The duality of computation

Hugo Herbelin, Pierre-Louis Curien

The Duality of Computation

(revision fixing typos and a few errors – January 2007)

Pierre-Louis Curien
CNRS and University Paris 7
2 place Jussieu
F-75251 Paris Cedex 05
Pierre-Louis.Curien@pps.jussieu.fr

Hugo Herbelin
INRIA
Domaine de Voluceau
F-78153 Rocquencourt Cedex
Hugo.Herbelin@inria.fr

ABSTRACT

We present the $\overline{\lambda}\mu\tilde{\mu}$-calculus, a syntax for λ-calculus + control operators exhibiting symmetries such as program/context and call-by-name/call-by-value. This calculus is derived from implicational Gentzen's sequent calculus LK, a key classical logical system in proof theory. Under the Curry-Howard correspondence between proofs and programs, we can see LK, or more precisely a formulation called $LK_{\mu\tilde{\mu}}$, as a syntax-directed system of simple types for $\overline{\lambda}\mu\tilde{\mu}$-calculus. For $\overline{\lambda}\mu\tilde{\mu}$-calculus, choosing a call-by-name or call-by-value discipline for reduction amounts to choosing one of the two possible symmetric orientations of a critical pair. Our analysis leads us to revisit the question of what is a natural syntax for call-by-value functional computation. We define a translation of $\lambda\mu$-calculus into $\overline{\lambda}\mu\tilde{\mu}$-calculus and two dual translations back to λ-calculus, and we recover known CPS translations by composing these translations.

1. INTRODUCTION

Programming languages present implicit symmetries such as input/output, or program/context. Less obviously – as shown recently by Selinger in a categorical setting [23] –, the picture can be extended to evaluation mechanisms: there exists a symmetry between call-by-name and call-by-value (an earlier attempt in this direction can be found in Filinski [8]).

On the logical side, the best fit for evidencing symmetries is sequent calculus (based on left and right introduction rules). But the correspondence between programs and proofs is traditionally explained through natural deduction (based on right introduction and right elimination rules), implication elimination (also called Modus Ponens) corresponding to procedure application. We believe that this tradition is in good part misleading. In this paper, we present a sequent calculus style syntax that exhibits the above symmetries in a precise, (and – we believe – compelling) way.

A key step in this program was already accomplished in [13, 12], where it was shown that simply-typed λ-terms (or $\lambda\mu$-terms) in (call-by-name) normal form are in bijective correspondence with cut-free sequent calculus proofs in a suitable restriction of Gentzen's LJ (or LK) [9]. Danos, Joinet, and Schellinx identified the same restriction of LK – and called it LKT – as part of a thorough investigation of linear logic encodings of classical proofs [5, 6]. Having gained through this correspondence the "naturalness" that was making the natural deduction usually preferred in practice, there was no reason any longer not to systematically study λ-calculus through sequent calculus rather than through the traditional Curry-Howard correspondence with natural deduction. Sequent calculus is far more well-behaved than natural deduction: it enjoys the subformula property, and destruction rules – cuts – are well characterized in contrast with the elimination rules of natural deduction which superimpose both a construction and a destruction operation: the application is a constructor in a term xM, but is destructive in a term $(\lambda x.M)N$.

The leading goal at the root of the present work was to conceive a "sequent calculus" version of call-by-value λ-calculus and $\lambda\mu$-calculus. Our starting point was the observation that the call-by-value discipline manipulates input much in the same way as (the classical extension of) λ-calculus manipulates output. Computing MN in call-by-value can be viewed as filling the hole (hence an input) of the context $M[\,]$ with the result of the evaluation of N. So the focus is on contexts waiting for values – a situation that sounds dual to that of (output) values being passed to continuations.

This leads us to a syntax with three different syntactic categories: contexts, terms, and commands[1]. Commands are pairs consisting of a term and a context, they represent a closed system containing both the program and its environment. Correspondingly, we type these different categories with three kinds of sequents. The usual sequents $\Gamma \vdash \Delta$ type commands, while the sequents typing terms (contexts) are of the form $\Gamma \vdash A \,|\, \Delta$ ($\Gamma \,|\, A \vdash \Delta$). The symbol "|" serves to single out a distinguished conclusion/output (hypothesis/input), which stands for "where the computation will continue" ("where it happened before").

In the rest of this introduction, we offer as a prologue

[1]Danos has also recognized the relevance of these three categories in [4] where he extends the work of Ogata [18] on the relation between LKQ and call-by-value CPS-translations (LKQ is the other natural restriction of LK considered in [5, 6]).

ICFP'00 Montreal, Canada

a simple justification of the relevance of sequent calculus to the computational study of the λ-calculus. Call-by-name evaluation in the λ-calculus can be specified by the following inference rule:

$$\frac{M \to^* \lambda x.P}{MN \ \to\ P[x \leftarrow N]}\ .$$

This recursive specification may be implemented using a stack as follows ($M \cdot S$ is the result of pushing M on top of S):

$$\begin{array}{rcl}(MN\ ,\ S) & \to & (M\ ,\ N\cdot S)\\ (\lambda x.P\ ,\ N\cdot S) & \to & (P[x\leftarrow N]\ ,\ S)\end{array}$$

This simple device is called Krivine abstract machine. It can be rephrased using contexts instead of stacks:

$$\begin{array}{rcl}(MN\ ,\ E) & \to & (M\ ,\ E[[\,]N])\\ (\lambda x.P\ ,\ E[[\,]N]) & \to & (P[x\leftarrow N]\ ,\ E)\end{array}$$

(Recall that a context is a λ-term with a hole, denoted $[\,]$, which can be filled with a term, or another context: e.g., $E[[\,]N]$ is the context obtained by filling the hole of E with the context $[\,]N$.) Consider the evolution of the types of the holes in the contexts during the execution of the rules. If N has type A and if the hole of E has type B, then the hole of $E[[\,]N]$ has type $A \to B$. This corresponds to a left introduction of implication (note that holes in contexts correspond to inputs). Then the second rule of Krivine abstract machine reads as a cut between an implication which has been introduced on the right and an implication which has been introduced on the left. We are here in the world of sequent calculi, not of natural deduction.

In section 2, we recall the second author's sequent calculus analysis of (call-by-name) $\lambda\mu$-calculus. In section 3, we discuss how to add call-by-value to the picture. This leads us in section 4 to $\overline{\lambda}\mu\tilde{\mu}$-calculus and its typing system $LK_{\mu\tilde{\mu}}$, a logical system for classical logic (limited to the implication connective) with the three kinds of sequents introduced above. In particular, we exhibit the symmetry between the call-by-name and call-by-value disciplines, by means of dual orientations of a single critical pair. In section 5, we analyze two subsyntaxes: the $\overline{\lambda}\mu\tilde{\mu}_T$-calculus and the $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus, and the corresponding sequent calculi $LKT_{\mu\tilde{\mu}}$ and $LKQ_{\mu\tilde{\mu}}$. These calculi correspond to LKT and LKQ; their relation with linear logic is explained in appendix B. Our translation of the $\lambda\mu$-calculus arrives in the intersection of these subsyntaxes, and the target reduction stays in $\overline{\lambda}\mu\tilde{\mu}_T$-calculus ($\overline{\lambda}\mu\tilde{\mu}_Q$-calculus) in the call-by-name (call-by-value) discipline.

Section 6 is a "reverse engineering" exercise. Guided by the goal of translating call-by-value normal forms into $\overline{\lambda}\mu\tilde{\mu}_Q$ normal forms, we revisit source call-by-value evaluation and syntax: we work on an extension of the λ-calculus with a *let* construct, and then on a restriction of this extension which in our opinion *is* the call-by-value counterpart of the λ-calculus.

In section 7, we complete the duality by adding the connective "-" (the difference). This allows us to exhibit fully the duality between terms and contexts. In section 8, we link our analysis to both the classical and the more recent works on continuation semantics. Finally, in section 9, we complete the description of cut-elimination in $LK_{\mu\tilde{\mu}}$. We conclude in section 10.

2. CALL-BY-NAME $\lambda\mu$-CALCULUS IN SEQUENT CALCULUS STYLE

In this section we present (a variant of) the $\overline{\lambda}\mu$-calculus of [13]. This calculus is to sequent calculus what $\lambda\mu$-calculus is to natural deduction. (The syntax and the typing rules of simply-typed $\lambda\mu$-calculus are recalled in appendix A.) The syntax (as well as those of the subsequent sections) embodies the three syntactic categories discussed in the introduction:

$$\begin{array}{ll}\text{Commands} & c ::= \langle v|E\rangle\\ \text{Contexts} & E ::= \alpha \mid v\cdot E\\ \text{Terms} & v ::= x \mid \mu\beta.c \mid \lambda x.v\end{array}$$

Its typing system is a sequent calculus based on judgements of the following form:

$$c : (\Gamma \vdash \Delta) \qquad \Gamma \,|\, E : A \vdash \Delta \qquad \Gamma \vdash v : A \,|\, \Delta$$

and typing rules are given below:

$$\frac{}{\Gamma \,|\, \alpha : A \vdash \alpha : A, \Delta}$$

$$\frac{}{\Gamma, x : A \vdash x : A \,|\, \Delta}$$

$$\frac{\Gamma \vdash v : A \,|\, \Delta \qquad \Gamma \,|\, E : B \vdash \Delta}{\Gamma \,|\, (v\cdot E) : A \to B \vdash \Delta}$$

$$\frac{c : (\Gamma \vdash \beta : B\ ,\ \Delta)}{\Gamma \vdash \mu\beta.c : B \,|\, \Delta}$$

$$\frac{\Gamma, x : A \vdash v : B \,|\, \Delta}{\Gamma \vdash \lambda x.v : A \to B \,|\, \Delta}$$

$$\frac{\Gamma \vdash v : A \,|\, \Delta \qquad \Gamma \,|\, E : A \vdash \Delta}{\langle v|E\rangle : (\Gamma \vdash \Delta)}$$

The notations $\langle v|E\rangle$ can be read as some context $E[\,]$ filled with v; when $E = \alpha$, it becomes just another notation for the naming construction $[\alpha]v$ in $\lambda\mu$-calculus.

Terms can be reduced by the following reductions rules:

$$\begin{array}{lrcl}(\to) & \langle \lambda x.v_1 | (v_2\cdot E)\rangle & \to & \langle v_1[x\leftarrow v_2]|E\rangle\\ (\mu) & \langle \mu\beta.c|E\rangle & \to & c[\beta\leftarrow E]\end{array}$$

Normal forms are those terms where either $v = x$ or ($E = \alpha$ and $v \neq \mu\beta.c$) in subexpressions of the form $\langle v|E\rangle$.

REMARK 2.1. *Our treatment of $\overline{\lambda}\mu$-calculus does not any longer follow totally the "cut=redex" paradigm of sequent calculus as in [13]. Another treatment could have been to add the two (contraction) rules*

$$\frac{\Gamma, x : A \,|\, E : A \vdash \Delta}{\langle x|E\rangle : (\Gamma, x : A \vdash \Delta)}$$

$$\frac{\Gamma \vdash V : A \,|\, \alpha : A\ ,\ \Delta \quad (V = x \ \textit{or}\ V = \lambda x.v)}{\langle V|\alpha\rangle : (\Gamma \vdash \alpha : A\ ,\ \Delta)}$$

and restrict the cut rule to the other combinations of v, E. Then we have the "cut=redex" paradigm of sequent calculus, but another annoying phenomenon shows up: there are two derivations of $\langle x|\alpha\rangle$. This can in turn be solved by removing the introduction rule for x but then $\langle v|E\rangle$ does not any longer include the $\langle x|E\rangle$ construction which must be added explicitly in the grammar. No perfect world.

Until section 9, we ignore the explicit process of substitution, i.e., as in natural deduction, we consider that the replacement is actually carried out completely in a single step. This makes it easier to convey our main observations and results.

We note that any normal $\overline{\lambda}\mu$-term v has the following form:

$$\begin{array}{l} v ::= x \mid \mu\beta.c \mid \lambda x.v \\ c ::= \langle \lambda x.v|\alpha\rangle \mid \langle x|v_1 \cdot \ldots v_n \cdot \alpha\rangle \end{array}$$

We now define two translations $^{\mathcal{N}}$ and $^{>}$ of $\lambda\mu$-calculus into the $\overline{\lambda}\mu$-calculus. The translation $^{\mathcal{N}}$ preserves normal forms, while the translation $^{>}$ is compositional, i.e., preserves the structure of (applicative) terms.

The translation $^{\mathcal{N}}$ involves a parameterization by a context (a trick that goes back to Plotkin's so-called colon translation [20]):

$$\begin{array}{l} x^{\mathcal{N}} = x \\ (\lambda x.M)^{\mathcal{N}} = \lambda x.M^{\mathcal{N}} \\ (MN)^{\mathcal{N}} = \mu\alpha.(MN)^{\mathcal{N}}_{\alpha} \quad \text{for } \alpha \text{ fresh} \\ (\mu\beta.c)^{\mathcal{N}} = \mu\beta.c^{\mathcal{N}} \\ ([\alpha]M)^{\mathcal{N}} = M^{\mathcal{N}}_{\alpha} \\ \\ (MN)^{\mathcal{N}}_{E} = M^{\mathcal{N}}_{N^{\mathcal{N}} \cdot E} \\ V^{\mathcal{N}}_{E} = \langle V^{\mathcal{N}}|E\rangle \quad \text{where } V = x \mid \lambda x.M \mid \mu\alpha.M \end{array}$$

Proposition 2.2. *The translation $^{\mathcal{N}}$ maps normal terms to normal terms.*

Proof. A $\lambda\mu$-normal form is either a variable x, or an abstraction $\lambda x.M$ ($\mu\beta.c$) where M (c)is normal, or an expression $[\alpha]M$ where M is normal and not a μ abstraction, or an expression $xM_1 \ldots M_n$. The latter two cases correspond to the two situations in which a "cut" $\langle v|E\rangle$ is not a redex. □

Notice that $\mu\beta.[\alpha](\ldots(xM_1)\ldots M_k)$ and its translation $\mu\beta.\langle x|(M_1^{\mathcal{N}} \cdot (\ldots(M_k^{\mathcal{N}} \cdot \alpha)\ldots))\rangle$ are essentially the same terms, up to a rearrangement. In the translation $^{\mathcal{N}}$, applicative terms are turned the other way round: a variable applied to a first argument, then to a second, and so on, becomes (an encoding of) a variable applied to a list of arguments.

With simple adjustments (consisting in restricting inessentially the syntax of the $\lambda\mu$-calculus and in atomizing the (μ)-rule), the statement of proposition 2.2 can be improved: it is essentially an isomorphism, i.e., a bijection that preserves reduction step by step both ways. Consider the following restriction of the syntax of the $\lambda\mu$-calculus that disallows applications in contexts of the form $\lambda x.[\,]$ and $M[\,]$ (this is no real restriction since any application MN can be replaced by an expansion $\mu\alpha.[\alpha](MN)$, cf. appendix A):

$$\begin{array}{l} v ::= x \mid \lambda x.v \mid \mu\beta.c \\ c ::= [\alpha]a \\ a ::= v \mid av \end{array}$$

As for reduction, we decompose the (μ) rule of $\overline{\lambda}\mu$-calculus in smaller steps according to the form of the context:

$$\begin{array}{lrcl} (\mu_{app}) & \langle \mu\beta.c|v \cdot E\rangle & \rightarrow & \langle \mu\beta.(c[\beta \leftarrow v \cdot \beta])|E\rangle \\ (\mu_{var}) & \langle \mu\beta.c|\alpha\rangle & \rightarrow & c[\beta \leftarrow \alpha] \end{array}$$

With these adjustments, proposition 2.2 can be restated as:

> The translation $^{\mathcal{N}}$ is an isomorphism: it is bijective, maps normal forms to normal forms, and preserves reductions step by step.

The translation $^{>}$ which we define now is compositional but does not preserve normal forms. This sort of translation is quite well known, since it amounts to translate natural deduction into sequent calculus.

$$\begin{array}{l} x^{>} = x \\ (\lambda x.M)^{>} = \lambda x.M^{>} \\ (MN)^{>} = \mu\alpha.\langle M^{>}|N^{>} \cdot \alpha\rangle \\ (\mu\beta.c)^{>} = \mu\beta.c^{>} \\ ([\alpha]M)^{>} = \langle M^{>}|\alpha\rangle \end{array}$$

The translation $^{>}$ maps a normal form to its image by the previous translation modulo the use of the rule (μ) only ("administrative redexes"). The translation simulates the reduction rule of $\lambda\mu$-calculus up to (μ) expansion (an expansion $\langle v|v' \cdot \alpha\rangle \rightarrow \langle \mu\beta.\langle v|v' \cdot \beta\rangle|\alpha\rangle$ is needed for simulating the rule $(\mu\beta.c)M \rightarrow \mu\alpha.c[\beta \leftarrow (\alpha, M)]$).

Proposition 2.3. *The translation $^{>}$ is a homomorphism from $\lambda\mu$-terms to $\overline{\lambda}\mu$-terms for (call-by-name) reduction, up to (μ) expansion. Moreover, for any $\lambda\mu$-term M, $M^{>}$ reduces by repeated applications of the rule (μ) to $M^{\mathcal{N}}$.*

Proof. Preservation of reduction up to the rule (μ) is trivial. The second part of the statement is an easy consequence of the following:

$$\begin{array}{l} (\mu\beta.[\alpha](xM_1 \ldots M_k))^{>} \rightarrow^{\star} \\ \quad \mu\beta.\langle x|(M_1^{>} \cdot \ldots (M_k^{>} \cdot \alpha)\ldots)\rangle \ \ (k\ (\mu) \text{ steps}) \end{array}$$

□

Remark 2.4. *Without logical or computational loss, one may force the body of a λ-abstraction to have the form $\mu\alpha.c$ (expanding $\lambda x.v$ as $\lambda x.\mu\alpha.\langle v|\alpha\rangle$ when necessary). This observation leads to a variant of the $\overline{\lambda}\mu$-calculus where the λ-abstraction is replaced by a double abstraction $\lambda(x, \alpha).c$, with the following typing rule:*

$$\frac{c : (\Gamma, x : A \vdash \beta : B, \Delta)}{\Gamma \vdash \lambda(x, \alpha).c : A \rightarrow B \mid \Delta}$$

.

3. CALL-BY-VALUE: INTRODUCING $\tilde{\mu}$

Traditionally, one explains how to encode call-by-name in call-by-value by introducing explicit operators that freeze the evaluation of arguments. The same idea can be applied to encode call-by-value on top of call-by-name, now freezing the function until its argument is evaluated. The familiar construct *let* $x = N$ *in* P can be understood in this way. Suppose that we want to compute an application MN in a call-by-value discipline. A first step may consist in writing (*let* $x = N$ *in* Mx), with the intention that N should be evaluated before being passed to Mx, or equivalently that the application of M should be delayed until the argument N is evaluated. The expression can also be written as $E[N]$, where $E = (let\ x = [\,]\ in\ Mx)$. In order to encode such contexts, we introduce a new binding operator $\tilde{\mu}$, which will turn out to be dual to μ. We encode $(let\ x = [\,]\ in\ P)$, evaluated in a context E_0, as $\tilde{\mu}x.\langle P|E_0\rangle$. The $\tilde{\mu}$-abstraction allows us to turn (or *freeze*) the expression P into a context waiting for the value of its hole. If $P = Mx$, we get $\tilde{\mu}x.\langle\mu\alpha'.\langle M|x\cdot\alpha'\rangle|E_0\rangle$, which reduces by (μ) to $\tilde{\mu}x.\langle M|x\cdot E_0\rangle$.

What is the typing rule for $\tilde{\mu}$? First, if c is a command, then $\tilde{\mu}x.c$ is a context (which is dual to $\mu\beta.c$). The typing rule is as follows:

$$\frac{c : (\Gamma, x : B \vdash \Delta)}{\Gamma \,|\, \tilde{\mu}x.c : B \vdash \Delta}$$

One adds the following cut-elimination rule:

$$(\tilde{\mu}) \qquad \langle v|\tilde{\mu}x.c\rangle \quad\rightarrow\quad c[x \leftarrow v]$$

which forms a critical pair with the (μ)-rule, in any command of the form $\langle\mu\beta.c_1|\tilde{\mu}x.c_2\rangle$. We impose that the (μ)-rule has priority in such a redex, yielding $c_1[\beta \leftarrow \tilde{\mu}x.c_2]$. The (compositional) interpretation of the $\lambda\mu$-calculus is now redefined as follows:

$$\begin{array}{l} x^< = x \\ (\lambda x.M)^< = \lambda x.M^< \\ (MN)^< = \mu\alpha.\langle N^<|\tilde{\mu}x.\langle M^<|x\cdot\alpha\rangle\rangle \\ (\mu\beta.c)^< = \mu\beta.c^< \\ ([\alpha]M)^< = \langle M^<|\alpha\rangle \end{array}$$

We next show how the call-by-value reduction is simulated through this translation:

$$\begin{array}{rcl} ((\lambda x.M)N)^< & = & \mu\alpha.\langle N^<|\tilde{\mu}x.\langle\lambda x.M^<|x\cdot\alpha\rangle\rangle \\ & \rightarrow_{(\rightarrow)} & \mu\alpha.\langle N^<|\tilde{\mu}x.\langle M^<|\alpha\rangle\rangle \\ & \rightarrow_{(\tilde{\mu})} & \mu\alpha.\langle M^<[x \leftarrow N^<]|\alpha\rangle \\ & & \qquad (N = x \text{ or } \lambda y.P) \end{array}$$

The last *unfreezing* step is conditioned by the form of $N^<$: if N is a value in the sense of [20], i.e., is an abstraction or a variable, then the $(\tilde{\mu})$-reduction can be applied. Otherwise, $N^<$ begins with a μ, which prevents an immediate application of $(\tilde{\mu})$ and forces the evaluation of $N^<$.

A final remark before we can start to capitalize our analysis of call-by-name and call-by-value is that the translation we just defined for call-by-value works as well *as it stands* for call-by-name, provided one changes the priorities in the reduction system. If one now applies $\tilde{\mu}$ as early as possible, then $M^<$ reduces by repeated use of the $\tilde{\mu}$ rule to $M^>$. We define the *call-by-name (call-by-value) discipline* as the application of the three rewrite rules $(\rightarrow)$, (μ), and $(\tilde{\mu})$ giving priority to $(\tilde{\mu})$ (to (μ)). Then, in call-by-name, $M^>$ is just an optimized version of the translation $M^<$. Therefore, proposition 2.3 can now be rephrased as follows:

> The translation $^<$ is a homomorphism for call-by-name reduction up to (μ) and $(\tilde{\mu})$ expansions. Moreover, for any $\lambda\mu$-term M, $M^<$ reduces by repeated applications of the rules $(\tilde{\mu})$ and (μ) to $M^{\mathcal{N}}$.

This suggests to consider a call-by-value counterpart $^{\mathcal{V}}$ to translation $^{\mathcal{N}}$ and to proposition 2.2. But this raises the question of what should actually be considered as call-by-value normal forms in the λ-calculus. We defer this analysis until section 6.

Another approach to the switch between call-by-name and call-by-value is to refine rule $(\rightarrow)$ into

$$(\rightarrow') \qquad \langle\lambda x.v_1|v_2\cdot e\rangle \quad\rightarrow\quad \langle v_2|\tilde{\mu}x.\langle v_1|e\rangle\rangle$$

If the $(\tilde{\mu})$-rule has the priority, then we recover previous rule $(\rightarrow)$ by a $(\tilde{\mu})$ step, whether v_2 is a value or not (call-by-name discipline). Otherwise, if the (μ)-rule has the priority, we need to evaluate v_2 first and we get the call-by-value discipline. We exploit this approach in the next section.

4. THE $\overline{\lambda}\mu\tilde{\mu}$-CALCULUS

Collecting together the ingredients of the last two sections, we arrive at the $\overline{\lambda}\mu\tilde{\mu}$-calculus whose syntax is

$$\begin{array}{l} c ::= \langle v|e\rangle \\ v ::= x \mid \mu\beta.c \mid \lambda x.v \\ e ::= \alpha \mid \tilde{\mu}x.c \mid v\cdot e \end{array}$$

and evaluation rules are:

$$\begin{array}{ccccc} (\rightarrow') & \langle\lambda x.v_1|v_2\cdot e\rangle & \rightarrow & \langle v_2|\tilde{\mu}x.\langle v_1|e\rangle\rangle \\ (\mu) & \langle\mu\beta.c|e\rangle & \rightarrow & c[\beta \leftarrow e] \\ (\tilde{\mu}) & \langle v|\tilde{\mu}x.c\rangle & \rightarrow & c[x \leftarrow v] \end{array}$$

Observe we have not supposed yet any commitments for call-by-name or call-by-value reduction. This depends on the order of the last two rules:

> Call-by-value consists in giving priority to the (μ)-redexes (which serve to encode the terms, say, of the form MN), while call-by-name gives priority to the $(\tilde{\mu})$-redexes.

The two disciplines are hereafter referred to as CBN reduction and CBV reduction, respectively.

At the typing level, we obtain $LK_{\mu\tilde{\mu}}$ whose typing judgements are:

$$\begin{array}{c} c : (\Gamma \vdash \Delta) \\ \Gamma \vdash v : A \,|\, \Delta \\ \Gamma \,|\, e : A \vdash \Delta \end{array}$$

and whose typing rules are:

$$\frac{\Gamma \vdash v : A \,|\, \Delta \qquad \Gamma \,|\, e : A \vdash \Delta}{\langle v|e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{}{\Gamma \,|\, \alpha : A \vdash \alpha : A, \Delta}$$

$$\frac{}{\Gamma, x : A \vdash x : A \,|\, \Delta}$$

$$\frac{c : (\Gamma \vdash \beta : B, \Delta)}{\Gamma \vdash \mu\beta.c : B \,|\, \Delta}$$

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \,|\, \tilde{\mu}x.c : A \vdash \Delta}$$

$$\frac{\Gamma \vdash v : A \,|\, \Delta \qquad \Gamma \,|\, e : B \vdash \Delta}{\Gamma \,|\, v \cdot e : A \to B \vdash \Delta}$$

$$\frac{\Gamma, x : A \vdash v : B \,|\, \Delta}{\Gamma \vdash \lambda x.v : A \to B \,|\, \Delta}$$

We recall the two ways to translate a $\lambda\mu$-term into a $\overline{\lambda}\mu\tilde{\mu}$-term. The first injection is:

$$\begin{array}{l} x^< = x \\ (\lambda x.M)^< = \lambda x.M^< \\ (MN)^< = \mu\alpha.\langle N^< | \tilde{\mu}x.\langle M^< | x \cdot \alpha\rangle\rangle \\ (\mu\beta.c)^< = \mu\beta.c^< \\ ([\alpha]M)^< = \langle M^< | \alpha\rangle \end{array}$$

and a judgement $\Gamma \vdash M : A \,|\, \Delta$ of the $\lambda\mu$-calculus is translated into a $LK_{\mu\tilde{\mu}}$ judgement $\Gamma \vdash M^< : A \,|\, \Delta$. The second injection is:

$$\begin{array}{l} x^> = x \\ (\lambda x.M)^> = \lambda x.M^> \\ (MN)^> = \mu\alpha.\langle M^> | N^> \cdot \alpha\rangle \\ (\mu\beta.c)^> = \mu\beta.c^> \\ ([\alpha]M)^> = \langle M^> | \alpha\rangle \end{array}$$

and a judgement $\Gamma \vdash M : A \,|\, \Delta$ of the $\lambda\mu$-calculus is translated into a $LK_{\mu\tilde{\mu}}$ judgement $\Gamma \vdash M^> : A \,|\, \Delta$.

Thanks to the rule $(\to')$, both translations $^<$ and $^>$ are compatible with call-by-value discipline. It happens that $^<$ provides us with a right-to-left call-by-value discipline while $^>$ yields a left-to-right call-by-value discipline (whence the notations for these translations). The translation $^>$ relates call-by-value $\lambda\mu$-calculus (see e.g. Ong-Stewart [17]) and $\overline{\lambda}\mu\tilde{\mu}$-calculus in call-by-value discipline, if both considered at an extended equational level. With rule $(\to)$ instead, only translation $^<$ remains correct w.r.t. call-by-value (consider $\mu\beta.[\beta]((\lambda x.z)(\mu\alpha.[\beta]y))$ which should reduce to $\mu\beta.[\beta]y$ under call-by-value).

On the other hand, both translations collapse under the call-by-name discipline (see rephrasing of proposition 2.3 in section 3). Actually rule $(\to')$ presents some redundancy with the translation $^<$. The previous rule $(\to)$ works as well as the new one as far as call-by-name is concerned.

Besides forcing a right-to-left evaluation, the translation $^<$ has another property: only variables (therefore values) occur in position of argument. Then its image (unlike that of $^>$) lies in the intersection of two natural subsystems of $\overline{\lambda}\mu\tilde{\mu}$-calculus for which $(\to)$ is valid in any reduction discipline. This is the purpose of the next section.

REMARK 4.1. *We give some hints on a possible way to relate $LK_{\mu\tilde{\mu}}$ and LK. Consider the version of implicational LK with weakening moved in axiom rules and contraction associated to introduction rules and give a name to all formulas. Then proofs of LK map to the following subset of $\overline{\lambda}\mu\tilde{\mu}$-calculus*

$$c ::= \langle x|\alpha\rangle \parallel \langle y|\mu\alpha.c \cdot \tilde{\mu}x.c\rangle \parallel \langle \lambda x.\mu\alpha.c|\beta\rangle \parallel \langle \mu\alpha.c|\tilde{\mu}x.c\rangle$$

(the constructions respectively correspond to axiom, left introduction, right introduction and cut) which can be seen as $\overline{\lambda}\mu\tilde{\mu}$-calculus normalized (in all configurations but $\langle x|E\rangle$ and $\langle V|\alpha\rangle$) w.r.t. the (non logical) rules

$$\begin{array}{lcll} E & \to & \tilde{\mu}x.\langle x|E\rangle & x \text{ fresh} \\ V & \to & \mu\alpha.\langle V|\alpha\rangle & \alpha \text{ fresh} \end{array}$$

which echo to the last rule discussed in appendix A for $\lambda\mu$-calculus (see also remarks 6.2 and 6.3).

5. TWO WELL-BEHAVED SUBSYNTAXES

In this section, we define subcalculi of $\overline{\lambda}\mu\tilde{\mu}$-calculus that we call $\overline{\lambda}\mu\tilde{\mu}_T$-calculus and $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus because their systems of simple types correspond to the systems LK^t and LK^q, where LK^t (rest LK^q) is LK^{tq} from Danos *et al* [5, 6] where all formulas are colored t (resp q).

Their definition is guided by the requirement of stability under call-by-name and call-by-value evaluation, respectively (propositions 5.1 and 5.3).

Syntax of $\overline{\lambda}\mu\tilde{\mu}_T$	Judgements of $LKT_{\mu\tilde{\mu}}$
$c ::= \langle v\|e\rangle$	$c : (\Gamma \vdash \Delta)$
$v ::= x \parallel \mu\beta.c \parallel \lambda x.v$	$\Gamma \vdash v : A \,\|\, \Delta$
$E ::= \alpha \parallel v \cdot E$	$\Gamma ;\, E : A \vdash \Delta$
$e ::= \tilde{\mu}x.c \parallel E$	$\Gamma \,\|\, e : A \vdash \Delta$

The contexts E are called *applicative contexts*. The typing rules are the same as those of $LK_{\mu\tilde{\mu}}$ for $\langle v|e\rangle$, $\mu\beta.c$, $\tilde{\mu}x.c$, x, and $\lambda x.v$. The other rules are as follows:

$$\frac{}{\Gamma ;\, \alpha : A \vdash \alpha : A, \Delta}$$

$$\frac{\Gamma \vdash v : A \,|\, \Delta \quad \Gamma ;\, E : B \vdash \Delta}{\Gamma ;\, (v \cdot E) : A \to B \vdash \Delta}$$

$$\frac{\Gamma ;\, E : A \vdash \Delta}{\Gamma \,|\, E : A \vdash \Delta}$$

In the judgement $\Gamma ;\, E : A \vdash \Delta$, the sign ";" not only delineates a distinguished hypothesis, but also puts linearity constraints on this hypothesis: it is a stoup, in the terminology of Girard [10]. Notice that implicit contractions are present in the left implication rule. On the other hand, the $\tilde{\mu}$ mechanism is the only way to switch from a distinguished hypothesis to another hypothesis. The syntactic restrictions on $LKT_{\mu\tilde{\mu}}$ say that this can be done only at the price of turning the ";" into a "|". Putting these observations together, we see that the rules of $LKT_{\mu\tilde{\mu}}$ guarantee that a formula in the stoup is never subject to a contraction rule. For the same reasons, it cannot be subject to a weakening rule (weakening *outside the stoup* is implicit in the typing rule for α).

PROPOSITION 5.1. *For any $\lambda\mu$-term M, $M^<$ and any of its CBN reducts in $\overline{\lambda}\mu\tilde{\mu}$-calculus lies in $\overline{\lambda}\mu\tilde{\mu}_T$-calculus.*

REMARK 5.2. *The typing system considered in section 2 lives within $LKT_{\mu\tilde{\mu}}$. Therefore, we shall call it LKT_μ. In retrospect, in that section, we should have written $\Gamma ;\, E : A \vdash \Delta$ as judgement instead of $\Gamma \,|\, E : A \vdash \Delta$. Notice also that with $^>$ or $^{\mathcal{N}}$ instead of $^<$, one has a sharpening: the reducts of the translation lie all in the $\overline{\lambda}\mu$-calculus, in either case.*

We now turn to the call-by-value restriction.

Syntax of $\overline{\lambda}\mu\tilde{\mu}_Q$	Judgements of $LKQ_{\mu\tilde{\mu}}$
$c ::= \langle v \vert e\rangle$	$c : (\Gamma \vdash \Delta)$
$V ::= x \mid \lambda x.v$	$\Gamma \vdash V : A \,;\, \Delta$
$v ::= \mu\beta.c \mid V$	$\Gamma \vdash v : A \mid \Delta$
$e ::= \alpha \mid \tilde{\mu}x.c \mid V \cdot e$	$\Gamma \mid e : A \vdash \Delta$

The terms V are called *values*. The typing rules are the same as in $LK_{\mu\tilde{\mu}}$ for $\langle v|e\rangle$, $\mu\beta.c$, $\tilde{\mu}x.c$, α, and $\lambda x.v$. The other rules are as follows:

$$\frac{}{\Gamma, x : A \vdash x : A \,;\, \Delta}$$

$$\frac{\Gamma \vdash V : A \,;\, \Delta \qquad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid (V \cdot e) : A \to B \vdash \Delta}$$

$$\frac{\Gamma, x : A \vdash v : B \mid \Delta}{\Gamma \vdash \lambda x.v : A \;\to\; B \,;\, \Delta}$$

$$\frac{\Gamma \vdash V : A \,;\, \Delta}{\Gamma \vdash V : A \mid \Delta}$$

PROPOSITION 5.3. *For any $\lambda\mu$-term $\underline{M}$, $M^<$ and any of its CBV reducts in $\overline{\lambda}\mu\tilde{\mu}$-calculus lies in $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus.*

Hence, for any $\lambda\mu$-term M, $M^<$ stands in the intersection of $\overline{\lambda}\mu\tilde{\mu}_T$-calculus and $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus (it uses only V's and E's, in our notation), and its reducts stay in the relevant subsyntax once an evaluation discipline has been fixed.

The systems $\overline{\lambda}\mu\tilde{\mu}_T$-calculus and $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus are also well-behaved without reference to the $\lambda\mu$-calculus. It is easy to check that $\overline{\lambda}\mu\tilde{\mu}_T$-calculus ($\overline{\lambda}\mu\tilde{\mu}_Q$-calculus) is stable under CBN (CBV) reduction and that normal commands lies in $\overline{\lambda}\mu$-calculus ($\overline{\lambda}\tilde{\mu}$-calculus, defined in next section).

6. WHAT IS CBV λ-CALCULUS?

In section 2, we arrived at a perfect correspondence between call-by-name $\lambda\mu$-normal forms and (call-by-name) $\overline{\lambda}\mu$-normal forms. We wish to reach the same goal for call-by-value normal forms. Moreover, for the purposes of duality, we wish to eliminate the need of the μ-operation to encode call-by-value computation, since we did not need the $\tilde{\mu}$ operator to encode call-by-name computation. Recall Plotkin's definition of call-by-value reduction:

$$(\beta_V) \quad (\lambda x.M)V \;\to\; M[x \leftarrow V] \quad (V \text{ variable or abstraction})\ .$$

A typical (β_V) normal form is thus $(\lambda x.M)(yN)$, whose translation

$$\mu\alpha.\langle \mu\beta.\langle N^< | \tilde{\mu}z.\langle y | z \cdot \beta\rangle\rangle | \tilde{\mu}t.\langle \lambda x.M^< | t \cdot \alpha\rangle\rangle$$

contains a $(\to)$ redex. A simple way out is to extend the syntax of the λ-calculus with a *let* construct:

$$M ::= x \mid \lambda x.M \mid MN \mid \mathit{let}\ x = N\ \mathit{in}\ M$$

and to replace (β_V) by the following reduction rules:

$$\begin{array}{ll} (\mathit{let}) & (\lambda x.M)N \;\to\; (\mathit{let}\ x = N\ \mathit{in}\ M) \\ (\mathit{let}_\beta) & (\mathit{let}\ x = V\ \mathit{in}\ M) \;\to\; M[x \leftarrow V] \\ & \quad (V \text{ variable or abstraction}) \end{array}$$

Then we extend the translation in the following way:

$$(\mathit{let}\ x = N\ \mathit{in}\ M)^< = \mu\alpha.\langle N^< | \tilde{\mu}x.\langle M^< | \alpha\rangle\rangle\ .$$

But consider now a term of the form $(\lambda xx'.M)(yN)V$, which is normal for β_V, and whose $(\mathit{let}) + (\mathit{let}_\beta)$ normal form is $(\mathit{let}\ x = yN\ \mathit{in}\ \lambda x'.M)V$. We have:

$$\begin{array}{c} ((\mathit{let}\ x = yN\ \mathit{in}\ \lambda x'.M)V)^< \\ = \\ \mu\alpha.\langle V^< | \tilde{\mu}z.\langle \mu\alpha'.\langle (yN)^< | \tilde{\mu}x.\langle \lambda x'.M^< | \alpha'\rangle\rangle | z \cdot \alpha\rangle\rangle \\ \downarrow_{(\tilde{\mu})} \\ \mu\alpha.\langle \mu\alpha'.\langle (yN)^< | \tilde{\mu}x.\langle \lambda x'.M^< | \alpha'\rangle\rangle | V^< \cdot \alpha\rangle \\ \downarrow_{(\mu)} \\ \mu\alpha.\langle (yN)^< | \tilde{\mu}x.\langle \lambda x'.M^< | V^< \cdot \alpha\rangle\rangle \\ \downarrow_{(\to)} \\ \mu\alpha.\langle (yN)^< | \tilde{\mu}x.\langle M^<[x' \leftarrow V^<] | \alpha\rangle\rangle \\ = \\ (\mathit{let}\ x = yN\ \mathit{in}\ M[x' \leftarrow V])^< \end{array}$$

Hence the translation is able to reduce the "hidden" redex $(\lambda x'.M)V$. To avoid this mismatch, we introduce a further rule in the source language:

$$(\mathit{let}_{app}) \quad (\mathit{let}\ x = a\ \mathit{in}\ M)N \;\to\; (\mathit{let}\ x = a\ \mathit{in}\ (MN)) \quad (x \text{ not free in } N)\ .$$

This rule allows us to reduce $(\lambda xx'.M)(yN)V$ as follows:

$$\begin{array}{lll} (\lambda xx'.M)(yN)V & \to & (\mathit{let}\ x = yN\ \mathit{in}\ \lambda x'.M)V \\ & \to & \mathit{let}\ x = yN\ \mathit{in}\ (\lambda x'.M)V \\ & \to & \mathit{let}\ x = yN\ \mathit{in}\ M[x' \leftarrow V]\ . \end{array}$$

Consider now a term of the form $(\lambda x.M)((\lambda y.V)(zN))$ (y not free in M), which is normal for β_V, and whose $(\mathit{let}) + (\mathit{let}_\beta)$ normal form is $\mathit{let}\ x = (\mathit{let}\ y = zN\ \mathit{in}\ V)\ \mathit{in}\ M$. We have:

$$\begin{array}{c} (\mathit{let}\ x = (\mathit{let}\ y = zN\ \mathit{in}\ V)\ \mathit{in}\ M)^< \\ = \\ \mu\alpha.\langle \mu\alpha'.\langle (zN)^< | \tilde{\mu}y.\langle V^< | \alpha'\rangle\rangle | \tilde{\mu}x.\langle M^< | \alpha\rangle\rangle \\ \downarrow_{(\mu)} \\ \mu\alpha.\langle (zN)^< | \tilde{\mu}y.\langle V^< | \tilde{\mu}x.\langle M^< | \alpha\rangle\rangle\rangle \\ \downarrow_{(\tilde{\mu})} \\ \mu\alpha.\langle (zN)^< | \tilde{\mu}y.\langle M^<[x \leftarrow V^<] | \alpha\rangle\rangle \\ = \\ (\mathit{let}\ y = zN\ \mathit{in}\ M[x \leftarrow V])^< \end{array}$$

Here again the translation manages to reduce the "hidden" redex $(\mathit{let}\ x = V\ \mathit{in}\ M)$. This leads us to introduce another rule:

$$\begin{array}{ll} (\mathit{let}_{let}) & \mathit{let}\ x = (\mathit{let}\ y = N\ \mathit{in}\ M)\ \mathit{in}\ P \\ & \quad \to \mathit{let}\ y = N\ \mathit{in}\ (\mathit{let}\ x = M\ \mathit{in}\ P) \\ & \quad\quad (y \text{ not free in } P) \end{array}$$

It is easily checked that the $(\mathit{let}) + (\mathit{let}_\beta) + (\mathit{let}_{app}) + (\mathit{let}_{let})$ normal forms are as follows:

$$M ::= x \mid \lambda x.M \mid \mathit{let}\ x = xMM_1 \ldots M_n\ \mathit{in}\ M \mid xMM_1 \ldots M_n$$

It is possible at this stage to write a translation $^{\mathcal{V}}$ from this set of normal forms to $\overline{\lambda}\mu\tilde{\mu}$-terms in call-by-value normal form. But we would have to use μ in the translation, typically in a term like $x(yM)$, for which one has to write

$$(x(yM))^{\mathcal{V}}_\alpha = \langle x | \mu\beta.\langle y | M^{\mathcal{V}} \cdot \beta\rangle \cdot \alpha\rangle\ .$$

In order to avoid placing applicative subterms in the contexts, and hence in order to achieve our second goal of "getting rid of μ", we introduce a last rule:

$$(let_{exp}) \quad Ma \rightarrow (let\ x = a\ in\ Mx) \\ (a \text{ application or let expression})$$

The $(let) + (let_\beta) + (let_{app}) + (let_{let}) + (let_{exp})$ normal forms are as follows:

$$V ::= x \mid \lambda x.M$$
$$M ::= V \mid let\ x = yVV_1 \ldots V_n\ in\ M \mid xVV_1 \ldots V_n$$

We are now ready for our call-by-value normal form to normal form translation, which is defined below. Notice the use of the double abstraction (cf. remark 2.4 – the point is that we need μ only under a λ):

$$x^{\mathcal{V}} = x$$
$$(\lambda x.M)^{\mathcal{V}} = \lambda(x,\alpha).M^{\mathcal{V}}_\alpha \quad \text{for } \alpha \text{ fresh}$$

$$V^{\mathcal{V}}_\alpha = \langle V^{\mathcal{V}} | \alpha \rangle$$
$$(xVV_1 \ldots V_n)^{\mathcal{V}}_\alpha = \langle x | V^{\mathcal{V}} \cdot V_1^{\mathcal{V}} \cdot \ldots \cdot V_n^{\mathcal{V}} \cdot \alpha \rangle$$
$$(let\ x = yVV_1 \ldots V_n\ in\ M)^{\mathcal{V}}_\alpha$$
$$= \langle y | V^{\mathcal{V}} \cdot V_1^{\mathcal{V}} \cdot \ldots \cdot V_n^{\mathcal{V}} \cdot \tilde{\mu}x.M^{\mathcal{V}}_\alpha \rangle$$

In this translation, there is no μ. This suggests to consider a calculus symmetric to the $\overline{\lambda}\mu$-calculus of section 2, which we therefore call the $\overline{\lambda}\tilde{\mu}$-calculus. At the typing level, we call the system $LKQ_{\tilde{\mu}}$, to stress that it is a subsystem of $LKQ_{\mu\tilde{\mu}}$ (just as LKT_μ is a subsystem of $LKT_{\mu\tilde{\mu}}$, cf. remark 5.2).

Syntax of $\overline{\lambda}\mu$ (LKT_μ)	Syntax of $\overline{\lambda}\tilde{\mu}$ ($LKQ_{\tilde{\mu}}$)
$c ::= \langle v \| E \rangle$	$c ::= \langle V \| e \rangle$
$v ::= x \mid \lambda(x,\alpha).c \mid \mu\beta.c$	$V ::= x \mid \lambda(x,\alpha).c$
$E ::= \alpha \mid v \cdot E$	$e ::= \alpha \mid \tilde{\mu}x.c \mid V \cdot e$

The rewrite rules for $LKQ_{\tilde{\mu}}$ are $(\tilde{\mu})$ and the following new incarnation of the rule $(\rightarrow)$:

$$\langle \lambda(x,\alpha).c | V \cdot e \rangle \quad \rightarrow \quad c[x \leftarrow V, \alpha \leftarrow e]$$

Reading this back in λ-calculus style, we arrive at the following syntax, which we call $\lambda\tilde{\mu}$-calculus:

$$c ::= [\alpha](VV_1 \ldots V_n) \mid let\ x = VV_1 \ldots V_n\ in\ c$$
$$V ::= x \mid \lambda(x,\alpha).c$$

where usual $\lambda x.V$ can be seen as a shortcut for $\lambda(x,\alpha).[\alpha]V$ (α not free in V). We endow the $\lambda\tilde{\mu}$-calculus with the following reduction rules:

$$(\beta^1_V) \quad let\ y = (\lambda(x,\alpha).c_1)VV_1 \ldots V_n\ in\ c_2$$
$$\rightarrow\ c_1[x \leftarrow V][\,[\alpha]a \leftarrow let\ y = aV_1 \ldots V_n\ in\ c_2]$$
$$(\beta^2_V) \quad [\beta]((\lambda(x,\alpha).c)VV_1 \ldots V_n)$$
$$\rightarrow\ c[x \leftarrow V][\,[\alpha]a \leftarrow [\beta](aV_1 \ldots V_n)]$$
$$(let_\beta) \quad let\ x = V\ in\ c\ \rightarrow\ c[x \leftarrow V]$$

where $c_1[\,[\alpha]a \leftarrow c_2]$ is the term obtained by replacing every occurrence of $[\alpha](VV_1 \ldots V_n)$ in c_1 by c_2 where a is replaced by $(VV_1 \ldots V_n)$. Remark that (β_V) derives directly from (β^2_V)(seeing $\lambda x.V$ as a shortcut for $\lambda(x,\alpha).[\alpha]V$). The translation $^{\mathcal{V}}$ is straightforwardly adapted to $\lambda\tilde{\mu}$-terms, and defines in fact a bijection between the two syntaxes:

$$x^{\mathcal{V}} = x$$
$$(\lambda(x,\alpha).c)^{\mathcal{V}} = \lambda(x,\alpha).c^{\mathcal{V}}$$

$$([\alpha](VV_1 \ldots V_n))^{\mathcal{V}} = \langle V^{\mathcal{V}} | V_1^{\mathcal{V}} \cdot \ldots \cdot V_n^{\mathcal{V}} \cdot \alpha \rangle$$
$$(let\ x = VV_1 \ldots V_n\ in\ c)^{\mathcal{V}} = \langle V^{\mathcal{V}} | V_1^{\mathcal{V}} \cdot \ldots \cdot V_n^{\mathcal{V}} \cdot \tilde{\mu}x.c^{\mathcal{V}} \rangle$$

We can now state the CBV counterpart of proposition 2.2.

PROPOSITION 6.1. *The translation $^{\mathcal{V}}$ is an isomorphism from $\lambda\tilde{\mu}$-terms to $\overline{\lambda}\tilde{\mu}$-terms.*

REMARK 6.2. *In [21], Sabry and Felleisen characterized the theory induced on λ-calculus by the call-by-value CPS translation as the theory induced by the following two equations in addition to (β_V):*

$$(\beta_{lift}) \quad E[(\lambda x.M)Q] \ = \ (\lambda x.E[M])Q \\ (E ::= [\,] \mid EN \mid (\lambda x.P)E)$$

$$(\beta_{flat}) \quad xV_1V_2 \ = \ (let\ y = xV_1\ in\ yV_2)$$

Our rules (let_{app}) and (let_{let}) correspond exactly to (β_{lift}) (cases EN and $(\lambda x.P)E$, respectively). Notice that the other (let) rules are transparent from the point of view of the λ-calculus (without let). The rule (β_{flat}), interpreted from right to left, corresponds to the following η-like equation[2]:

$$\tilde{\mu}x.\langle x | E \rangle \ = \ E \quad (x \text{ not free in } e)$$

Hence Sabry and Felleisen's analysis of call-by-value λ-calculus agrees with ours.

REMARK 6.3. *If we restrict to "η-long" $\overline{\lambda}\tilde{\mu}$-terms (obtained by repeatedly applying expansions $E \rightarrow \tilde{\mu}x.\langle x|E\rangle$), then we arrive at a syntax isomorphic to (a classical extension of) one of the calculi (based on Moggi's computational λ-calculus) considered by Sabry and Wadler in [22], called $\lambda_{c\star\star}$:*

$$c ::= [\alpha]V \mid [\alpha](VV_1) \mid let\ x = V\ in\ c \mid let\ x = VV_1\ in\ c$$
$$V ::= x \mid \lambda(x,\alpha).c$$

(See also [14].) Notice that in the restricted syntax, there are no nested applications, or, equivalently, no expressions of the form $V_1 \cdot (V_2 \cdot (\cdots (V_n \cdot E) \cdots))$ with $n \geq 2$. For example, $V_1 \cdot (V_2 \cdot E)$ expands to $V_1 \cdot \tilde{\mu}x.\langle x | V_2 \cdot E \rangle$.

7. COMPLETION OF THE DUALITY

In order to dualize terms and contexts, we introduce a connective dual to implication: the difference connective, denoted "-". The syntax of $\overline{\lambda}\mu\tilde{\mu}$-calculus is extended as follows (we still call the extension $\overline{\lambda}\mu\tilde{\mu}$-calculus):

$$c ::= \langle v | e \rangle$$
$$v ::= x \mid \mu\beta.c \mid \lambda x.v \mid e \cdot v$$
$$e ::= \alpha \mid \tilde{\mu}x.c \mid v \cdot e \mid \beta\lambda.e$$

We add the following computation rule:

$$(-') \qquad \langle (e_2 \cdot v) | \beta\lambda.e_1 \rangle \quad \rightarrow \quad \langle \mu\beta.\langle v | e_1 \rangle | e_2 \rangle$$

and the following typing rules to $LK_{\mu\tilde{\mu}}$:

$$\frac{\Gamma \mid e : B \vdash \beta : A, \Delta}{\Gamma \mid \beta\lambda.e : B - A \vdash \Delta}$$

[2]This equation is dual to the equation $\mu\alpha.[\alpha]M = M$ (α not free in M) in $\lambda\mu$-calculus (cf. appendix A).

$$\frac{\Gamma \mid e : A \vdash \Delta \qquad \Gamma \vdash v : B \mid \Delta}{\Gamma \vdash (e \cdot v) : B - A \mid \Delta}$$

We define a duality of $\overline{\lambda}\mu\tilde{\mu}$-calculus into itself which works as follows at the type level:

$$\begin{array}{l} X^\circ = X \\ (A \to B)^\circ = B^\circ - A^\circ \\ (B - A)^\circ = A^\circ \to B^\circ \end{array}$$

The translations c°, v°, and e° of commands, terms, and contexts are defined by recursively applying the following table of duality:

$$\begin{array}{cccccccccc} x & \alpha & \mu\beta & \tilde{\mu}x & \langle v|e\rangle & e \cdot v & v \cdot e & \lambda x & \beta\lambda \\ \alpha & x & \tilde{\mu}x & \mu\beta & \langle e|v\rangle & v \cdot e & e \cdot v & \beta\lambda & \lambda x \end{array}$$

Proposition 7.1. *In $LK_{\mu\tilde{\mu}}$, we have:*

$$\left.\begin{array}{r} c : (\Gamma \vdash \Delta) \\ \Gamma \vdash v : A \mid \Delta \\ \Gamma \mid e : A \vdash \Delta \end{array}\right\} \Leftrightarrow \left\{\begin{array}{c} c^\circ : (\Delta^\circ \vdash \Gamma^\circ) \\ \Delta^\circ \mid v^\circ : A^\circ \vdash \Gamma^\circ \\ \Delta^\circ \vdash e^\circ : A^\circ \mid \Gamma^\circ \end{array}\right.$$

One can extend the definition of $\overline{\lambda}\mu\tilde{\mu}_T$-calculus and $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus in such a way that the above proposition restricts and refines to a duality between these calculi. We just give the extended syntax of $\overline{\lambda}\mu\tilde{\mu}_T$-calculus and $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus and leave the rest to the reader:

$\overline{\lambda}\mu\tilde{\mu}_T$-calculus	$\overline{\lambda}\mu\tilde{\mu}_Q$-calculus
$c ::= \langle v\|e\rangle$	$c ::= \langle v\|e\rangle$
$v ::= x \parallel \mu\beta.c \parallel \lambda x.v \parallel E \cdot v$	$V ::= x \parallel \lambda x.c \parallel e \cdot V$
$E ::= \alpha \parallel v \cdot E \parallel \beta\lambda.e$	$v ::= \mu\beta.c \parallel V$
$e ::= \tilde{\mu}x.c \parallel E$	$e ::= \alpha \parallel \tilde{\mu}x.c \parallel V \cdot e \parallel \beta\lambda.e$

8. CPS TRANSLATIONS

In this section, R stands for a fixed (arbitrary) type constant. We define a translation of $LK_{\mu\tilde{\mu}}$ types into intuitionistic types (i.e., the types of the simply-typed λ-calculus, written using the mathematical notation where B^A means the space of functions from A to B) as follows:

$$\begin{array}{l} X^\triangleleft = X \\ (A \to B)^\triangleleft = R^{A^\triangleleft \times R^{B^\triangleleft}} = R^{(A-B)^\triangleleft} \\ (B - A)^\triangleleft = B^\triangleleft \times R^{A^\triangleleft} \end{array}$$

Notice that if we read R as "false", then the image of the translation of $A \to B$ (resp. $B - A$) reads as classically equivalent to $A \to B$ (resp. $B - A$). We next define a translation of $\overline{\lambda}\mu\tilde{\mu}$-terms to λ-terms as follows:

$$\begin{array}{l} \langle v|e\rangle^\triangleleft = v^\triangleleft e^\triangleleft \\ \alpha^\triangleleft = \alpha \\ x^\triangleleft = \lambda k.kx \\ (\mu\beta.c)^\triangleleft = \lambda\beta.c^\triangleleft \\ (\tilde{\mu}x.c)^\triangleleft = \lambda x.c^\triangleleft \\ (\lambda x.v)^\triangleleft = \lambda k.k(\lambda(x,\beta).v^\triangleleft\beta) \\ (v \cdot e)^\triangleleft = \lambda k.v^\triangleleft(\lambda x.k(x, e^\triangleleft)) \\ (\beta\lambda.e)^\triangleleft = \lambda(y,\beta).e^\triangleleft y \\ (e \cdot v)^\triangleleft = \lambda k.v^\triangleleft(\lambda y.k(y, e^\triangleleft)) \end{array}$$

Proposition 8.1.

$$\left.\begin{array}{r} c : (\Gamma \overset{LK_{\mu\tilde{\mu}}}{\vdash} \Delta) \\ \Gamma \overset{LK_{\mu\tilde{\mu}}}{\vdash} v : A \mid \Delta \\ \Gamma \mid e : A \overset{LK_{\mu\tilde{\mu}}}{\vdash} \Delta \end{array}\right\} \Longrightarrow \left\{\begin{array}{c} \Gamma^\triangleleft, R^{\Delta^\triangleleft} \overset{\lambda}{\vdash} c^\triangleleft : R \\ \Gamma^\triangleleft, R^{\Delta^\triangleleft} \overset{\lambda}{\vdash} v^\triangleleft : R^{R^{A^\triangleleft}} \\ \Gamma^\triangleleft, R^{\Delta^\triangleleft} \overset{\lambda}{\vdash} e^\triangleleft : R^{A^\triangleleft} \end{array}\right.$$

Moreover, the translation validates the CBV discipline.

Remark 8.2. *When restricted to $LKQ_{\mu\tilde{\mu}}$, proposition 8.1 can be sharpened in such a way that the following additional implication holds:*

$$\Gamma \vdash V : A\,;\,\Delta \quad \Longrightarrow \quad \Gamma^\triangleleft, R^{\Delta^\triangleleft} \overset{\lambda}{\vdash} V^\triangleleft : A^\triangleleft$$

provided one translates x as x, $\lambda x.V$ as $\lambda(x,\beta).V^\triangleleft\beta$ and V as $\lambda k.kV^\triangleleft$ when considered as a v.

Notice that the disymmetry of the λ-calculus forces the call-by-value orientation of the $(\mu) - (\tilde{\mu})$ critical pair:

$$\begin{array}{rcl} \langle \mu\beta.c_1|\tilde{\mu}x.c_2\rangle^\triangleleft & = & (\lambda\beta.c_1^\triangleleft)(\lambda x.c_2^\triangleleft) \\ & \to & c_1^\triangleleft[\beta \leftarrow \lambda x.c_2^\triangleleft]\,. \end{array}$$

But the translation also takes care of the call-by-name discipline, via duality. We set ${}^\triangleright = {}^\triangleleft \circ {}^\circ$. Then we have:

$$\begin{array}{l} X^\triangleright = X \\ (A \to B)^\triangleright = B^\triangleright \times R^{A^\triangleright} \\ (B - A)^\triangleright = R^{(B \to A)^\triangleright} = R^{A^\triangleright \times R^{B^\triangleright}} \end{array}$$

Notice that this time $A^\triangleright$ reads as classically equivalent to $\neg A[X \leftarrow \neg X]$. Notice also that ${}^\triangleright$ can alternatively be taken as primitive and ${}^\triangleleft$ defined as ${}^\triangleleft = {}^\triangleright \circ {}^\circ$.

Proposition 8.3.

$$\left.\begin{array}{r} c : (\Gamma \overset{LK_{\mu\tilde{\mu}}}{\vdash} \Delta) \\ \Gamma \overset{LK_{\mu\tilde{\mu}}}{\vdash} v : A \mid \Delta \\ \Gamma \mid e : A \overset{LK_{\mu\tilde{\mu}}}{\vdash} \Delta \end{array}\right\} \Longrightarrow \left\{\begin{array}{c} R^{\Gamma^\triangleright}, \Delta^\triangleright \overset{\lambda}{\vdash} c^\triangleright : R \\ R^{\Gamma^\triangleright}, \Delta^\triangleright \overset{\lambda}{\vdash} v^\triangleright : R^{A^\triangleright} \\ R^{\Gamma^\triangleright}, \Delta^\triangleright \overset{\lambda}{\vdash} e^\triangleright : R^{R^{A^\triangleright}} \end{array}\right.$$

Moreover, the translation validates the CBN discipline.

Combining ${}^\triangleright$ and ${}^\triangleleft$ with the translation ${}^>$ from $\lambda\mu$-terms, we obtain two CPS-translations to λ-terms:

$$\begin{array}{lccc} \text{(CBN)} & \Gamma \overset{\lambda\mu}{\vdash} M : A \mid \Delta & \Rightarrow & \Delta^\triangleright, R^{\Gamma^\triangleright} \overset{\lambda}{\vdash} M^{>\triangleright} : R^{A^\triangleright} \\ \text{(CBV)} & \Gamma \overset{\lambda\mu}{\vdash} M : A \mid \Delta & \Rightarrow & \Gamma^\triangleleft, R^{\Delta^\triangleleft} \overset{\lambda}{\vdash} M^{>\triangleleft} : R^{R^{A^\triangleleft}} \end{array}$$

The two translations are known in the literature: ${}^{>\triangleright}$ is the (call-by-name) Lafont-Hofmann-Streicher translation [15] , and ${}^{>\triangleleft}$ is (the extension to call-by-value $\lambda\mu$-calculus of) Plotkin's call-by-value translation [20]. The following dictionary is useful to recognize this:

CBN	CBV
$K_A = A^{\circ\triangleleft}$ $C_A = R^{K_A}$	$V_A = A^\triangleleft$ $K_A = R^{V_A}$ $C_A = R^{K_A}$
$C_\Gamma, K_\Delta \overset{\lambda}{\vdash} M^{>\triangleright} : C_A$	$V_\Gamma, K_\Delta \overset{\lambda}{\vdash} M^{>\triangleleft} : C_A$
$K_{A\to B}$ $= (B^\circ - A^\circ)^\triangleleft$ $= K_B \times C_A$	$V_{A\to B}$ $= R^{V_A \times R^{V_B}}$ $\cong V_A \to C_B$

Here, the letters V, K, and C stand for values, continuations, and computations, respectively. Lafont-Hofmann-Streicher semantics maps computations to computations and interprets a continuation of type $A \to B$ as a pair of a computation of type A and a continuation of type B (think of the stack $N \cdot S$ of section 1). Plotkin's call-by-value semantics maps values to computations, and interprets a value of type $A \to B$ as a function from values to computations. repeatedly

REMARK 8.4. *If we combine* $^{\triangleright}$ *and* $^{\triangleleft}$ *with the translation* $^{<}$ *rather than with the translation* $^{>}$*, the picture is as follows: in CBN,* $M^{<\triangleright}$ *reduces to* $M^{>\triangleright}$*, while, in CBV,* $M^{<\triangleleft}$ *provides a right-to-left variant of Plotkin's translation. For* $M = M_1M_2$*, we get respectively:*

$$\begin{array}{l} \lambda k.M_1^{>\triangleleft}(\lambda m_1.M_2^{>\triangleleft}(\lambda m_2.m_1(m_2,k))) \\ \lambda k.M_2^{<\triangleleft}(\lambda m_2.M_1^{<\triangleleft}(\lambda m_1.m_1(m_2,k))) \ . \end{array}$$

9. HEAD REDUCTION IN AN ABSTRACT MACHINE

In this section, we specify two kinds of (weak) head reduction machine.

The first machine is quite standard and based on environments. Instead of commands $\langle v|e\rangle$, we manipulate expressions having the form $\langle v\{\rho_1\}|e\{\rho_2\}\rangle$, where ρ_1 and ρ_2 are (explicit) environments, i.e., lists of bindings of the form $(x = v\{\rho\})$ or $(\alpha = e\{\rho\})$. Given ρ_1 and x, we write $\rho_1(x) = v\{\rho_2\}$ if $(x = v\{\rho_2\})$ is the first binding of x appearing in ρ_1. The notation $\langle v|e\rangle\{\rho\}$ is a shorthand for $\langle v\{\rho\}|e\{\rho\}\rangle$. To evaluate a command c, we start the machine with $c\{\}$.

$$\begin{array}{l} \langle(\lambda x.v_1)\{\rho_1\}|(v_2 \cdot e)\{\rho_2\}\rangle \\ \quad \to \langle v_1\{(x = v_2\{\rho_2\}) \cdot \rho_1\}|e\{\rho_2\}\rangle \end{array}$$

$$\begin{array}{l} \langle(\mu\beta.c)\{\rho_1\}|e\{\rho_2\}\rangle \\ \quad \to c\{(\beta = e\{\rho_2\}) \cdot \rho_1\} \end{array}$$

$$\begin{array}{l} \langle v\{\rho_1\}|(\tilde{\mu}x.c)\{\rho_2\}\rangle \\ \quad \to c\{(x = v\{\rho_1\}) \cdot \rho_2\} \end{array}$$

$$\begin{array}{l} \langle x\{\rho_1\}|e\{\rho_2\}\rangle \\ \quad \to \langle \rho_1(x)|e\{\rho_2\}\rangle \qquad (\rho_1(x) \text{ is defined}) \end{array}$$

$$\begin{array}{l} \langle v\{\rho_1\}|\alpha\{\rho_2\}\rangle \\ \quad \to \langle v\{\rho_1\}|\rho_2(\alpha)\rangle \qquad (\rho_2(\alpha) \text{ is defined}) \end{array}$$

Remark that bindings of terms (contexts) have the restricted form $(x = V\{\rho\})$ $((\alpha = E\{\rho\}))$ when reducing CBV (CBN).

The second machine exploits the idea of encoding environments by means of indexes in a stack as in the Pointer Abstract Machine from Danos-Regnier [7] (a restriction of it was studied in a previous work of the authors [3]).

The stack is a sequence of bindings that bind either a term or a context. The concrete syntax for bindings is:

$$(x \stackrel{n}{=} v\{p\}) \text{ or } (\alpha \stackrel{n}{=} e\{p\}) \quad (n, p \text{ natural numbers}) \ .$$

The expression $v\{p\}$ is a closure: p is a pointer in the stack to where the environment of v starts. Similarly for $e\{p\}$. Any pointer to the stack defines an environment by chaining the bindings with the pointer n.

A state of the machine is $\langle v\{p\}|e\{q\}\rangle s$. If s is a stack, we denote its length by $\| s \|$, and $s_{|n}$ is the stack restricted to its n first elements (thus, $\| s_{|n} \| = n$). Pointers are relative not to the top but from the bottom of the stack. To evaluate $\langle v|e\rangle$, we start from $\langle v\{0\}|e\{0\}\rangle$ (with 0 denoting the empty environment). The rules of the machine are as follows:

$$\begin{array}{l} \langle(\lambda x.v_1)\{p\}|(v_2 \cdot e)\{q\}\rangle\{s\} \\ \quad \to \langle v_1\{\| s \|\}|e\{q\}\rangle\{(x \stackrel{p}{=} v_2\{q\}) \cdot s\} \end{array}$$

$$\begin{array}{l} \langle(\mu\beta.\langle v|e'\rangle)\{p\}|e\{q\}\rangle\{s\} \\ \quad \to \langle v\{\| s \|\}|e'\{\| s \|\}\rangle\{(\beta \stackrel{p}{=} e\{q\}) \cdot s\} \end{array}$$

$$\begin{array}{l} \langle v\{p\}|(\tilde{\mu}x.\langle v'|e\rangle)\{q\}\rangle\{s\} \\ \quad \to \langle v'\{\| s \|\}|e\{\| s \|\}\rangle\{(x \stackrel{q}{=} v\{p\}) \cdot s\} \end{array}$$

$$\begin{array}{l} \langle x\{p\}|e\{q\}\rangle\{s\} \\ \quad \to \langle s_{|p}(x)|e\{q\}\rangle \qquad (s_{|p}(x) \text{ is defined}) \end{array}$$

$$\begin{array}{l} \langle v\{p\}|\alpha\{q\}\rangle\{s\} \\ \quad \to \langle v\{p\}|s_{|q}(\alpha)\rangle \qquad (s_{|q}(\alpha) \text{ is defined}) \end{array}$$

$$\begin{array}{l} \langle v\{p\}|e\{q\}\rangle\{s\} \\ \quad \to \langle v\{p\}|e\{q\}\rangle\{s_{|max(p,q)}\} \end{array}$$

where

$$\begin{array}{l} ((x \stackrel{n}{=} v\{p\}) \cdot s)(x) = v\{p\} \\ ((y \stackrel{n}{=} v\{p\}) \cdot s)(x) = s_{|n}(x) \quad x \neq y \\ ((\alpha \stackrel{n}{=} e\{q\}) \cdot s)(x) = s_{|n}(x) \end{array}$$

and similarly for $s(\alpha)$.

REMARK 9.1. *The last rule acts as a garbage collecting rule: it removes the part of the stack which is used neither by the term (the code) nor by its context. For instance, in a functional language with flat atomic types (even with fixpoints, e.g. PCF), the application of the last rule guarantees that a program of atomic type ends with an empty stack.*

10. RELATED AND FUTURE WORKS

We end by miscellaneous remarks organized along hopefully helpful keywords.

• *Symmetry.* Altogether, we have defined six calculi: the full $\overline{\lambda}\mu\tilde{\mu}$ syntax in CBN and CBV discipline (section 4), the subsyntaxes $\overline{\lambda}\mu\tilde{\mu}_T$-calculus / $LKT_{\mu\tilde{\mu}}$ and $\overline{\lambda}\mu\tilde{\mu}_Q$-calculus / $LKQ_{\mu\tilde{\mu}}$ (section 5), and as further restrictions the $\overline{\lambda}\mu$-calculus (section 2) and the $\overline{\lambda}\tilde{\mu}$-calculus (section 6). At all these levels, the duality of call-by-name and call-by-value is governed by the symmetry of the μ-terms and the $\tilde{\mu}$-terms. The situation is summarized in the following table. In the table, $\hookleftarrow$ shows a source-to-target direction (cf. proposition 5.3), while $\leftrightarrow$ indicates stronger one-to-one correspondences (cf. proposition 2.2 and 6.1). It would be interesting to complete this picture by adding more strong correspondences $\leftrightarrow$. One could in particular show that CBV $\lambda\mu$-calculus (for which *let* arrives naturally) relates to CBV $\overline{\lambda}\mu\tilde{\mu}$-calculus. One could also consider call-by-name λ-calculus plus *let*, which has CBN $\overline{\lambda}\mu\tilde{\mu}$-calculus as a target. In such a calculus, one could delay some substitutions, as in *let* $x = yM_1$ *in* M_2 (or $\mu\alpha.\langle y|v_1 \cdot \tilde{\mu}x.\langle v_2|\alpha\rangle\rangle$), i.e. we could force some sharing of subcomputations.

Logic	Syntax	Evaluation		Language
$LK_{\mu\tilde{\mu}}$	$\overline{\lambda}\mu\tilde{\mu}$	$\begin{cases} \text{ND} \\ \text{CBN} \\ \text{CBV} \end{cases}$		
$LKT_{\mu\tilde{\mu}}$	$\overline{\lambda}\mu\tilde{\mu}_T$	CBN		
$LKQ_{\mu\tilde{\mu}}$	$\overline{\lambda}\mu\tilde{\mu}_Q$	CBV	$\hookleftarrow$	CBV $\lambda\mu$
LKT_{μ}	$\overline{\lambda}\mu$	CBN	$\leftrightarrow$	CBN $\lambda\mu$
$LKQ_{\tilde{\mu}}$	$\overline{\lambda}\tilde{\mu}$	CBV	$\leftrightarrow$	$\lambda\tilde{\mu}$

- *Non-determinism.* In [1] and [24], the non-determinism of classical logic is encapsulated in critical pairs similar to the $(\mu) - (\tilde{\mu})$ pair. But no explicit connection with call-by-name / call-by-value appears in those works.

- *Semantics.* It is fairly clear that our syntax $LK_{\mu\tilde{\mu}}$ with the CBN (CBV) machine can be interpreted in Selinger's control (co-control) categories: the categorical construction interpreting $\rightarrow$ $(-)$ is an exponent (a co-exponent), and $-$ $(\rightarrow)$ is a weak co-exponent (a weak exponent). It should be interesting and useful to work out the details of this interpretation.

- *Dynamics.* Laurent has investigated (CBN) proof-nets for an extended polarized linear logic that closely corresponds to $\lambda\mu$-calculus [16]. These proof-nets enjoy a simple correctness criterion. This suggests that a proof-net representation of $LK_{\mu\tilde{\mu}}$ is possible.

- *Games.* We intend to develop a game interpretation of our calculi. A game-theoretic analysis of call-by-value has been given by Honda and Yoshida [14]. One could hope to sharpen the analysis so as to obtain a game-theoretic reading of the duality of computation.

- *Expressivity.* We have used the difference connective in a purely formal way. It would be interesting to study this connective for its own sake and to get insights into its computational meaning. Crolard has initiated this kind of investigation [2]. One way of seeing the difference connective is that it allows us to view contexts as values: $v \cdot e$ is both a context whose hole has a function type $A \rightarrow B$ (as explained in the introduction) and a pair of values of type $B - A$ (viewed as a product type). Under the latter interpretation, a cut of the form $\langle\lambda(x,\alpha).c|e\rangle$ appears as a destructive *let*: "evaluate e to a pair, and bind the two components of the pair to x and α, respectively".

11. REFERENCES

[1] F. Barbanera and S. Berardi 1996. A symmetric λ-calculus for "classical" program extraction. *Information and Computation 125*, 103–117.

[2] T. Crolard 1999. Typage des coroutines en logique soustractive. In *Proc. Journées Francophones des Langages Applicatifs*, Collection Didactique, INRIA (http://pauillac.inria.fr/jfla99).

[3] P.-L. Curien and H. Herbelin 1998. Computing with Abstract Böhm Trees. In *Proceedings of the 3rd Fuji International Symposium on Functional and Logic Programming*, Eds M. Sato & Y. Toyama, World Scientific, 20–39.

[4] V. Danos 1999. Sequent Calculus and Continuation Passing Style Compilation. To appear in *Proc. of the 11th Congress of Logic, Methodology and Philosophy of Science*, held in Cracow, Kluwer.

[5] V. Danos, J.-B. Joinet and H. Schellinx 1995. LKQ and LKT: sequent calculi for second order logic based upon dual linear decompositions of classical implication. In *Advances in Linear Logic*, Cambridge University Press, 211–224.

[6] V. Danos, J.-B. Joinet and H. Schellinx 1997. A New Deconstructive Logic: Linear Logic. In *Journal of Symbolic Logic 62(3)*, 755–807.

[7] V. Danos and L. Regnier 1990. Machina ex deo, ou encore quelque chose à dire sur la machine de Krivine. Unpublished.

[8] A. Filinski 1989. Declarative Continuations: An Investigation of Duality in Programming Language Semantics. In *Proc. of Category Theory and Computer Science (CTCS)*, LNCS 389, 224–249.

[9] G. Gentzen 1935. Investigations into logical deduction. E.g. in *Gentzen collected works*, Ed M. E. Szabo, North Holland, 68ff (1969).

[10] J.-Y. Girard 1993. On the unity of logic. *Annals of Pure and Applied Logic 59*, 201–217.

[11] Ph. de Groote 1994. On the relation between the $\lambda\mu$-calculus and the syntactic theory of control. In *Proc. of the International Conference on Logic Programming and Automated Reasoning (LPAR)*, Lecture Notes in Computer Science 822, 31–43.

[12] H. Herbelin 1994. A lambda-calculus structure isomorphic to sequent calculus structure. In *Proc. of Computer Science Logic (CSL)*, Lecture Notes in Computer Science 933, 61–75.

[13] H. Herbelin 1995. *Séquents qu'on calcule*, Thèse de Doctorat, Université Paris 7.

[14] K. Honda and N.Yoshida 1997. Game-theoretic analysis of call-by-value computation. In *Proc. of International Colloquium on Automata Languages and Programs (ICALP)*, Lecture Notes in Computer Science 1256.

[15] M. Hofmann and T. Streicher 1997. Continuation models are universal for $\lambda\mu$-calculus. In *Proc. of Logic in Computer Science (LICS)*, 387–397.

[16] O. Laurent 1999. Polarized proof-nets and $\lambda\mu$-calculus. Draft

[17] C. H. L. Ong and C. A. Stewart 1997. A Curry-Howard Foundation for functional computation with control. In *Proc. of ACM SIGPLAN-SIGACT Symposium on Principle of Programming Languages (POPL)*, Paris, ACM Press, January.

[18] I. Ogata 1999. Constructive Classical Logic as CPS-calculus. To appear in *International Journal of Foundations of Functional Programming*.

[19] M. Parigot 1992. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proc. of the International Conference on Logic Programming and Automated Reasoning (LPAR)*, St. Petersburg, Lecture Notes in Computer Science 624.

[20] G. D. Plotkin 1975. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science 1*, 125–159.

[21] A. Sabry and M. Felleisen 1993. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation 6(3/4)*, 287–358.

[22] A. Sabry and P. Wadler 1997. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems (TOPLAS) 19(6)*, 916–941.

[23] P. Selinger 1999. Control categories and duality: on the categorical semantics of the $\lambda\mu$-calculus. To appear in *Mathematical Structures in Computer Science*.

[24] C. Urban and G. Bierman 1999. Strong normalization of cut-elimination in classical logic. In *Proc. of Typed Lambda Calculus and Applications (TLCA)*, Lecture Notes in Computer Science 1581.

APPENDIX

A. THE $\lambda\mu$-CALCULUS

The $\lambda\mu$-calculus [19] is an extension of the λ-calculus that deals with multiple conclusions and therefore allows us to account for classical reasoning. Under the Curry-Howard isomorphism, it can be seen as a λ-calculus with control operators, and is indeed equivalent to, say, Felleisen's $\lambda\mathcal{C}$-calculus [11]. For the sake of consistency with our framework, we consider two syntactic categories: the terms and the commands, and, accordingly, two kinds of typing judgements:

Syntax:

$$\begin{array}{l} M ::= x \mid MN \mid \lambda x.M \mid \mu\beta.c \\ c ::= [\alpha]M \end{array}$$

Typing judgements:

$$\Gamma \vdash M : A \,|\, \Delta \qquad c : (\Gamma \vdash \Delta)$$

Typing rules:

$$\frac{}{\Gamma, x : A \vdash x : A \,|\, \Delta}$$

$$\frac{\Gamma \vdash M : A \to B \,|\, \Delta \quad \Gamma \vdash N : A \,|\, \Delta}{\Gamma \vdash MN : B \,|\, \Delta}$$

$$\frac{\Gamma, x : A \vdash M : B \,|\, \Delta}{\Gamma \vdash \lambda x.M : A \to B \,|\, \Delta}$$

$$\frac{c : (\Gamma \vdash \beta : B, \Delta)}{\Gamma \vdash \mu\beta.c : B \,|\, \Delta}$$

$$\frac{\Gamma \vdash M : A \,|\, \alpha : A, \Delta}{[\alpha]M : (\Gamma \vdash \alpha : A, \Delta)}$$

Reduction rules (in call-by-name):

$$\begin{array}{lcl} (\lambda x.M)N & \to & M[x \leftarrow N] \\ (\mu\beta.c)N & \to & \mu\alpha.c[\beta \leftarrow (\alpha, N)] \\ [\alpha](\mu\beta.c) & \to & c[\beta \leftarrow \alpha] \end{array}$$

where substitution is the usual (capture-avoiding) substitution in the first rule and the third rule, while in the second rule one replaces every subterm of c of the form $[\beta]M$ by $[\alpha](MN)$. There is an additional rule, similar in some sense to the η-reduction, which we do not include as a reduction rule (rather, we treat it implicitly as an expansion rule):

$$\mu\alpha.[\alpha]M \quad = \quad M \qquad (\alpha \text{ not free in } M)$$

B. LINEAR DECORATION OF $LKT_{\mu\tilde{\mu}}$ AND $LKQ_{\mu\tilde{\mu}}$

In this section, we complete the work of section 5 by providing translations of $LKT_{\mu\tilde{\mu}}$ and $LKQ_{\mu\tilde{\mu}}$ into linear logic. Arrow types are translated as in [5].

The translation of $LKT_{\mu\tilde{\mu}}$ into linear logic is defined as follows on formulas:

$$X^T = X \quad (A \to B)^T = !?A^T \multimap ?B^T \ .$$

Such a translation which consists only in inserting modalities at some places without any other modification is called a linear decoration.

Proposition B.1.

$$\left.\begin{array}{r} \Gamma \overset{LKT_{\mu\tilde{\mu}}}{\vdash} \Delta \\ \Gamma \overset{LKT_{\mu\tilde{\mu}}}{\vdash} A \,|\, \Delta \\ \Gamma\, ;\, A \overset{LKT_{\mu\tilde{\mu}}}{\vdash} \Delta \\ \Gamma \,|\, A \overset{LKT_{\mu\tilde{\mu}}}{\vdash} \Delta \end{array}\right\} \Longrightarrow \left\{\begin{array}{l} !?\Gamma^T \overset{LL}{\vdash} ?\Delta^T \\ !?\Gamma^T \overset{LL}{\vdash} ?A^T, ?\Delta^T \\ !?\Gamma^T, A^T \overset{LL}{\vdash} ?\Delta^T \\ !?\Gamma^T, !?A^T \overset{LL}{\vdash} ?\Delta^T \end{array}\right.$$

The linear decoration for $LKQ_{\mu\tilde{\mu}}$ is defined as follows:

$$X^Q = X \quad (A \to B)^Q = !A^Q \multimap ?!B^Q \ .$$

Proposition B.2.

$$\left.\begin{array}{r} \Gamma \overset{LKQ_{\mu\tilde{\mu}}}{\vdash} \Delta \\ \Gamma \overset{LKQ_{\mu\tilde{\mu}}}{\vdash} A\, ;\, \Delta \\ \Gamma \overset{LKQ_{\mu\tilde{\mu}}}{\vdash} A \,|\, \Delta \\ \Gamma \,|\, A \overset{LKQ_{\mu\tilde{\mu}}}{\vdash} \Delta \end{array}\right\} \Longrightarrow \left\{\begin{array}{l} !\Gamma^Q \overset{LL}{\vdash} ?!\Delta^Q \\ !\Gamma^Q \overset{LL}{\vdash} A^Q, ?!\Delta^Q \\ !\Gamma^Q \overset{LL}{\vdash} ?!A^Q, ?!\Delta^Q \\ !\Gamma^Q, !A^Q \overset{LL}{\vdash} ?!\Delta^Q \end{array}\right.$$