Duality of computation and sequent calculus: a few more remarks

Hugo Herbelin
INRIA-Saclay
École Polytechnique
F-91128 Palaiseau Cedex
Hugo.Herbelin@inria.fr

Abstract

A succession of works have contributed to the understanding of the computational content of Gentzen-style classical sequent calculus. Especially, it has been shown that the left-right duality of sequent calculus expresses a syntactic duality between programs and their evaluation contexts and that sequent calculus has two dual syntactic restrictions that respectively correspond to call-by-name evaluation and call-by-value evaluation. We propose here an interpretation of the unrestricted syntax of sequent calculus, which, thanks to the use of laziness operators, validates eta-equalities and for which the call-by-name and call-by-value subsystems are obtained by restrictions at the semantic level only. Also, to reason with connectives in sequent calculus, we introduce a purely computational and generic approach of the notion of logical connective.

Introduction

A succession of works have contributed to the understanding of the computational content of Gentzen-style classical sequent calculus since the awareness of the proofs-as-programs correspondence exemplified by Curry [2] and Howard [9] in the context of Hilbert's style logic and natural deduction: Parigot contributed with his μ operator [13] to the understanding of multiple-conclusions sequents, [7] contributed to the understanding of the left introduction rules, and [1] provided an analysis of the symmetry of sequent calculus as a syntactic duality between terms and evaluation contexts and a semantical duality between call-by-name and call-by-value evaluation. Besides this latter line of work that is characterized by a close connection with λ-calculus and slight divergences on the way to deal with the non logical rules of sequent calculus, Danos, Joinet and Schellinx investigated all possible embeddings of Gentzen's LK into linear logic [3], and van Bakel, Lengrand and Lescanne [18], following Lengrand [11], investigated some properties of the underlying computational structure of LK. Relying on [1], Wadler also investigated how to interpret pairs and sums in sequent calculus.

None of these works really cared about η-equalities. The interpretations from [1] and [20] provide with close correspondences with more well studied λ-calculi such as $\lambda\mu$-calculus [8, 20] and they consequently support η-equality but the normal terms in these interpretations capture only a subset of the normal forms of sequent calculus. For the other interpretations, all normal proofs of sequent calculus are captured by normal expressions of the associated computational language, but evaluation of the arguments of connectives is "weak" (e.g. a pair is in normal form even if its arguments are not values) and η-equalities are not validated (see the end of Section 1).

We here address the question of computationally interpreting all normal forms of sequent calculus while still being compatible with η-equalities (such as surjective pairing or the usual η rule of λ-calculus). We achieve this goal by introducing an explicit operator to delay the evaluation of the arguments of the constructors of the calculus (such as the arguments of a pair or the argument of sum injections).

Incidentally, and to provide with a more uniform presentation of sequent calculus with its different connectives, we investigate a generic notion of connective, attacked from the computational perspective, that covers most of standard constructions such as pairs and projections, λ-abstraction and application, sums and case analysis.

1 Review of previous works

In a previous work with Pierre-Louis Curien was introduced the $\overline{\lambda}\mu\tilde{\mu}$-calculus [1]. It is a calculus which computationally interprets a variant of Gentzen's sequent calculus LK in the spirit of the Curry-Howard correspondence between Hilbert-style logic and combinatory logic and between natural deduction and λ-calculus.

The syntax of $\overline{\lambda}\mu\tilde{\mu}$-calculus has three categories of expressions:

Commands	c	$::=$	$\langle v \vert e\rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \lambda x.v$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid v \cdot e$

Commands interpret the cut rule of sequent calculus and they create the opportunity for a term and an evaluation context to interact together. *Terms* interpret right introduction rules of sequent calculus and *evaluation contexts* interpret left introduction rules. These latter categories are dual one of the other. Especially, there are variables for terms, written x, y, ... and variables for evaluation contexts, written α, β, ... The operators $\mu\alpha.c$ and $\tilde{\mu}x.c$ are binders. The operator $\mu\alpha.c$ builds a term which, when in interaction with an evaluation context e, binds e to α and expects to substitute it at the bound occurrences of α. The operator $\tilde{\mu}x.c$ is the evaluation context which, when in interaction with a term v, binds it to x and expects to substitute it at the bound occurrences of x. Finally, $\lambda x.t$ is ordinary abstraction of λ-calculus and $v \cdot e$ is the evaluation context that first applies v to its argument before continuing the evaluation with context e.

The variant of sequent calculus in Curry-Howard correspondence with $\overline{\lambda}\mu\tilde{\mu}$-calculus is called $LK_{\mu\tilde{\mu}}$ and it is shown, simultaneously with its proof-term annotations, in Figure 1. The definition of this sequent calculus relies on three kinds of sequents. In each kind of sequent, the contexts, written Γ, Γ', ... on the left-hand side and Δ, Δ', ... on the right-hand side are finite maps from names to formulas. The first kind of sequent, $\Gamma \vdash \Delta$ interprets commands, and the interpretation of $\Gamma \vdash \Delta$ as a typing judgments for c is written $c : (\Gamma \vdash \Delta)$. The second kind of sequent, $\Gamma \vdash A \mid \Delta$, has a distinguished formula on the right-hand side and it corresponds to the typing of a term, the distinguished formula being the type of the term. Symmetrically, sequents of the third kind, $\Gamma \mid A \vdash \Delta$, corresponds to the typing of an evaluation context expecting to interact with a term of type A. The sequent calculus has two axiom rules, one for introducing each kind of variable and the operators $\mu\alpha.c$ and $\tilde{\mu}x.c$ act as focusing rules. Note that weakening rules are

$$Ax_R \; \frac{}{\Gamma, x : A \vdash x : A \mid \Delta} \qquad Ax_L \; \frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Delta}$$

$$\mu \; \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta} \qquad \tilde{\mu} \; \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c : A \vdash \Delta}$$

$$Cut \; \frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : A \vdash \Delta}{c : (\Gamma \vdash \Delta)}$$

$$\to_R \frac{\Gamma, x : A \vdash t : B \mid \Delta}{\Gamma \vdash \lambda x.t : A \to B \mid \Delta} \quad \to_L \frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid v \cdot e : A \to B \vdash \Delta}$$

Figure 1. $LK_{\mu\tilde{\mu}}$**: a system of simple types for** $\overline{\lambda}\mu\tilde{\mu}$

treated at the level of axiom rules, as is it common in type systems and that contraction rules are simulated by a cut with an axiom. Contexts are treated additively.

Due to the absence of explicit contraction and weakening rules and to the additive management of contexts, $LK_{\mu\tilde{\mu}}$ also admits a presentation as a derivation of formulas [8], with no explicit representation of the context, but using *discharge of assumption* for binders as it was originally done for natural deduction in Gentzen [5]. This is given in Figure 1. Since $LK_{\mu\tilde{\mu}}$ has three kinds of sequents, so has its presentation with implicit context. The three kinds of derivation are $\vdash A$, standing for A is true, $A \vdash$ standing for A is false, and $\perp\!\!\!\perp$ standing for a contradiction. For instance, here follows a derivation of Peirce law where each discharge is annotated by a number. The corresponding $\overline{\lambda}\mu\tilde{\mu}$-term is $\lambda x.\mu\alpha.\langle x \vert (\lambda y.\mu\beta.\langle y \vert \alpha\rangle) \cdot \alpha\rangle$.

$$\dfrac{\dfrac{\dfrac{[\vdash (A \to B) \to A]_2 \qquad \dfrac{\dfrac{\dfrac{\dfrac{[\vdash A]_3 \qquad [A \vdash]_1}{\perp\!\!\!\perp}}{\vdash B}\,4}{\vdash A \to B}\,3 \qquad [A \vdash]_1}{(A \to B) \to A \vdash}}{\perp\!\!\!\perp}}{\vdash A}\,1}{\vdash ((A \to B) \to A) \to A}\,2$$

There are three rules of reduction for $\overline{\lambda}\mu\tilde{\mu}$ and seen from the perspective of $LK_{\mu\tilde{\mu}}$, these three rules imple-

$$\mu\ \dfrac{\begin{array}{c}[A\vdash]\\ \vdots\\ \perp\!\!\!\perp\end{array}}{\vdash A} \qquad\qquad \tilde{\mu}\ \dfrac{\begin{array}{c}[\vdash A]\\ \vdots\\ \perp\!\!\!\perp\end{array}}{A\vdash}$$

$$Cut\ \dfrac{\vdash A \qquad A\vdash}{\perp\!\!\!\perp}$$

$$\rightarrow_R\ \dfrac{\begin{array}{c}[\vdash A]\\ \vdots\\ \vdash B\end{array}}{\vdash A\rightarrow B} \qquad\qquad \rightarrow_L\ \dfrac{\vdash A \qquad B\vdash}{A\rightarrow B\vdash}$$

Figure 2. Presentation of $LK_{\mu\tilde{\mu}}$ with implicit context

ment elimination of cuts (to the exception of cuts that involves an axiom and no focusing rule since these latter cuts are used to simulate contraction – this restriction does not preclude the subformula property which still holds):

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

where $c[\alpha\leftarrow e]$ and $c[x\leftarrow v]$ are capture-free substitutions.

The above system is non-deterministic and non-confluent: The first two rules create a critical pair and it is shown in [1] that giving priority to (μ) leads to a call-by-value language and giving priority to ($\tilde{\mu}$) leads to a call-by-name language: in the critical pair $\langle\mu\alpha.c|\tilde{\mu}x.c\rangle$, it is a call-by-name evaluation discipline if the evaluation context binds its argument as it stands to x, it is a call-by-value evaluation discipline if the evaluation context expects first its argument to be evaluated before binding it to x, which means yielding its priority to the term.

To the contrary of, say, natural deduction and λ-calculus, abstraction and the constructor of evaluation context that stands for application are not necessary for the rest of the calculus to be meaningful. The resulting subsystem, called system $\mu\tilde{\mu}$, has been studied in [8].

Two restrictions of the $\overline{\lambda}\mu\tilde{\mu}$-calculus are given in [1]. In both cases, the constructor of evaluation contexts that stands for application is constrained. More precisely, the restricted syntax for call-by-name is

$$\begin{array}{ll}
c ::= \langle v|e\rangle & E ::= \alpha \,\|\, v\cdot E\\
v ::= x \,\|\, \mu\beta.c \,\|\, \lambda x.v & e ::= \tilde{\mu}x.c \,\|\, E
\end{array}$$

and the restricted syntax for call-by-value is

$$\begin{array}{ll}
c ::= \langle v|e\rangle & V ::= x \,\|\, \lambda x.v\\
e ::= \alpha \,\|\, \tilde{\mu}x.c \,\|\, V\cdot e & v ::= \mu\beta.c \,\|\, V
\end{array}$$

In 2003, Wadler presented a variant of $\overline{\lambda}\mu\tilde{\mu}$ based on disjunction, conjunction and negation instead of implication [19]. The syntax of the calculus is:

Statements	S	$::=$	$M\bullet K$
Terms	M,N	$::=$	$(S).\alpha \mid x \mid \langle M,N\rangle \mid [K]\text{not}$
		$\mid$	$\langle M\rangle\text{inl} \mid \langle M\rangle\text{inr}$
Coterms	K,L	$::=$	$x.(S) \mid \alpha \mid [K,L] \mid \text{not}\langle M\rangle$
		$\mid$	$\text{fst}[K] \mid \text{snd}[L]$

where $(S).\alpha$ matches $\mu\alpha.c$ and $x.(S)$ matches $\tilde{\mu}x.c$.

Two dual systems of reduction were defined, one for call-by-name evaluation and the other for call-by-value evaluation. The definition of call-by-name reduction relies on a notion of covalue and of coterm context:

Covalues	P,Q	$::=$	$\alpha \mid [P,Q] \mid \text{not}\langle M\rangle$
		$\mid$	$\text{fst}[P] \mid \text{snd}[Q]$
Coterm contexts	F	$::=$	$[\{\},K] \mid [P,\{\}]$
		$\mid$	$\text{fst}[\{\}] \,\|\, \text{snd}[\{\}]$

$$\begin{array}{llll}
(\beta\&) & \langle M,N\rangle\bullet\text{fst}[P] & \rightarrow_n & M\bullet P\\
(\beta\&) & \langle M,N\rangle\bullet\text{snd}[Q] & \rightarrow_n & N\bullet Q\\
(\beta\vee) & \langle M\rangle\text{inl}\bullet[P,Q] & \rightarrow_n & M\bullet P\\
(\beta\vee) & \langle N\rangle\text{inr}\bullet[P,Q] & \rightarrow_n & N\bullet Q\\
(\beta\neg) & [K]\text{not}\bullet\text{not}\langle M\rangle & \rightarrow_n & M\bullet K\\
(\beta L) & M\bullet x.(S) & \rightarrow_n & S\{M/x\}\\
(\beta R) & (S).\alpha\bullet P & \rightarrow_n & S\{P/\alpha\}\\
(\varsigma) & F\{K\} & \rightarrow_n & y.((y\bullet F\{\alpha\}).\alpha\bullet K)
\end{array}$$

Observe that the rule (ς) is a rule that forces strong evaluation of the argument of fst[ ], snd[ ] and [,] (which are the dual of the sum injections and of pairing in call-by-value). Due to this strong evaluation discipline, it can be shown that the normal forms of the call-by-name subcalculus of Wadler's calculus are generated by the following syntax:

Statements	S	$::=$	$x\bullet P \mid W\bullet\alpha$
Weak terms	W	$::=$	$\langle M,N\rangle \mid [K]\text{not}$
		$\mid$	$\langle M\rangle\text{inl} \mid \langle M\rangle\text{inr}$
Terms	M,N	$::=$	$(S).\alpha \mid x \mid W$
Covalues	P,Q	$::=$	$\alpha \mid [P,Q] \mid \text{not}\langle M\rangle$
		$\mid$	$\text{fst}[P] \mid \text{snd}[Q]$

Obviously, the same can be said by duality of the call-by-value fragment of Wadler's calculus.

In both [1] and [19], the call-by-name and call-by-value subcalculi are then obtained by restricting not only the reduction rules but also the syntax.

If one does not restrict the syntax of $\overline{\lambda}\mu\tilde{\mu}$-calculus, the η rule for abstraction, which is stated as $\lambda y.\mu\alpha.\langle v|y \cdot \alpha\rangle = v$ for x and α not free in v, does not hold in the call-by-name fragment. Indeed, we have the following divergence (_ denotes a variable that binds no occurrence):

$$\begin{array}{ccc}
\langle \lambda y.\mu\alpha.\langle \mu\_.c_1|y \cdot \alpha\rangle|v \cdot \tilde{\mu}\_.c_2\rangle & = & \langle \mu\_.c_1|v \cdot \tilde{\mu}\_.c_2\rangle \\
\downarrow_n & & \downarrow_n \\
\langle v|\tilde{\mu}y.\langle \mu\alpha.\langle \mu\_.c_1|y \cdot \alpha\rangle|\tilde{\mu}\_.c_2\rangle\rangle & & c_2 \\
\downarrow_n^\star & & \\
c_1 & &
\end{array}$$

The $\lambda\xi$-calculus is the re-interpretation of a non-deterministic cut-elimination procedure for LK formalized by Urban [17] as an untyped language admitting two dual call-by-name and call-by-value restrictions (see Lengrand [11]). The $\lambda\xi$-calculus (which has been renamed in more recent works into $\mathcal{X}$ [18]) interprets all normal proofs of LK but for the same reason as above, it does not support η-equality.

The sequent calculus LK^{tq}, defined in Danos, Joinet and Schellinx [3], provides with a computational analysis of LK through linear logic. It does not validate η-equality either.

2 Computational connectives

The addition of a connective to the system $\mu\tilde{\mu}$ is a modular operation. New connectives are obtained by providing syntax rules for forming terms, syntax rules for forming evaluation contexts and reduction rules describing the interaction between a term and an evaluation context of the same connective. For instance, implication is characterized by the constructions $\lambda x.t$ and $v \cdot e$ and the reduction rule $(\rightarrow)$, while the conjunction of Wadler's calculus is characterized by the constructions $\langle M, N\rangle$, fst$[K]$, snd$[K]$ and the two reduction rules named $(\beta\&)$.

Especially, connectives are characterized by their constructions and they are no longer seen a constructor of formula or type. This suggests to introduce a notion of purely computational connective. A *computational connective* (or simply *connective* hereafter) is the pair of a sign, called the *sign* of the connective, and of a family of finite sequences of signs, at most one sign of each sequence being possibly distinguished and called *dotted*. If $\mathcal{S}$ is a connective, its domain, written $\mathcal{D}_\mathcal{S}$, is the domain of the family and the cardinal of the connective is defined to be the cardinal of $\mathcal{D}_\mathcal{S}$. The signs are either $+$ or $-$, referring respectively the class of terms and the class of evaluation contexts. We use the notation $\{s^1_1 \ldots s^1_{n_1}, \ldots, s^q_{n_q}\}s$ for representing connectives of finite cardinal where each s is a sign and, in each sequence, at most one s^i_j is possibly dotted. For a connective of infinite cardinal, we write $\{s^i_1 \ldots s^i_{n_i} | i \in \mathcal{D}\}s$ or also $\{s^i_1 \ldots s^i_{n_i}\}_{i \in \mathcal{D}}s$. This kind of notations is called a *signature*. If s is $+$, we say that the connective is *positive*, otherwise, we say that it is *negative*.

To each connective of cardinality α is associated a family of *general constructors* of cardinality $\alpha + 1$ and a family of reduction rules of cardinality α. The general constructors are split into a family of α constructors called *irreversible constructors* (or simply *constructors*) and a single constructor called *reversible constructor* (or also *co-constructor*). If the connective is positive, the irreversible constructors are constructors of terms and the reversible constructor is a constructor of evaluation contexts. If the connective is negative, the α irreversible constructors are constructors of evaluation contexts and the co-constructor is a constructor of terms.

Each irreversible constructors of a connective takes as many arguments as the length of the sequence in the corresponding component of the family, and each of these arguments is a term if the corresponding sign in the sequence is $+$ and an evaluation context if the corresponding sign in the sequence is $-$. For $i \in \mathcal{D}_\mathcal{S}$, we write $\iota_i(w^{S^i_1}, ..., w^{S^i_{n_i}})$ (resp. $\pi_i[w^{S^i_1}, ..., w^{S^i_{n_i}}]$) for the irreversible constructor of a positive (resp. negative) connective $\mathcal{S}$ where S^i_j is the jth sign of the sequence of index i and w^+ and w^- are respectively a term and an evaluation context. In case the connective is of cardinal 1, we simply abbreviate $\iota_1(w^{s_1}, ..., w^{s_n})$ into $(w^{s_1}, ..., w^{s_n})$ and $\pi_1[t^{s_1}, ..., t^{s_n}]$ into $[t^{s_1}, ..., t^{s_n}]$.

The unique reversible constructor of a connective has as many arguments as the cardinality of the family of sequences and each of these arguments is obtained by binding as many variables as the number of non-dotted elements in the corresponding sequence. If there is no dotted element, the multiple binders takes a command as argument, otherwise, it takes as argument an evaluation context if the dotted sign is $+$ or a term if the dotted sign is $-$. In case no sign is dotted at all, we write $[\tilde{\mu}(a^{S^1_1}, ..., a^{S^1_{n_1}}).c_1, ..., \tilde{\mu}(a^{S^q_1}, ..., a^{S^q_{n_q}}).c_q]$ (resp. $(\mu[a^{S^1_1}, ..., a^{S^1_{n_1}}].c_1, ..., \mu[a^{S^q_1}, ..., a^{S^q_{n_q}}].c_q)$) for the reversible constructor of a positive (resp. negative) connective of finite cardinal. We write $[\tilde{\mu}(a^{S^i_1}, ..., a^{S^i_{n_i}}).c_i]_i$ (resp. $(\mu[a^{S^i_1}, ..., a^{S^i_{n_i}}].c_i)_i$) in case the cardinal is infinite. In each case, n_i is the length of the sequence associated to i in the domain of the connective and each a_i is a term variable or an evaluation context variable depending on whether the corresponding sign in the sequence is $+$ or $-$ respectively. If the l^{th} sign is dotted in the sequence of order i, the corresponding subexpressions $\mu[a_1,...,a_n].c$ (resp. $\tilde{\mu}(a_1,...,a_n).c$) is replaced by $\tilde{\mu}(a_1,...,a_{l-1},a_{l+1},\ldots,a_n).e$ (resp. $\mu[a_1,...,a_{l-1},a_{l+1},\ldots,a_n].e$) if a_l is a term variable or $\tilde{\mu}(a_1,...,a_{l-1},a_{l+1},\ldots,a_n).v$ (resp. $\mu[a_1,...,a_{l-1},a_{l+1},\ldots,a_n].v$) if a_l is an evaluation context variable. In case the connective is of cardinal 1 and the single sequence not made of a dotted single sign, we may drop the surrounding parentheses or brackets.

We now give the reduction rules for computational connectives. There are two variants, depending on the sign of the connective. For non-dotted connective, the rules are given in Figure 2.

In case the sequence of index i_0 has a dot, when the associated expression is $\tilde{\mu}(a_1,...,a_{l-1},a_{l+1},...,a_n).e$ or $\mu(a_1,...,a_{l-1},a_{l+1},...,a_n).e$ instead of $\tilde{\mu}(a_1,...,a_n).c$ or $\mu(a_1,...,a_n).c$ then the right-hand side of the rule is $\langle w_l | e[a_1,...,a_{l-1},a_{l+1},...,a_n \leftarrow w_1,...,w_{l-1},w_{l+1},...,w_n]\rangle$, and dually when the expression associated to the dot sequence ends with some term v. Notice then that dotted sign are here only for convenience so as to obtain shortest notation. Indeed, the right-hand side of the reduction rules for a connective with dots and for the same connective expressed without any dot are the same. We give some examples.

Additive disjunction We call computational additive disjunction and write $\vee_a$ for the connective characterized by $\{\dot{+},\dot{+}\}+$ (with domain $\{1,2\}$). It introduces the following constructors and reduction rules:

$$\begin{array}{lll} \text{Terms} & v & ::= \ \iota_1(v) \mid \iota_2(v) \\ \text{Evaluation contexts} & e & ::= \ [e,e] \end{array}$$

$$\begin{array}{llll} (\vee_a^1) & \langle \iota_1(v_1) | [e_1,e_2]\rangle & \to & \langle v_1 | e_1\rangle \\ (\vee_a^2) & \langle \iota_2(v_2) | [e_1,e_2]\rangle & \to & \langle v_2 | e_2\rangle \end{array}$$

Additive conjunction We call computational additive conjunction and write $\wedge_a$ for the connective characterized by $\{\dot{-},\dot{-}\}-$ (with domain $\{1,2\}$). It introduces the following constructors and reduction rules:

$$\begin{array}{lll} \text{Terms} & v & ::= \ (v,v) \\ \text{Evaluation contexts} & e & ::= \ \pi_1[e] \mid \pi_2[v] \end{array}$$

$$\begin{array}{llll} (\wedge_a^1) & \langle (v_1,v_2) | \pi_1[e_1]\rangle & \to & \langle v_1 | e_1\rangle \\ (\wedge_a^2) & \langle (v_1,v_2) | \pi_2[e_2]\rangle & \to & \langle v_1 | e_2\rangle \end{array}$$

Multiplicative conjunction We call computational multiplicative conjunction and write $\wedge_m$ for the connective characterized by $\{++\}+$ (with domain $\{1\}$). It introduces the following constructors and reduction rules:

$$\begin{array}{lll} \text{Terms} & v & ::= \ (v,v) \\ \text{Evaluation contexts} & e & ::= \ \tilde{\mu}(x,y).c \end{array}$$

$$(\wedge_m) \quad \langle (v_1,v_2) | \tilde{\mu}(x_1,x_2).c\rangle \ \to \ c[x_1 \leftarrow v_1][x_2 \leftarrow v_2]$$

Multiplicative disjunction We call computational multiplicative disjunction and write $\vee_m$ for the connective characterized by $\{--\}-$ (with domain $\{1\}$). It introduces the following constructors and reduction rules:

$$\begin{array}{lll} \text{Terms} & v & ::= \ \mu[\alpha,\beta].c \\ \text{Evaluation contexts} & e & ::= \ [e,e] \end{array}$$

$$(\vee_m) \quad \langle \mu[\alpha_1,\alpha_2].c | [e_1,e_2]\rangle \ \to \ c[\alpha_1 \leftarrow e_1][\alpha_2 \leftarrow e_2]$$

Implication Computational implication, written $\to$ is characterized by $\{+\dot{-}\}-$ (with domain $\{1\}$). It introduces the following constructors and reduction rules:

$$\begin{array}{lll} \text{Terms} & v & ::= \ \lambda x.t \\ \text{Evaluation contexts} & e & ::= \ v \cdot e \end{array}$$

$$(\to) \quad \langle \lambda x.t | v \cdot e\rangle \ \to \ \langle t[x \leftarrow v] | e\rangle$$

where $\lambda x.t$ is just another writing for the standardized notation $\mu[x].t$ and $v \cdot e$ an alternative writing for the standardized notation $[v,e]$.

Negation There are two isomorphic forms of computational negation. The positive negation is characterized by $\{\dot{-}\}+$ and the negative negation by $\{\dot{+}\}-$ (both with domain $\{1\}$). The constructors and reduction rules of positive negation are:

$$\begin{array}{lll} \text{Terms} & v & ::= \ (e) \\ \text{Evaluation contexts} & e & ::= \ [v] \end{array}$$

$$(\neg^+) \quad \langle (e) | [v]\rangle \ \to \ \langle v | e\rangle$$

Relation with Girard's ludics connectives

Computational connections can be seen as a two-sided variant of Girard's notion of synthetic connective [6] with an emphasis on the purely computational aspect of the connective. A (finite) synthetic connective of the form $\{\{n_1^1,...,n_{p_1}^1\},...,\{n_1^q,...,n_{p_q}^q\}\}$ is interpreted by the computational connective

$$\{\overbrace{+\ldots+}^{p_1 \text{ times}},...,\overbrace{+\ldots+}^{p_q \text{ times}}\}+$$

$$
\begin{array}{lll}
(\mathcal{S}) & \langle \iota^{\mathcal{S}}_{i_0}(w^{S^{i_0}}{}_1, ..., w^{S^{i_0}}{}_{n_{i_0}}) \| [\tilde{\mu}(a^{S^i}{}_1, ..., a^{S^i}{}_{n_i}).c_i]_i\rangle \xrightarrow{h} & c_{i_0}[a^{S^i}{}_1, ..., a^{S^i}{}_{n_i} \leftarrow w^{S^{i_0}}{}_1, ..., w^{S^{i_0}}{}_{n_{i_0}}] \\
(\mathcal{T}) & \langle (\mu[a^{T^i}{}_1, ..., a^{T^i}{}_{n_i}].c_i)_i | \pi^{\mathcal{T}}_{i_0}[w^{T^{i_0}}{}_1, ..., w^{T^{i_0}}{}_{n_{i_0}}]\rangle \xrightarrow{h} & c_{i_0}[a^{T^i}{}_1, ..., a^{T^i}{}_{n_i} \leftarrow w^{T^{i_0}}{}_1, ..., w^{T^{i_0}}{}_{n_{i_0}}]
\end{array}
$$

Figure 3. Generic reduction rules for non-dotted positive and negative computational connectives

of domain $\mathcal{D} \triangleq \{\{n^1_1, ..., n^1_{p_1}\}, ..., \{n^q_1, ..., n^q_{p_q}\}\}$. More generally, the infinite form of synthetic connectives used in ludics' designs [6] is interpreted by the computational connective

$$\{\ldots, \overbrace{+\ldots+}^{|I| \text{ times}}, \ldots\}_I+$$

where I ranges over finite subsets of $\mathbb{N}$ and $|I|$ is the cardinality of I. Observe incidentally that only the cardinal of I is relevant from the computational point of view and that we also need arbitrary many distinct copies of a subset of cardinal n. Henceforth, an alternative definition of ludics' connective that computationally behaves the same can be obtained if one replaces the indexation over the subset of $\mathbb{N}$ by an indexation over the pair of a finite cardinality and of a multiplicity number. Concretely, this corresponds to the computational connective

$$\{\ldots, \overbrace{+\ldots+}^{p \text{ times}}, \ldots\}_{\{n,p\}}+$$

where n and p range over $\mathbb{N}$. Note that to be computationally strictly equivalent to ludics' connective, p must not range over $\mathbb{N}$ when n is 0 since there is only one empty subset of $\mathbb{N}$. Making p range over $\mathbb{N}$ even when n is 0 provides with a generalization that allows for instance to represent the connective $1 \oplus 1$, what cannot be captured in ludics.

Computational connectives not captured by the general definition

Observe that some other forms of constructions escape this definition. For instance, the if-and-only-if connective defined in Raghunandan and Summers [14] is characterized by

$$
\begin{array}{rcl}
V & ::= & (\mu(x,\beta).c_1, \mu(y,\alpha).c_2) \\
E & ::= & [\tilde{\mu}(x,y).c_1, \tilde{\mu}(\alpha,\beta).c_2]
\end{array}
$$

with reduction either (we overline the inner command to better visualize the nested structure of the expressions):

$$
\begin{array}{l}
\langle(\mu(x,\beta).c_1, \mu(y,\alpha).c_2)\|[\tilde{\mu}(x',y').c'_1, \tilde{\mu}(\alpha',\beta').c'_2]\rangle \\
\rightarrow \langle\mu\beta.\overline{\langle\mu\alpha'.c'_2[\beta'\leftarrow\beta]\|\tilde{\mu}x.c_1\rangle}|\tilde{\mu}y.\overline{\langle\mu\alpha.c_2|\tilde{\mu}x'.c'_1[y'\leftarrow y]\rangle}\rangle
\end{array}
$$

or:

$$
\begin{array}{l}
\langle(\mu(x,\beta).c_1, \mu(y,\alpha).c_2)\|[\tilde{\mu}(x',y').c'_1, \tilde{\mu}(\alpha',\beta').c'_2]\rangle \\
\rightarrow \langle\mu\alpha.\overline{\langle\mu\beta'.c'_2[\alpha'\leftarrow\alpha]\|\tilde{\mu}y.c_2\rangle}|\tilde{\mu}x.\overline{\langle\mu\beta.c_1|\tilde{\mu}y'.c'_1[x'\leftarrow x]\rangle}\rangle
\end{array}
$$

Other connectives not captured by our definition are the dependent product type and dependent sum type of type theory.

Eta-equalities for computational connectives

Eta-equalities can be formulated in a generic way on computational connectives. Eta-equalities expresses an observational property: any variable in the category of the opposite sign of the connective behaves as if it were an expression constructed from a reversible constructor. We have two such η-equalities depending on the sign of the connective:

$$
\begin{array}{llcl}
(\eta_{\mathcal{S}}) & \alpha & = & [\tilde{\mu}(a^{S^i}{}_1, ..., a^{S^i}{}_{n_i}).\langle\iota_i(a^{S^i}{}_1, ..., a^{S^i}{}_{n_i})|\alpha\rangle]_i \\
(\eta_{\mathcal{T}}) & x & = & (\mu[a^{T^i}{}_1, ..., a^{T^i}{}_{n_i}].\langle x|\pi_i[a^{T^i}{}_1, ..., a^{T^i}{}_{n_i}]\rangle)
\end{array}
$$

where, in case the lth sign is dotted in the sequence of index i, the expression fragment $\tilde{\mu}(a^{S^i}{}_1, ..., a^{S^i}{}_{n_i})$ (resp. $\mu[a^{T^i}{}_1, ..., a^{T^i}{}_{n_i}]$) is replaced by $\tilde{\mu}(a^{S^i}{}_1, ..., a^{S^i}{}_{l-1}, a^{S^i}{}_{l+1}, ..., a^{S^i}{}_{n_i}).\mu^+ a^{S^i}{}_l$ (resp. $\mu(a^{T^i}{}_1, ..., a^{T^i}{}_{l-1}, a^{T^i}{}_{l+1}, ..., a^{T^i}{}_{n_i}).\mu^+ a^{T^i}{}_l$) with μ^+ being μ is S^i_l is negative and $\tilde{\mu}$ otherwise.

As examples, we give the η-equality for the connectives considered above.

$$
\begin{array}{llcl}
(\eta_{\vee_a}) & \alpha & = & [\tilde{\mu}x_1.\langle\iota_1(x_1)|\alpha\rangle, \tilde{\mu}x_2.\langle\iota(x_2)|\alpha\rangle] \\
(\eta_{\vee_m}) & x & = & \mu[\alpha_1, \alpha_2].\langle x|[\alpha_1, \alpha_2]\rangle \\
(\eta_{\wedge_a}) & x & = & (\mu\alpha_1.\langle x|\pi_1[\alpha]\rangle, \mu\alpha_2.\langle x|\pi_2[\alpha_2]\rangle] \\
(\eta_{\wedge_m}) & \alpha & = & \tilde{\mu}(x_1, x_2).\langle(x_1, x_2)|\alpha\rangle \\
(\eta_{\rightarrow}) & x & = & \lambda y.\mu\alpha.\langle x|y \cdot \alpha\rangle \\
(\eta_{\neg^+}) & \alpha & = & [\tilde{\mu}x.\langle x|\alpha\rangle]
\end{array}
$$

3 System L

We are now ready to define an extension of system $\mu\tilde{\mu}$ that interprets all normal forms of sequent calculus, which still admits two confluent call-by-name and call-by-value restrictions and which restrictions still interpret all normal forms of sequent calculus and additionally validate η-equalities. This system is called system L (by reference to Gentzen's *Logistischen Kalküle* LJ and LK). Compared to system $\mu\tilde{\mu}$ and its extensions, system L has laziness operators ("quotes") whose purpose is to delay evaluation. We formulate the calculus with a generic positive computational connective and a generic negative connective without any dotted sign, but it could be formulated with any arbitrary collection of computational connectives, possibly with dots, such as, say, implication, multiplicative disjunction, additive conjunction, etc. It is the use of quotes in the arguments of the connective that allows the η-reduction to hold. The syntax and reduction semantics of system L are given in Figure 3. The notation w is used to denote weak expressions, i.e. w^+ is W and w^- is F. Standard notions of calculi with binders such as closed expressions, free variables, bound variables, etc. apply to system L.

Like system $\mu\tilde{\mu}$, the reduction system has the critical pair $\langle \mu\alpha..c|\tilde{\mu}x.c'\rangle$ but it also has the new critical pair $\langle \acute{}v|\acute{}e\rangle$. Like for system $\mu\tilde{\mu}$ and its extensions, we can consider two confluent restrictions that respectively correspond to a call-by-name and a call-by-value restriction. We only present the call-by-name restriction, knowing that the call-by-value restriction is dual.

3.1 Call-by-name system L

Call-by-name system L, shortly L_n, is obtained by giving priority to the right-hand side in the command $\langle \mu\alpha..c|\tilde{\mu}x.c'\rangle$, and to the left-hand side in the command $\langle \acute{}v|\acute{}e\rangle$. This is obtained by restricting e to be a weak evaluation context in (μ) and by restricting W to be a value in $('_R)$. As a consequence of the first restriction, evaluation context variables can only be substituted by weak evaluation contexts and as a consequence of the second restriction, quoted terms behave the same as non quoted terms with respect to interaction.

This suggests to identify terms and quoted terms and then to merge the categories of terms and weak terms and the categories of linear evaluation contexts and weak evaluation contexts together. We then arrive to the reformulation of the syntax and semantics given in Figure 3.1 where now, w^+ is v (since v and W have been merged) and w^- is E (since F and E have been merged). The reduction $\rightarrow_n$ is defined as the congruence of $\overset{h}{\rightarrow}$ and the reduction $\overset{*}{\rightarrow}_n$ as the reflexive-transitive closure of $\rightarrow_n$.

The reduction system can be embedded as a Higher-order Rewriting System (HRS). It has no critical pairs and is hence confluent (for the theory of HRS, see Nipkow [12]).

Proposition 1 *The reduction rules of system L_n are confluent.*

As a corollary, system L_n, whatever connectives it is equipped with, is consistent, in the sense that it does not identify all expressions (take for instance $\langle x|\alpha\rangle$ and $\langle y|\alpha\rangle$ which have no redex and are distinct as soon as x and y are themselves distinct).

3.2 Normal forms and operational completeness

A command is said to be *in weak-head normal form* if it has the form $\langle x|E\rangle$ or $\langle V|\alpha\rangle$. An expression of L_n is *syntactically in normal form* (or simply *normal*) if all commands occurring in the expression are in weak-head normal form. Our notions of normal form and reduction match in the following sense:

Proposition 2 *An expression is syntactically in normal form iff it has no redex.*

Based on typing, we could in fact assert a stronger statement of operational completeness by observing that typed normal terms satisfy the subtype property (i.e. any type occurring in the derivation of a normal term is a subtype of a type occurring in the end sequent of the typing derivation).

3.3 Eta-equalities

A *call-by-name* substitution is a multiple substitution of evaluation contexts variables by weak evaluation contexts and of term variables by arbitrary terms.

We say that two normal forms v_1 and v_2 (resp. e_1 and e_2) are separable in L_n if for all commands c_1 and c_2, there is a call-by-name substitution ρ and an evaluation context e (resp. a term v) such that $\langle v_1[\rho]|e\rangle \overset{*}{\rightarrow}_n c_1$ and $\langle v_2[\rho]|e\rangle \overset{*}{\rightarrow}_n c_2$ (resp. $\langle v|e_1[\rho]\rangle \overset{*}{\rightarrow}_n c_1$ and $\langle v|e_2[\rho]\rangle \overset{*}{\rightarrow}_n c_2$). We say that an equation $v_1 = v_2$ (resp. $e_1 = e_2$) between normal expressions with free variables *belongs to the observational closure of* $\rightarrow_n$ if v_1 and v_2 (resp. e_1 and e_2) are not separable.

Collecting the observational equalities that are not already captured by the reduction system is a priori not an easy task. At least, the following equalities can be proved to be part of the observational closure of $\rightarrow_n$.

$$\begin{array}{llcl}
\text{Commands} & c & ::= & \langle v | e \rangle \\
\text{Values} & V & ::= & \ldots \mid \iota_i^{\mathcal{S}}(w^{S^i_1}, ..., w^{S^i_{n_i}}) \mid \ldots \mid (\mu[a^{T^i_1}, ..., a^{T^i_{n_i}}].c)_i \\
\text{Weak terms} & W & ::= & \acute{}v \mid V \\
\text{Terms} & v & ::= & \mu\alpha.c \mid x \mid W \\
\text{Linear eval. ctx} & E & ::= & \ldots \mid \pi_i^{\mathcal{T}}[w^{T^i_1}, ..., w^{T^i_{n_i}}] \mid \ldots \mid [\tilde{\mu}(a^{S^i_1}, ..., a^{S^i_{n_i}}).c]_i \\
\text{Weak eval. ctx} & F & ::= & \acute{}e \mid E \\
\text{Eval. contexts} & e & ::= & \tilde{\mu}x.c \mid \alpha \mid F
\end{array}$$

$$\begin{array}{llcl}
(\mu) & \langle \mu\alpha.c | e \rangle & \overset{h}{\to} & c[\alpha \leftarrow e] \\
(\tilde{\mu}) & \langle v | \tilde{\mu}x.c \rangle & \overset{h}{\to} & c[x \leftarrow v] \\
(\acute{}_L) & \langle \acute{}v | F \rangle & \overset{h}{\to} & \langle v | F \rangle \\
(\acute{}_R) & \langle W | \acute{}e \rangle & \overset{h}{\to} & \langle W | e \rangle \\
(\mathcal{S}) & \langle \iota_{i_0}^{\mathcal{S}}(w^{S^{i_0}_1}, ..., w^{S^{i_0}_{n_{i_0}}}) | [\tilde{\mu}(a^{S^i_1}, ..., a^{S^i_{n_i}}).c_i]_i \rangle & \overset{h}{\to} & c_{i_0}[a^{S^i_1}, ..., a^{S^i_{n_i}} \leftarrow w^{S^{i_0}_1}, ..., w^{S^{i_0}_{n_{i_0}}}] \\
(\mathcal{T}) & \langle (\mu[a^{T^i_1}, ..., a^{T^i_{n_i}}].c_i)_i | \pi_{i_0}^{\mathcal{T}}[w^{T^{i_0}_1}, ..., w^{T^{i_0}_{n_{i_0}}}] \rangle & \overset{h}{\to} & c_{i_0}[a^{T^i_1}, ..., a^{T^i_{n_i}} \leftarrow w^{T^{i_0}_1}, ..., w^{T^{i_0}_{n_{i_0}}}]
\end{array}$$

Figure 4. Syntax and reduction semantics of system L

$$\begin{array}{llcl}
\text{Commands} & c & ::= & \langle v | e \rangle \\
\text{Values} & V & ::= & \ldots \mid \iota_i^{\mathcal{S}}(w^{S^i_1}, ..., w^{S^i_{n_i}}) \mid \ldots \mid (\mu[a^{T^i_1}, ..., a^{T^i_{n_i}}].c)_i \\
\text{Terms} & v & ::= & \mu\alpha.c \mid x \mid V \\
\text{Weak eval. contexts} & E & ::= & \alpha \mid \acute{}e \mid \ldots \mid \pi_i^{\mathcal{T}}[w^{T^i_1}, ..., w^{T^i_{n_i}}] \mid \ldots \mid [\tilde{\mu}(a^{S^i_1}, ..., a^{S^i_{n_i}}).c]_i \\
\text{Evaluation contexts} & e & ::= & \tilde{\mu}x.c \mid E
\end{array}$$

$$\begin{array}{llcl}
(\mu) & \langle \mu\alpha.c | E \rangle & \overset{h}{\to}_n & c[\alpha \leftarrow E] \\
(\tilde{\mu}) & \langle v | \tilde{\mu}x.c \rangle & \overset{h}{\to}_n & c[x \leftarrow v] \\
(\acute{}) & \langle V | \acute{}e \rangle & \overset{h}{\to}_n & \langle V | e \rangle \\
(\mathcal{S}) & \langle \iota_{i_0}^{\mathcal{S}}(w^{S^{i_0}_1}, ..., w^{S^{i_0}_{n_{i_0}}}) | [\tilde{\mu}(a^{S^i_1}, ..., a^{S^i_{n_i}}).c_i]_i \rangle & \overset{h}{\to}_n & c_{i_0}[a^{S^i_1}, ..., a^{S^i_{n_i}} \leftarrow w^{S^{i_0}_1}, ..., w^{S^{i_0}_{n_{i_0}}}] \\
(\mathcal{T}) & \langle (\mu[a^{T^i_1}, ..., a^{T^i_{n_i}}].c_i)_i | \pi_{i_0}^{\mathcal{T}}[w^{T^{i_0}_1}, ..., w^{T^{i_0}_{n_{i_0}}}] \rangle & \overset{h}{\to}_n & c_{i_0}[a^{T^i_1}, ..., a^{T^i_{n_i}} \leftarrow w^{T^{i_0}_1}, ..., w^{T^{i_0}_{n_{i_0}}}]
\end{array}$$

Figure 5. Syntax and reduction semantics of system L_n

Proposition 3 (Some observational equalities) *The following equations belong to the observational closure of $\to_n$:*

$$\begin{array}{llcl}
(\eta_\mu) & x & = & \mu\alpha.\langle x | \alpha \rangle \\
(\eta_{\tilde{\mu}}) & \alpha & = & \tilde{\mu}x.\langle x | \alpha \rangle \\
(\eta_{\mathcal{S}}) & \alpha & = & [\tilde{\mu}(a^{S^i_1}, ..., a^{S^i_{n_i}}).\langle \iota_i(a^{S^i_1}, ..., a^{S^i_{n_i}}) | \alpha \rangle]_i \\
(\eta_{\mathcal{T}}) & x & = & (\mu[a^{T^i_1}, ..., a^{T^i_{n_i}}].\langle x | \pi_i[a^{T^i_1}, ..., a^{T^i_{n_i}}] \rangle)
\end{array}$$

Observational equalities do not change the consistency of the underlying reduction system. Hence, as a corollary, we have the consistency of the equational theory generated by the reduction system and the η rules.

3.4 System L and sequent calculus

We give in Figure 3.4 a typing system for system L. For simplicity we do not give the generic introduction rules of a positive connective $\mathcal{S}$ and a negative connective $\mathcal{T}$ but instead consider the case with a single negative connective, namely implication. There are seven kinds of judgments accordingly to the number of syntactic categories in system L:

- $\Gamma \vdash v : A \,|\, \Delta$ for terms
- $\Gamma \,|\, e : A \vdash \Delta$ for evaluation contexts
- $\Gamma \vdash W : A \,||\, \Delta$ for weak terms
- $\Gamma \,||\, K : A \vdash \Delta$ for weak evaluation contexts
- $\Gamma \vdash V : A \,;\, \Delta$ for values (linear terms)

$$Ax_R\ \frac{}{\Gamma, x : A \vdash x : A \,|\, \Delta} \qquad Ax_L\ \frac{}{\Gamma \,|\, \alpha : A \vdash \alpha : A, \Delta}$$

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

$$\acute{}_R\ \frac{\Gamma \vdash v : A \,|\, \Delta}{\Gamma \vdash \acute{v} : A \,\|\, \Delta} \qquad \acute{}_L\ \frac{\Gamma \,|\, e : A \vdash \Delta}{\Gamma \,\|\, \acute{e} : A \vdash \Delta}$$

$$Der_R\ \frac{\Gamma \vdash V : A \,;\, \Delta}{\Gamma \vdash V : A \,\|\, \Delta} \qquad Der_L\ \frac{\Gamma \,;\, E : A \vdash \Delta}{\Gamma \,\|\, E : A \vdash \Delta}$$

$$Cut\ \frac{\Gamma \vdash v : A \,|\, \Delta \quad \Gamma \,|\, e : A \vdash \Delta}{c : (\Gamma \vdash \Delta)}$$

$$\to_R \frac{\Gamma, x : A \vdash v : B \,|\, \Delta}{\Gamma \vdash \lambda x.v : A \to B \,;\, \Delta} \qquad \to_L \frac{\Gamma \vdash W : A \,\|\, \Delta \quad \Gamma \,\|\, F : B \vdash \Delta}{\Gamma \,;\, W \cdot F : A \to B \vdash \Delta}$$

Figure 6. A system of simple types for system L with the implication connective

- $\Gamma \,;\, E : A \vdash \Delta$ for linear evaluation contexts
- $c : (\Gamma \vdash \Delta)$ for commands

Obviously, any cut-free proof of LK can be interpreted as a typed command of system L. To this aim, let's first consider a version of LK with no explicit weakening and contraction rule thanks to the integration of weakening at the level of the axiom rule and of the contraction at the level of the logical introduction rules. In the interpretation of LK, the cut rule is then used to simulate the axiom rule of LK (leading to a normal form such as $\langle x|\alpha\rangle$), and to simulate the contraction rule (leading to normal forms such as $\langle x|E\rangle$ or $\langle V|\alpha\rangle$). As for the introduction rules of the connectives, the rule on one side is simply interpreted by a reversible constructor (followed by a cut with an axiom to perform contraction) and the rules on the other side by irreversible constructors whose arguments have been first focused (with μ or $\tilde{\mu}$) then quoted.

The same interpretation still holds for the type system of system L_n which is shown in Figure 3.4. We can check that reduction preserves typing for both systems L and L_n.

$$Ax_R\ \frac{}{\Gamma, x : A \vdash x : A \,|\, \Delta} \qquad Ax_L\ \frac{}{\Gamma \,\|\, \alpha : A \vdash \alpha : A, \Delta}$$

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

$$Der_R\ \frac{\Gamma \vdash V : A \,;\, \Delta}{\Gamma \vdash V : A \,|\, \Delta} \qquad \acute{}_L\ \frac{\Gamma \,|\, e : A \vdash \Delta}{\Gamma \,\|\, \acute{e} : A \vdash \Delta}$$

$$Cut\ \frac{\Gamma \vdash v : A \,|\, \Delta \quad \Gamma \,|\, e : A \vdash \Delta}{c : (\Gamma \vdash \Delta)}$$

$$\to_R \frac{\Gamma, x : A \vdash v : B \,|\, \Delta}{\Gamma \vdash \lambda x.v : A \to B \,;\, \Delta} \qquad \to_L \frac{\Gamma \vdash v : A \,|\, \Delta \quad \Gamma \,\|\, E : B \vdash \Delta}{\Gamma \,\|\, v \cdot E : A \to B \vdash \Delta}$$

Figure 7. A system of simple types for system L_n with the implication connective

3.5 Simulation in linear logic

The typed system L_n can be embedded into linear logic so that the reduction follows cut-elimination in linear logic. The embedding is not a strict decoration (in the sense of Joinet [10]) because the quote operator introduces a cut. The sequents $\Gamma \vdash \Delta$, $\Gamma \vdash A \,;\, \Delta$, $\Gamma \vdash A \,|\, \Delta$, $\Gamma \,\|\, A \vdash \Delta$ and $\Gamma \,|\, A \vdash \Delta$ are respectively interpreted as $!?!\Gamma \vdash ?!\Delta$, $!?!\Gamma \vdash A, ?!\Delta$, $!?!\Gamma \vdash ?!A, ?!\Delta$, $!?!\Gamma, !A \vdash ?!\Delta$ and $!?!\Gamma, !?!A \vdash ?!\Delta$, and the quote operator is interpreted as a cut with the proof of $!A \vdash !?!A$.

3.6 Strong constructors

Compared to $\bar{\lambda}\mu\tilde{\mu}$-calculus or to Wadler's dual calculus, system L misses connectives whose arguments are evaluated following a strong strategy as in the reduction $\langle \lambda x.v|\mu\alpha.c \cdot e\rangle \to_v \langle \mu\alpha.c|\tilde{\mu}x.\langle v|e\rangle\rangle \to_v c[\alpha \leftarrow \tilde{\mu}x.\langle v|e\rangle]$ of call-by-value $\bar{\lambda}\mu\tilde{\mu}$-calculus or as in the reduction $\mathrm{snd}[x.(S)] \to_n y.((y \bullet \mathrm{snd}[\alpha]).\alpha \bullet x.(S)) \to_n y.(S\,[x \leftarrow (y \bullet \mathrm{snd}[\alpha]).\alpha])$ of call-by-name Wadler's calculus.

However, it is still possible to provide connectives whose arguments are strongly reduced by using syntactic sugar. In the case of implication and additive conjunction, this amounts to define

$$\begin{array}{lcl} v \cdot e & \triangleq & \tilde{\mu}x.\langle v|\tilde{\mu}y.\langle \mu\alpha.\langle x|\acute{x} \cdot \acute{\alpha}\rangle|e\rangle\rangle \\ (v_1, v_2) & \triangleq & mu\alpha.\langle v_1|\tilde{\mu}x_1.\langle v_2|\tilde{\mu}x_2.\langle(\acute{x}_1, \acute{x}_2)|\alpha\rangle\rangle\rangle \end{array}$$

Besides validating η-equality, this approach has the advantage of letting the asymmetry of strong evaluation of connectors, such as the asymmetry of conjunction in call-by-value setting, outside of the language, hence keeping pure the symmetry of the underlying language (we follow in this way comments made Selinger in [15]).

Conclusion

We extended the existing computational interpretations of sequent calculus with laziness operators ("quotes") so as to capture the full set of normal form of sequent calculus without sacrificing η-equalities. At the end, we obtain some form of call-by-name calculus with strict features, and dually, we would have obtain a call-by-value calculus with laziness features. We believe that it is worth to investigate from a more practical side such calculi which integrate both call-by-name and call-by-value features.

Beyond η-equalities, what can be said about the observational completeness of these languages? Do we capture all observational equalities in the same way as Böhm's theorem asserts that it is enough to add η to get the observational completeness of β in λ-calculus for the finite Böhm trees? Certainly not since David and Py showed that $\lambda\mu$-calculus does not satisfy Böhm separation theorem [4], a result which holds for call-by-name $\overline{\lambda}\mu\tilde{\mu}$-calculus too. Thanks to our uniform computational characterization of connectives, we can imagine extending the calculus with heterogeneous reductions, say between the constructors of implication and the constructors of additive conjunction, as done by Støvring to prove the conservativity of λ-calculus when extended with surjective pairing [16].

Acknowledgments

I'm particularly grateful to Guillaume Munch with whom I had many fruitful discussions on the topic. Especially, I was strongly influenced by his own use of a quote operator to computationally interpret the modalities of linear logic. I'm also grateful to Pierre-Louis Curien for his support and feedback on this work.

References

[1] P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of ICFP 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000.

[2] H. B. Curry, R. Feys, and W. Craig. *Combinatory Logic*, volume 1. North-Holland, 1958. §9E.

[3] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *J. Symb. Log.*, 62(3):755–807, 1997.

[4] R. David and W. Py. Lambda-mu-calculus and Böhm's theorem. *J. Symb. Log.*, 66(1):407–413, 2001.

[5] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210,405–431, 1935. English Translation in *The Collected Works of Gerhard Gentzen*, "Investigations into logical deduction", North Holland, 1969, pages 68-131.

[6] J.-Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001.

[7] H. Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *CSL '94, Selected Papers*, volume 933 of *LNCS*, pages 61–75. Springer, 1995.

[8] H. Herbelin. *C'est maintenant qu'on calcule: au cœur de la dualité*. Habilitation thesis, University Paris 11, Dec. 2005.

[9] W. A. Howard. The formulae-as-types notion of constructions. In *to H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980. Unpublished manuscript of 1969.

[10] J.-B. Joinet. *Étude de la normalisation du calcul des séquents classique à travers la logique linéaire*. Ph.D. thesis, University Paris 7, Jan. 1993.

[11] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. *Electr. Notes Theor. Comput. Sci.*, 86(4), 2003.

[12] T. Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symp. Logic in Computer Science*, pages 342–349. IEEE Press, 1991.

[13] M. Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR '92 Proceedings*, pages 190–201. Springer-Verlag, 1992.

[14] J. Raghunandan and A. J. Summers. On the computational representation of classical logical connectives. *Electr. Notes Theor. Comput. Sci.*, 171(3):85–109, 2007.

[15] P. Selinger. Some remarks on control categories. Available on the author web page, 2003.

[16] K. Støvring. Extending the extensional lambda calculus with surjective pairing is conservative. *Logical Methods in Computer Science*, 2(2), 2006.

[17] C. Urban. *Classical Logic and Computation*. Ph.D. thesis, University of Cambridge, Oct. 2000.

[18] S. van Bakel, S. Lengrand, and P. Lescanne. The language chi: Circuits, computations and classical logic. In M. Coppo, E. Lodi, and G. M. Pinna, editors, *ICTCS 2005, Proceedings*, volume 3701 of *LNCS*, pages 81–96. Springer, 2005.

[19] P. Wadler. Call-by-value is dual to call-by-name. In C. Runciman and O. Shivers, editors, *Proceedings of ICFP 2003*, volume 38(9) of *SIGPLAN Notices*, pages 189–201. ACM, 2003.

[20] P. Wadler. Call-by-value is dual to call-by-name - reloaded. In J. Giesl, editor, *RTA 2005, Proceedings*, volume 3467 of *LNCS*, pages 185–203. Springer, 2005.