_ \_ \_ x] = t_1 \ldots ; \_ \_ \_ x = t_{\_ \_ \_ x}\) for such a thing looking like an environment. Note that the definition does not yield uniqueness of the decomposition of a command \(c\) under the form \(c' [\sigma]\).

Now we can look at the rules of \(L_{\text{loc}}\), in particular the ones related to \(\_ \_ \_ x\), as if they were rules of an environment-machine:

\[\langle \mu \_ \_ \_ x \parallel t_{\_ \_ x} \rangle [\sigma] = c[\_ \_ \_ x] = t_{\_ \_ x} \parallel \sigma\]

\[V \parallel [\_ \_ \_ x] \rightarrow (V \parallel t_{\_ \_ x}) [\sigma]\]

\[c[\_ \_ \_ x] = t_{\_ \_ x} \parallel \sigma\]

\[c[\_ \_ \_ x] = \mu q.c'[\_ \_ \_ x] = t_{\_ \_ x} \parallel \sigma\]

The environment \(\sigma\) is only accessed when the variable \(c\) comes in head-position, that is to say, is encountered against a value \(V_{\_ \_ x}\).

It is therefore sensible to consider a head-reduction strategy \(\rightarrow_{\_ \_ x} \leq \rightarrow\) for terms of \(L_{\text{loc}}\), defined by the following:

\[c[\sigma] \rightarrow_{\_ \_ x} c'[\sigma] \text{ whenever } c \rightarrow_{\_ \_ x} c'\]

With this head-reduction we recover (an abstract and symmetric variant of) the environment machines for \(\lambda_{\_ \_ x}\mu_{\_ \_ x}\) of Herbelin and Ghilezan [HG08]. (See also Herbelin’s [Her05].)

1.2 CPS translation

See Fig. 3 for the syntax of the target of our CPS translations \(\lambda^*\), and Fig. 4 for the CPS translation of \(L_{\text{loc}}\) into \(\lambda^*\).

The abstraction for a pair is obtained with \(\lambda(x,y). t \equiv \lambda z. \text{let } (x,y) = z \text{ in } t\).

Remark 4. In \(\lambda^*\), if \(t \rightarrow_{\text{admin}} u\), then \(t \approx u\).

Proof. We note that we have in general:

\[(\lambda(x.u) t) \equiv \lambda y . (\lambda(x.u)(\text{let } x \text{ be } t \text{ in } x))\]

\[\equiv \lambda y . \text{let } x \text{ be } t \text{ in } (\lambda(x.u) x)\]

\[\equiv \text{let } x \text{ be } t \text{ in } u\]

Here we have \(u = E' [x]\) with \(E'\) a finite sequence of linear contexts and therefore we can conclude (by induction on \(E'\)) with a sequence of \(\approx\) equivalences.

\[\text{Proposition 5 (Compositionality). For } p \text{ a command or a term, } x \text{ and } \alpha, V \text{ a value, } V_\alpha \text{ a positive value and } t_{\_ \_ x} \text{ a negative term, one has:}\]

\[(V \parallel [V_\alpha/x]) \rightarrow (V \parallel [V_\alpha/x])\]

\[(p \parallel [V_\alpha/x]) \rightarrow (p \parallel [V_\alpha/x])\]

\[(p \parallel [t_{\_ \_ x}/\alpha]) \rightarrow (p \parallel [t_{\_ \_ x}/\alpha])\]

Proof. Here and in the remaining of the paper, proofs that are omitted are routine verification.

\[\text{Proposition 6 (Simulation). If } L_{\text{loc}} \vdash c \rightarrow c' \text{ then } \lambda^* \vdash c_{\text{CPS}} \rightarrow c'_{\text{CPS}}. \text{ (Here and for other systems defined in this paper, we always define } \approx \equiv (\approx \cup \approx'').\]

\[\text{Proof.} \text{ The result holds for } \rightarrow_{\_ \_ x} \text{ by case analysis. We conclude by induction on the definition of a contextual closure for } \rightarrow_{\_ \_ x}.\]

\[\text{Proposition 7 (Soundness). If } L_{\text{loc}} \vdash c \approx c' \text{ then } \lambda^* \vdash c_{\text{CPS}} \approx c'_{\text{CPS}}. \text{ (Here and for other systems defined in this paper, we always define } \approx \equiv (\approx \cup \approx'').\]

\[\text{Proof.} \text{ Routine is obvious for the cases of the } (c_{\_ \_ x} \text{ and } c_{\_ \_ x}) \text{ rules is that } (\lambda_q.t)u \approx \text{let } q \text{ be } u \text{ in } t \text{ in } \lambda^*.\]

1.3 A type-and-effect system

We propose in Fig. 2 a type system for \(L_{\text{loc}}\), which is an extension with annotations of effects of Girard’s LC — or more precisely of Danos et al’s LK\(^4\) [DJS97].

In order to obtain a convenient type system, we had to add an intermediate judgement that corresponds to the stoup\(^3\) of LC [Gir91]. This distinguished formula, which asserts a linearity constraint stronger than classical provability, is inspired by Andreoli’s focusing proof search discipline [And92]. Its purpose in [Gir91] is to formulate the focalising cut elimination procedure. (See Curien & the author’s [CM10] for a variant of \(L_{\text{loc}}\) where the role of the stoup is emphasised.)

Our stoup appears in the judgement \(V : P_{\_ \_ x} ; \Gamma \) where \(V\) is a value, possibly negative, and \(P_{\_ \_ x}\) is of the form \(P_{\_ \_ x}\). We feel that allowing negatives in the stoup — a choice directly related to our convention that negatives are values — is a sensible choice for having a good treatment of coercions between polarities in our sequent calculus.\(^4\)

\(^3\)In fact it is slightly more constrained and corresponds more closely to Danos et al’s η restriction.

\(^4\)Tradition of having negatives in the stoup goes back to Girard [Gir93]. An alternative to our concise treatment would be to instantiate each logical rule for all possible combination of polarities of the premises, as in LC or LU [Gir91, Gir93]. (Witness the number of rules in these systems!)
Syntax

\[ V ::= x \mid \lambda x.t \mid (V, V) \]
\[ t ::= V \mid t \mid \text{let } x \text{ be } t \mid \text{let } (x, y) \text{ be } t \]

Reduction rules

\[ (\lambda x.t) \to t [V/x] \]
\[ \text{let } x \text{ be } V \to t [V/x] \]
\[ \text{let } (x, y) \text{ be } (V', V') \to t [V/V', V'/y] \]

Equational theory

Linear evaluation contexts are defined with:

\[ E \square ::= \text{let } q \text{ be } \square \text{ in } t \mid \text{let } q \text{ be } t \text{ in } \square \mid \square \square \]

where \( q \) stands for \( x \) and \((x, y)\).

\[ \lambda x.(V x) =_t V \]

Figure 3: A \( \lambda_c \) calculus with pairs: \( \lambda^*_c \).

**CPS translation**

We assume that \( \lambda^*_c \) is given in a standard system of simple types with connectives \( \to, \times \). What justifies our type system is that it is mapped through the CPS translation into the type system of \( \lambda^*_c \).

Positive and annotated negative types translate as follows:

\[ X_{\text{CPS}} \triangleq X \]
\[ (\neg \alpha)_{\text{CPS}} \triangleq (N^-)_{\text{CPS}} \to P_{\text{CPS}} \]
\[ (\alpha \otimes \beta)_{\text{CPS}} \triangleq P_{\text{CPS}} \times Q_{\text{CPS}} \]
\[ (N)_{\text{CPS}} \triangleq (N^-)_{\text{CPS}} \to P_{\text{CPS}} \]

Judgements translate as follows:

\[ c : (\neg \Gamma, \alpha : P) \to (\Gamma, \beta : Q) \]
\[ \text{let } V : P \text{ in } \Gamma \]
\[ \text{let } t : N \text{ in } \Gamma \]
\[ \text{let } \mu c : N \text{ in } \Gamma \]
\[ \text{let } t : P \text{ in } \Gamma \]

\[ \nu \lambda \mu \alpha \beta \]

with \[ \Gamma_{\text{CPS}} = (\neg x : N^-_{\text{CPS}}, \hat{k}_a : P^\alpha_{\text{CPS}}) \] whenever \[ \Gamma = (\hat{x} : \hat{N}, \hat{a} : R) \].

The fact that the type system we gave for \( \text{let}_1 \) maps into a system of simple types for \( \lambda^*_c \) could be checked directly, but it will become obvious once we decompose the translation in smaller steps through LL in Section 2.

1.4 Call-by-value delimited control

Fig. 5 gives the syntax, equations and CPS semantics of Ariola, Herbelin and Sabry’s \( \lambda \mu \alpha \beta \) calculus [AHS04] along the lines of Herbelin and Ghilezan [HG08].

Proposition 8 (Soundness). If \( \lambda \mu \alpha \beta \vdash p \approx q \) then \( \lambda \mu \alpha \beta \vdash p_{\text{CPS}} \approx q_{\text{CPS}} \) where \( p \) is a term or a command.

We come close to having a simulation result as well, but do not, essentially because of the structural substitution defined in the figure.

The definitions in Example 2 canonically induce a translation \( p_{\text{POS}} \) from terms and commands of \( \lambda \mu \alpha \beta \), to terms and commands of \( \text{let}_1 \) (just take \( \epsilon \to \epsilon \), \( x \) to \( x \), \( \mu \alpha \beta \) to \( \mu \alpha \beta \) and \( \mu \alpha \beta \) to \( \mu \alpha \beta \)):

\[ \lambda \mu \alpha \beta \vdash p_{\text{POS}} \Rightarrow \text{let}_1 \]

Proposition 9 (Soundness). If \( \lambda \mu \alpha \beta \vdash p \approx q \) then \( \text{let}_1 \vdash p_{\text{POS}} \approx q_{\text{POS}} \) where \( p \) is a term or a command.

**Proof**: Routine verification except for \( \lambda x.(V x) \approx V \). The latter follows from the fact that we have \( \mu \alpha \beta (V \parallel \mu \alpha \beta (\lambda x.x)) \approx V \) as a consequence of the rule \( (\Gamma) \).

**Proof**: Routine verification except for \( \lambda x.(V x) \approx V \). The latter follows from the fact that we have \( \mu \alpha \beta (V \parallel \mu \alpha \beta (\lambda x.x)) \approx V \) as a consequence of the rule \( (\Gamma) \).

Proposition 10 (Factoring). For \( p \) a command or a term of \( \lambda \mu \alpha \beta \), we have \( p_{\text{POS}} \approx p_{\text{CPS}} \).

1.5 Call-by-name delimited control

Fig. 6 defines the syntax and reduction rules of the \( \lambda \mu \alpha \beta \) calculus along the lines of Herbelin-Ghilezan’s [HG08] as well as a CPS translation into \( \lambda^*_c \) based on the Lafont-Reus-Streicher CPS translation [LRS93].

The CPS translation is essentially\(^6\) the one of Herbelin-Ghilezan. However we replaced Herbelin-Ghilezan’s target calculus (a call-by-name \( \lambda \) calculus with pairs) by \( \lambda^*_c \). We could do so because the image of this CPS translation is indifferent to the evaluation strategy as far as reduction rules are concerned. This is witnessed by the fact that this alternate semantics validates the same rules of reductions.

However, with \( \lambda^*_c \) as a target, the translation no longer validates the equations \( [\hat{p}][\nu \lambda \mu \alpha \beta c] = \eta c \) given in [HG08]; in the sense that we do not have \( ([\hat{p}][\nu \lambda \mu \alpha \beta c]_{\text{CPS}} \approx \eta c_{\text{CPS}} \) in general. This

\( ^{\text{We}} \) differ from [HG08] in that we applied an \( \eta \) expansion in the case of \( \mu \alpha \beta c \), which leaves the denotational semantics of Herbelin & Ghilezan unchanged for a target in call-by-name. This is possible because \( c_{\text{CPS}} \) only reduces to functions.
\( \lambda \mu \tilde{\varphi} \)_n

**Syntax**

\[
\begin{align*}
V & ::= x \mid \lambda x. t \\
t & ::= V \mid t \mid [\mu \tilde{\varphi}]. c \mid [\mu \alpha. c] \\
c & ::= c \mid [\tilde{\varphi}] \\
\equiv & \equiv (\leftrightarrow \cup \equiv)^* 
\end{align*}
\]

**Reductions**

The following reductions are contextual:

\[
\begin{align*}
(\lambda x. t) V & \rightarrow t [V/x] \\
(\mu \alpha. c) t & \rightarrow \mu \beta. c \left( \left[ \beta \right] \left( \square t \right) / \alpha \right) \quad (\beta \text{ fresh}) \\
V (\mu \alpha. c) & \rightarrow [\mu \beta. c \left( \left[ \beta \right] \left( V \square \right) \right) / \alpha] \quad (\beta \text{ fresh}) \\
\alpha & \equiv c \left( e / \alpha \right) \\
\mu \tilde{\varphi}. [\tilde{\varphi}] V & \rightarrow V \quad \text{(even if} \tilde{\varphi} \text{ occurs in} V) \\
\end{align*}
\]

(where \( c \left[ \left[ \beta \right] E \left[ \right] \right] / \alpha \) is the structural substitution: "replace all occurrences of the form \( [\alpha] t \) in \( c \) by \( [\beta] E \{ t \} \))

**CPS translation from \( \lambda \mu \tilde{\varphi} \), to \( \lambda \nu \)

**Reductions**

\[
\begin{align*}
\lambda \mu \tilde{\varphi} & \rightarrow \lambda \nu \\
\frac{t \rightarrow V}{CPS_v : t \rightarrow t} \\
\frac{CPS_v : c \rightarrow t}{CPS_v : t \rightarrow t}
\end{align*}
\]

**Equational theory**

The following equations are contextual:

\[
E, \square ::= \square | E, [\square t] | E, [V \square]
\]

\[
\begin{align*}
[\tilde{\varphi}] \mu \tilde{\varphi}. c =_{\eta} & c \\
\mu \tilde{\varphi}. [\tilde{\varphi}] (\lambda \mu \tilde{\varphi}. [\tilde{\varphi}]) & =_{\eta} (\lambda \mu \tilde{\varphi}. [\tilde{\varphi}]) (\mu \tilde{\varphi}. c) \\
\mu \alpha. [\tilde{\varphi}] & =_{\eta} (\lambda \mu \tilde{\varphi}. [\tilde{\varphi}]) (\mu \tilde{\varphi}. c) \\
\end{align*}
\]

\[
\begin{align*}
\mu \alpha. c & \rightarrow \mu \beta. c \left( \left[ \beta \right] \left( \square t \right) / \alpha \right) \quad (\beta \text{ fresh}) \\
\alpha & \equiv c \left( e / \alpha \right) \\
\mu \tilde{\varphi}. [\tilde{\varphi}] V & \rightarrow V \quad (\text{even if} \tilde{\varphi} \text{ occurs in} V) \\
\end{align*}
\]

**CPS translation from \( \lambda \mu \tilde{\varphi} \), to \( \lambda \nu \)

**Reductions**

\[
\begin{align*}
\lambda \mu \tilde{\varphi} & \rightarrow \lambda \nu \\
\frac{t \rightarrow V}{CPS_v : t \rightarrow t} \\
\frac{CPS_v : c \rightarrow t}{CPS_v : t \rightarrow t}
\end{align*}
\]

**Equational theory**

The following equations are contextual:

\[
E, \square ::= \square | E, [\square t] | E, [V \square]
\]

\[
\begin{align*}
[\tilde{\varphi}] \mu \tilde{\varphi}. c =_{\eta} & c \\
\mu \tilde{\varphi}. [\tilde{\varphi}] (\lambda \mu \tilde{\varphi}. [\tilde{\varphi}]) & =_{\eta} (\lambda \mu \tilde{\varphi}. [\tilde{\varphi}]) (\mu \tilde{\varphi}. c) \\
\mu \alpha. [\tilde{\varphi}] & =_{\eta} (\lambda \mu \tilde{\varphi}. [\tilde{\varphi}]) (\mu \tilde{\varphi}. c) \\
\end{align*}
\]

\[
\begin{align*}
\mu \alpha. c & \rightarrow \mu \beta. c \left( \left[ \beta \right] \left( \square t \right) / \alpha \right) \quad (\beta \text{ fresh}) \\
\alpha & \equiv c \left( e / \alpha \right) \\
\mu \tilde{\varphi}. [\tilde{\varphi}] V & \rightarrow V \quad (\text{even if} \tilde{\varphi} \text{ occurs in} V) \\
\end{align*}
\]

**Related Work**

Herbelin and Ghilezan give in [HG08] type-and-effect systems for \( \lambda \mu \tilde{\varphi} \), and \( \lambda \mu \tilde{\varphi} \). For \( \lambda \mu \tilde{\varphi} \), sequents are of the form:

\[
\tilde{x} : A \vdash t : B \mid \tilde{a} : C \quad ; \quad \tilde{\varphi} : D
\]
where the connective for implication has two annotations: $A \rightarrow B \downarrow$. Now the positive coding of implication yields for $A, B, T, U$ positive formulae of $\LLP_{\ foc, \ ip}$ the following positive definition for this connective:

$$A \rightarrow B \downarrow \overset{\text{def}}{=} \langle A \Rightarrow B \rangle'_{\gamma}.$$ 

It is related to the positive translation [Lau02] through the translation into $\LLP^{\rightarrow \gamma}$ described in next section. With this definition we find back the type system for $\lambda \mu \hat{t}_\mathbb{P}$, as a sub-part of our type system for $\LLP_{\ foc, \ ip}$.

For $\lambda \mu \hat{t}_\mathbb{P}$, sequents are of the form:

$$\hat{\alpha} : \hat{A} \vdash \hat{t} : \hat{B} : \hat{\Sigma} ; \hat{\mu} : \hat{\Xi}$$

with $\hat{\Sigma}, \Xi ::= \downarrow | N \cdot \hat{\Sigma},$ and where the connective for implication has one annotation: $\hat{A} \rightarrow \hat{B}$. Our coding of $\lambda \mu \hat{t}_\mathbb{P}$ yields, for $A, B, C$ denoting negative formulae, the following definition:

$$A_\downarrow \rightarrow B \overset{\text{def}}{=} A \downarrow \Rightarrow B$$

$$\downarrow : 0 \overset{\text{def}}{=} 0$$

$$C : \cdot \downarrow \overset{\text{def}}{=} \downarrow C$$

where $0$ is the positive false constant. This allows us to retrieve the type-and-effect system of $\lambda \mu$ as a sub-part. This coding is related to the negative translation of call-by-name $\lambda \mu$ into $\LLP$ [Lau02].

## 2 LLP with annotated exponentials

Laurent’s LLP [Lau02, LR03] drew attention by giving a fresh view on translations of classical logic. LLP was provided with a theory of proof nets. Such a graphical representation helped in understanding the fact that the Lafont-Reus-Streicher variant [LR93] of the) call-by-name CPS translation is very similar [Lau02, CM10] to Girard’s translation of call-by-name $\lambda$ calculus into linear logic, although the former could seem more intricate than the latter.

A key point for this clarification is that LLP decomposes the type of continuations into a co-variant modality “!” and an invo-
r
tinct than the latter.

2. A negative term $t_\downarrow$ is sent onto a context of $\lambda \hat{\nu}^\times$, i.e. a term with a hole (written $h$ here), with the type of $t_\downarrow$ being mapped to the hole. As a consequence, the translation of sequent calculus into natural deduction accounts for the appearance of administrative redexes.

### Proposition 15 (Simulation & soundness). If $\LLP^{\rightarrow \gamma} \vdash c \rightarrow c^\prime$ then $\lambda \hat{\nu}^\times \vdash c_\downarrow \rightarrow c_\downarrow^\prime$ for some $i \leq 1$. If $\LLP^{\rightarrow \gamma} \vdash c =_\eta c^\prime$ then $\lambda \hat{\nu}^\times \vdash c_\downarrow =_\eta c_\downarrow^\prime$.

### Proposition 16 (Factoring). For $p$ a command or a term of $\LLP_{\ foc, \ ip}$, we have $\lambda \hat{\nu}^\times \vdash p_\LLP_{\ foc, \ ip} =_\eta p_\mathbb{P}$ and for $V$ a value we have $\lambda \hat{\nu}^\times \vdash \hat{V} \LLP_{\ foc, \ ip} =_\eta \hat{V}_\mathbb{P}$. 

Proof. We have:

$$(\mu \hat{t}_\mathbb{P},c)_\LLP_{\ foc, \ ip} = (\text{let } x = b \text{ in } y \text{ be } c_\LLP_{\ foc, \ ip} \text{ in } x y) =_\eta k \hat{c}_\LLP_{\ foc, \ ip}.$$

In all other cases we have an equality.  

Recall that $\hat{\nu}^\times$ is given a standard system of simple types with connectives $\rightarrow, \times$. Formulae and judgements translate as follows:

$$\begin{align*}
X_{\downarrow} & \overset{\text{def}}{=} X \\
(P \& Q)_{\downarrow} & \overset{\text{def}}{=} P_{\downarrow} \times Q_{\downarrow} \\
(N:[N]_{\downarrow})_{\downarrow} & \overset{\text{def}}{=} P_{\downarrow}
\end{align*}$$

\begin{align*}
\vdash t : P & \rightarrow \Gamma \quad \Rightarrow \quad \Gamma_{\downarrow} \vdash t_{\downarrow} : P_{\downarrow} \\
\vdash t : N & \rightarrow \Gamma \cdot x : P \\
\Gamma_{\downarrow} \vdash k : N & \downarrow \rightarrow \Gamma_{\downarrow} : P_{\downarrow} \\
c : \:: \Gamma_{\downarrow} & \vdash c_{\downarrow} : P_{\downarrow}
\end{align*}

with $\Gamma_{\downarrow} = \left( x : (N)_{\downarrow} \right)$ whenever $\Gamma = \left( x : N \right)$.

### 2.2 From $\LLP_{\ foc, \ ip}$ to $\LLP^{\rightarrow \gamma}$

In Fig. 10 we translate $\LLP_{\ foc, \ ip}$ into $\LLP^{\rightarrow \gamma}$. This is an extension to delimited control of the translation of LC into LLP.

For non-delimited control and when negation is explicit, a se-
quent of $\mathbb{L}$ is translated into a sequent of LLP according to the following pattern [Lau02]:

$$\begin{align*}
\overline{\hat{p}, \hat{N}} & \vdash \hat{Q}, \hat{M} \quad \Rightarrow \quad \overline{\hat{p}, \hat{N}}_{\LLP} \vdash \hat{Q}, \hat{M} \\
\overline{\hat{N}} & \vdash \hat{Q} \\
\overline{\hat{N}}_{\LLP} & \vdash \hat{Q}_{\LLP}
\end{align*}$$

For the only-positive case, the construction is related to Moggi’s description of computations in call-by-value through the Kleisli category for a monad, here the $\mathbb{L}$ monad of LLP. The only-
itive case, by symmetry, is related to the co-Kleisli of the $\mathbb{L}$
Co-monad of LLP. (These two dual “mono-polarised” construc-
tions have been investigated by Selinger [Sel01].)

The full translation thus describes a construction which is a generalisation of Kleisli constructions of both a monad and a co-

mond.

Seen from the proof-theoretic side, the goal of this construction of LC from LLP is to conceal exponentials, such that shifts of
LLP<sub>\text{\textsuperscript{\dagger}}</sub>

**Syntax** uses positive variables \(x, y \ldots\) and a distinguished negative variable \(\ast\).

(values) \[ V ::= x | (V, V) | \mu x \cdot c \]

(positive terms) \[ t_+ ::= V | \mu \ast \cdot c \]

(negative terms) \[ t_- ::= \ast | \mu x \cdot c | \mu(x, x) \cdot c | V^* \]

(commands) \[ c ::= (t_+ || t_-) (=_{t_+} t_-) \]

Equational theory The following equations are contextual:

\[ \mu q . \langle q \| t_- \rangle =_\eta t_- \]

\[ \mu x^* . \langle V \| x^* \rangle =_\eta V \]

\[ \langle \mu \ast . c \| \ast \rangle =_\eta c \]

where \(q\) denotes \(x\) and \((x, y)\).

**Reductions** Let \(\to\) the contextual closure of the following rewrite rules:

\[ \langle V \| \mu x \cdot c \rangle \to_\eta c \langle V / x \rangle \]

\[ \langle (V, V') \| \mu(x, y) \cdot c \rangle \to_\eta c \langle V, V' / x, y \rangle \]

\[ \langle \mu x^* \cdot c \| V^* \rangle \to_\eta c \langle V / x \rangle \]

\[ \langle \mu \ast . \langle t_+ \| \ast \rangle \| t_- \rangle \to_\eta \langle t_+ \| t_- \rangle \]

**Formulæ**

(positive) \[ P ::= \text{X} | P \otimes P | \top \]

(negative) \[ N ::= \text{X}^* | N \otimes N \? \]

**Judgements**

\[ \Gamma, \Delta \text{ of the form } x_1 : N_1, \ldots, x_n : N_n \]

\[ \Pi \text{ of the form } \ast : P \ (\Pi \neq \emptyset \text{ always}) \]

\[ \vdash t_- : N \| \Gamma, \Pi \]

\[ \vdash t_+ : P \| \Gamma \]

\[ c : (\vdash \Gamma, \Pi) \]

**Identity**

\[ \vdash x : P \| x : P \]

\[ \vdash \ast : N \| \ast : N \]

\[ \vdash \mu \ast . c : N \| \Gamma \]

**Logic**

\[ \vdash V : P \| \Gamma \]

\[ \vdash V' : Q \| \Delta \]

\[ \vdash \mu(x, y) : N \otimes M \| \Gamma, \Pi \]

**Structure**

\[ \vdash \mu x^* . c : \top \cdot N \| \Gamma \]

**Figure 7:** Syntax, reduction rules and additional equations of LLP<sub>\text{\textsuperscript{\dagger}}</sub>.

**Figure 8:** A system of simple types for LLP<sub>\text{\textsuperscript{\dagger}}</sub>.

**Figure 9:** The translation of LLP<sub>\text{\textsuperscript{\dagger}}</sub> into \(\lambda_\ast^c\).
polari\-ties in LC can be transparent at the level of types, and unobtru-\-sive at the level of terms. The least we can ask of a classical logic is that there be no apparent modality in the system!

We reflect this in $L_{\text{let,ib}}$ and its type system: we provide implicit shifts, thanks to which it is possible to introduce the strict pair of a negative $t_\ldots$ and a positive $V: P$ and obtain $(t_\ldots; V): N \otimes P$ for instance. An exponential is of course introduced in the translation into $\text{LLP}^{\eta,\tau}_L$: $! N \otimes P$. But this exponential already belongs to the translation of $t_\ldots$ to $N$. We see that by construction, there is semantically no need in $L_{\text{let,ib}}$ to have a redundant symbol that would represent a shift in this place.

This involved two design choices: 1) dereliction is implicit because positive values and non-values share the same syntactic category; 2) promotion is implicit because we took the convention that negative terms are values.

**Proposition 17** (Simulation). If $\vdash_{\text{let,ib}} c \rightarrow c'$ then $\vdash_{\text{LLP}^{\eta,\tau}_L} c \rightarrow c'$.

**Proposition 18** (Soundness). If $\vdash_{\text{let,ib}} c \Rightarrow c'$ then $\vdash_{\text{LLP}^{\eta,\tau}_L} c \Rightarrow c'$.

Types and sequents translate as follows:

$$X_{\text{LLP}} \overset{\text{def}}{=} X$$
$$\lambda x.A_{\text{LLP}} \overset{\text{def}}{=} \lambda x.A_{\text{LLP}}$$

(by duality $p_{\text{LLP}} = p_{\text{LLP}}$):

$$\begin{align*}
\langle \lambda x.A_{\text{LLP}}, B_{\text{LLP}} \rangle \overset{\text{def}}{=} & \lambda x.\langle A_{\text{LLP}}, B_{\text{LLP}} \rangle \\
\langle A_{\text{LLP}}, p_{\text{LLP}} \rangle \overset{\text{def}}{=} & \lambda x.\langle A_{\text{LLP}}, B_{\text{LLP}} \rangle \\
\langle A_{\text{LLP}}, B_{\text{LLP}} \rangle \overset{\text{def}}{=} & \lambda x.\langle A_{\text{LLP}}, B_{\text{LLP}} \rangle \\
\vdash_{\text{LLP}} c \rightarrow c' \Rightarrow & \vdash_{\text{LLP}} c \rightarrow c'
\end{align*}$$

We are voluntarily brief: the reader can already refer to Benton & Wadler [BW95] which establish a correspondence between the call-by-value calculus obtained by the boring translation and Moggi's monadic model for a commutative strong monad (recall that our $\lambda x^* \gamma$ is essentially Moggi's calculus plus equations reflecting commutativity of the monad).

In LL, it is possible to take the following definitions:

$$! A \overset{\text{def}}{=} ! (A \otimes P)$$
$$? A \overset{\text{def}}{=} ? (A \otimes P)$$

This modality enjoys the following rules:

$$\begin{array}{ll}
\vdash ! \Gamma, A, P & \vdash ! \Gamma, ! A \\
\vdash ! \Gamma, A, P & \vdash ! \Gamma, ? A, P
\end{array}$$

The translation is an extension of Laurent and Regnier's reversing translation ($\rho$) of LLP into LL [LR03], which is itself inspired from Quatrini and Tortora de Falco [QTF96].

The goal of $\rho$ is to confine structural rules to places where they conform to the constraints of LL — thus it takes care that contraction, weakening and the context of promotion are restricted to formulae of the form $? A$.

At the level of types, all the translation requires is to replace all the atoms $X$ and $X^\perp$ respectively by $! X$ and $? X$. At the level of terms, we can describe such a translation through $\eta$-expansion (invertibility) of negative connectives.

Since we are in a typed setting, $\eta$-expansion allows us to replace contractions and weakenings on formulae $N$ by the same rules on formuale of the form $X^\perp$ or $? X$. Indeed, for instance, if $z$ is not linear in $c : \vdash z : X^\perp \otimes Y^\perp, \Gamma$, then it is possible to get rid of structural rules on $X^\perp \otimes Y^\perp$ by replacing $c$ by $\langle \mu x, c \rangle \langle [\langle x, y \rangle, z] \otimes [\langle x, y \rangle, z] \rangle$, which is equivalent and only uses structural rules on $X^\perp$ and $Y^\perp$. The negative context of promotion is decomposed similarly so that it only contains formulae of the form $X^\perp$ or $? X$ — in addition to the positive premise of our modified promotion, which is the main difference with the case of LLP. This promotion and dereliction are replaced by their derivation above.

We are done when we get rid of structural rules on atoms $X$ and $X^\perp$ by replacing these atoms by $! X$ and $? X$ (the usual exponential, not the annotated one).

3 Comparison with other works

3.1 The polarised analysis of CPS translations

Our analysis of CPS translations remains valid if we restrict to non-delimited control: if we omit the $\text{tp}$ part from $L_{\text{let,ib}}$ we essentially get the syntax for Girard’s polarised classical logic that we introduced in [Mun09]. A simple restriction of our syntax for $\text{LLP}^{\eta,\tau}_L$ yields a syntax for LLP. Therefore, the introductory explanation of CPS translations is obtained by restricting $\lambda \mu \text{tp}$ and $\lambda \mu \text{tp}$ to the $\lambda \mu$ part. As we have seen in the details of the proofs, the factorings we obtain:

$$\begin{array}{c}
\lambda \mu_{\text{H}} \overset{1)}{\longrightarrow} \text{LC} \overset{2)}{\longrightarrow} \text{CPS(\eta/\tau)} \overset{3)}{\longrightarrow} \text{LLP}
\end{array}$$
are even plain equalities — and our explanation is still relevant.

Many aspects of existing works are brought together by our account, as non-delimited CPS translations have already been extensively studied from the point of view of proof theory. We do not hope to establish a comprehensive list of contributions in this vast field, therefore we only mention the closest related works.

Beside Curien-Herbelin’s $\lambda\mu$ [CHO00], we inherit from Wadler’s Dual Calculus [Wad03] the idea that abstraction and application can be derived even in a term syntax. There is no polarisation the works of Curien & Herbelin or Wadler, in the sense that call-by-value and call-by-name are conceived as global reduction strategies, and these calculi do not “mix” call-by-value and call-by-name in the same sense as $\omega\omega$.

Laurent introduced LLP and gave the “positive” translation into LLP for call-by-value $\lambda\mu$ calculus and the “negative” one for call-by-name $\lambda\mu$ calculus [Lau02]. Zeilberger later gave a logical and computational explanation of polarities in terms of values and computations. He described a system [Zei08] in which “call-by-value and call-by-name evaluation freely interact” in a “programming language in which evaluation order is explicitly reflected at the level of types”, he claims. This allowed him to give a first explanation of CPS translations in terms of polarities [Zei09], based on Laurent’s decomposition.

All these works themselves are part of a long series of works in proof theory or programming languages theory (sometimes both), including but not limited to ones of Girard, Danos et al., Laurent et al., Selinger, and others [And92, DJ597, Fil89, Gir87, Gir91, Gir93, Gir90, LRS93, LQdF05, Mog89, Mur92, Par92, QTDF96, Sel01].

While LLP and Zeilberger’s system deal with polarities, they miss the classical polarities of LC and $\LLP\LLP$. In particular, we incorrectly stated, at the time of [Mun09], that we shared with Zeilberger a computational interpretation of polarities. We should have looked more carefully, as we believe indeed that demand-driven and data-driven computations are an interpretation of the polarities of LC, and that the difference between these classical polarities and the notion of polarity in systems like LLP or Zeilberger’s should not be overlooked.

For instance, Zeilberger described his positives as values while the positives of $\omega\omega$, like negatives, are computations that may carry effects. Also, the fact that LLP is based on the adjunction $\dag \vdash$ (or equivalently $\dag \vdash$ for Zeilberger’s [Zei08]), but that the shifts in LC and $\omega\omega$ work the other way ($\dag$ is invertible while $\vdash$ is not), makes a definitive distinction in how the two notions of polarity work.

By emphasizing the importance of the classical polarities of LC and $\LLP\LLP$ in the decomposition, we believe that we brought one missing element to the big picture of how polarisation clarifies the contents of CPS translations.

### 3.2 Polarised linear logic and the study of double-negation translations

Laurent and Regnier [Lau02, LR03] showed — among other contributions — that double-negation translations are not limited to the CPS translations that we can decompose in the spirit of the present paper.

More precisely, they decompose translations of classical logic into linear logic in two steps:

$$\lambda\mu \xrightarrow{G} \LLP \xrightarrow{K} \LL$$

where $G$ represents the Girard-Laurent translations (with our definitions, $\POS\LLP$ and $\NEG\LLP$) and $K$ can be the $\rho$ translation or another translation described in [LR03] called boxy. They showed that these translations can be performed in the reverse order through a call-by-name $\lambda$ calculus that they call CLC, to retrieve previous translations:

$$\lambda\mu \xrightarrow{K'} \CLC \xrightarrow{\sigma} \LL$$

where $K'$ denote again double-negation translations. For instance, with $G$ the negative translation (corresponding to the Lafont-Reus-Streicher one [LRS93]), and $K$ the translation $\rho$, they show that $G'$ is Girard’s “$A \nrightarrow B$” translation and $K'$ is nothing else than the Krivine double-negation translation!

We note that while some double-negation translations can be found in both the $K'$ and the $G$ steps, like the Lafont-Reus-Streicher translation, we have that Krivine’s translation is specific to $K'$, whereas translations that involve recursively defined values, like some CPS translations, are specific to $G$.

Our explanation of this fact is that the steps $K$ and $G$ answer distinct questions: the $K$ translations answer to “how do we obtain structural rules on negative formulae?” whereas the $G$ translations answer to “how do we obtain structural rules on positive formulae?”. It is no surprise that in order to construct classical logic — which requires structural rules on all formulae — from linear logic, both questions have to be answered.

This delineates two families of double negation translations, and also means that it is false to identify double-negation translations in general with CPS translations.

### 3.3 Polarities and delimited control

The first attempt of applying polarisation to the study of delimited control goes back to Shan [Sha03]. Shan introduced an extension of LLP with linear shifts of polarities where he could translate Danvy-Filinski’s calculus. He already hinted at a link with Girard’s boring translation: his translation of intuitionistic logic into his system “is essentially a polarised variant of Girard’s ‘boring’ translation”, he writes. However this translation is more directly related to the CBV CPS translation that corresponds to the second stage of double CPS translations, than the boring translation. As a consequence, Shan’s system seems designed to study double-CPS translations of delimited control, while we decompose single CPS translations. Shan does not study any other system with delimited control than Danvy-Filinski’s.

Together with Kiselyov, Shan [KS07] later gave a type system and a type checking algorithm for Danvy-Filinski’s calculus, whose goal is to be more liberal than the original type system of Danvy and Filinski, while we merely claim that (for the positive part) we retrieve the type system of $\lambda\mu\rho$ [HG08], Kiselyov and Shan use connectives $\downarrow T$ and $\uparrow T$, which correspond modulo duality to annotated negation modalities ($\uparrow\rho Q$) similar to our exponentials $\rho^\uparrow$ and $\rho^\downarrow$. They note that typing a term amounts to transforming it into continuation-passing style. Our type system for $\LLP\LLP\rho$ does not improve this aspect, since it anticipates already to some degree the translation into $\LLP\LLP\rho^\downarrow$, because we use explicit rules for getting in and out of the stoup that anticipate to some degree the translation into $\LLP\LLP\rho^\uparrow$, but which are not reflected in the term syntax. However, “investigating the delimited case” of the duality between call-by-name and call-by-value classical logics was left for “future work” in Kiselyov & Shan [KS07].

More recently, Zeilberger proposed [Zei10] — among other contributions — a “simple account of delimited continuations through a natural generalisation of classical polarised logic” (not to be confused with polarised classical logic, i.e. LC or $\LLP\LLP$) and “in continuation-passing style”, thus expectedly located at the level of LLP. This work is based on hypotheses that make values returned by delimited continuations affect the structure of the polarity-changing negation.
Our own analysis of the polarised decomposition of CPS translations showed that the continuations are related to the rules for the shifts of LLP rather than to the polarity-changing negation, thanks to which we supposed instead that it is the structure of the shifts — here exponentials — that is affected.

In particular, the hypothesis that the symmetry between positive and negative connectives has to be "broken" does not seem necessary. Our own polarised target for delimited CPS translations (LLP\textsuperscript{\text{d}},\text{\text{b}}) contrasts indeed with Zeilberger's by being symmetric.\footnote{For instance we are able to formally describe in \text{L}_{\mu\beta} operators \text{\{\text{\mu},\text{\beta}\text{\}}}}\text{\text{c}} and \text{\{\text{\beta}\}}; a dynamic input binder for a special variable. We did not mention them because we do not believe that structures obtained by duality are necessarily useful, but one might expect a completeness result for some co-monadic "effects" dual to the one of Filinski [Fil09].

Such a difference might find an explanation in the fact that polarisation is given a different meaning in the two works: inspired by a abstract description in the case of Zeilberger's [Zei10], and more faithful to linear logic in our case.

While it is expected that a formal translation of Danvy-Filinski's calculus be given into the positive part of Zeilberger's system, there are no claims of connection with other delimited control calculi of the literature.

3.4 The "subtractive" decomposition of delimited continuations

The work of Ariola, Herbelin and Sabry [AHS04] presents a study of delimited control calculi through translations into a calculus with a subtraction (see also [HGS10]). In linear logic, \(\otimes\text{Q} = \otimes(\text{Q} \otimes \text{P}^-)\) happens to look like a subtraction, and we expect that it is possible, as in Laurent-Regnier [LR03], to translate \(\text{L}_{\mu\beta}\) into LL in the reverse order, starting with a reversing translation into a negative intuitionistic logic with a subtraction whose semantics would be inherited from LL’s \(\otimes\text{N}^-\) modality. This might relate our decomposition to the one of Ariola et al., but would also yield translations in the style of Krivine that do not exist yet.

4 Perspectives

Our explanation of CPS translations shows the relevance of syntaxes à la \(\lambda\mu\varphi\mu\varphi\) / system L: their role is that of an interface between programming languages and semantics. On the one hand, they can be closer to the semantics than (idealised) programming languages, because they present computation in its interactive form (here, in interaction with some context). On the other hand, because we accept that the resulting system looks more like abstract machines, we have to devote the role of reflecting traditions of programming languages (such as presenting the features of the system in a functional style) to an additional layer of abstraction.

This two-step connection between programming languages and semantics is indeed what we find in our decomposition of CPS translations. Here, step 2) defines a CPS-like semantics. The interactive paradigm is relevant, because making explicit an interaction with some context is indeed the idea behind CPS translations. \(\lambda\) calculi with delimited control operators in call-by-value or call-by-name are then retrieved through the syntactic sugar of step 1), which answers the question "How do we get \(a\text{\{n idealised\} programming language back from a system such as L}_{\mu\beta}\)". This question could merit a further development on its own, but is not directly related to denotational semantics.

Now this does not speak of the last step. In fact, step 3) seems to have little semantic value: on the contrary, this step necessarily equates commands modulo duality, and needless bureaucracy is introduced with administrative redexes, all of which seem to only serve the cause of obscurity of CPS translations. This suggests that LLP\textsuperscript{\text{d}},\text{\text{b}} is a more appropriate target than the \(\lambda\) calculus if we want to understand, and be able to question, delimited CPS translations. (And we do not mean our particular syntax for LLP\textsuperscript{\text{d}},\text{\text{b}}, but more generally the system presented: 1- in sequent calculus style and 2- where the double negation modality is decomposed into two distinct co-variant modalities.)

But CPS translations are not the only source of semantics for which we found that system L provided relevant notation: in proof theory already, we found that \(L_{\mu\beta}\) was the appropriate syntax with which we could define for Girard’s LC a polarity-conscious variant of Krivine’s classical realisability. (See [Mun09].)

As far as non-delimited CPS translations are concerned, a stronger result would be to prove the conjecture of Urban and Ratiu [UR06]. They suggest that regarding computation in classical logic, the tq-protocol of Danos-Jointel-Schellinx [DJS97] is as expressive as double-negation translations, by asking if the normal forms that can be obtained via any kind of double-negation translation could be obtained as well with the tq-protocol. So far, the result is only established for particular double-negation translations.

Urban and Ratiu mention two obstacles to the proof. One was the lack of an usable presentation of the tq-protocol. We believe that \(L_{\mu\alpha}\) provides an answer, being a rich syntax with a concise equational theory. (We can indeed simulate \(L_{\mu\beta}\) in \(L^{\text{d},\text{b}}\), by encoding colours \(r\) and \(q\) with polarities, by introducing shifts for instance to simulate the lack of \(\eta\) restriction, and if necessary at the cost of replacing some additive connectives with multiplicatives and vice-versa.) The other obstacle is the absence of a positive definition of what a “double-negation translation” is. To this question we provide no answer.

If the one lesson we would retain from the call-by-value CPS translation is the construction by which strong monads describe call-by-value computation with effects, then the equivalent construction we should retain from our polarised decomposition of CPS translations is the one described in step 2).

This construction corresponds to a generalisation of monadic and co-monadic constructions; and therefore it describes how logical connectives and structures of computation that are algebraic can mix and interact with ones that are co-algebraic.

We believe that the question of a categorical account of step 2) is important for semantics of programming languages. Of course, “by duality”, LC can be identified with its positive and negative subsystems, and we are back to the case of monads (e.g. [Sel01]). But we would request a description of the structure that treats the polarity-changing negation seriously, if only because the perfect symmetry induced by duality might not always be relevant, in particular for computer science. Twenty years after Girard wrote his landmark paper on polarisation [Gir91], this question still has no answer.

Acknowledgements

It was a real chance to benefit from the knowledge of Hugo Herbelin on delimited continuations during the many discussions that we had. I would also like to thank Beniamino Accattoli for discussions related to proof nets for LLP\textsuperscript{\text{d}},\text{\text{b}}. We did not mention such proof nets here, but they had a capital importance in the design of our decompositions and systems.

References

[AHS04] Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A type-
theoretic foundation of continuations and prompts, ICFP ’04,


Hugo Herbelin, C’est maintenant qu’on calcule, au cœur de la dualité, 2005, Habilitation thesis.

An intuitionistic logic that proves markov’s principle, LICS, 2010, pp. 50–56.


Hugo Herbelin, Silvia Ghilezan, and Alexis Saurin, A uniform approach to call-by-name and call-by-value delimited control, Draft, January 2010.


Yves Lafont, B. Reus, and Thomas Streicher, Continuation semantics or expressing implication by negation, Tech. report, University of Munich, 1993.


Christian Urban and Diana Ratiu, Classical logic is better than intuitionistic logic: a conjecture about double-negation translations, CL&C ’06, 2006.


Philip Wadler, Call-by-name is dual to call-by-name, SIGPLAN Not. 38 (2003), no. 5, 189–201.


Yves Lafont, B. Reus, and Thomas Streicher, Continuation semantics or expressing implication by negation, Tech. report, University of Munich, 1993.


Hugo Herbelin, C’est maintenant qu’on calcule, au cœur de la dualité, 2005, Habilitation thesis.

Myriam Quatrini and Lorenzo Tortora de Falco, Polarisation des preuves classiques et renversement, Sciences, Paris t.322, Serie 1, 1996.


Chung-chieh Shan, From shift and reset to polarized linear logic, Manuscript, 2003.


Christian Urban and Diana Ratiu, Classical logic is better than intuitionistic logic: a conjecture about double-negation translations, CL&C ’06, 2006.