Towards a canonical classical natural deduction system

José Espírito Santo

Centro de Matemática
Universidade do Minho
Portugal
jes@math.uminho.pt

Abstract. This paper studies a new classical natural deduction system, presented as a typed calculus named $\underline{\lambda}\mu\mathsf{let}$. It is designed to be isomorphic to Curien-Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus, both at the level of proofs and reduction, and the isomorphism is based on the correct correspondence between cut (resp. left-introduction) in sequent calculus, and substitution (resp. elimination) in natural deduction. It is a combination of Parigot's $\lambda\mu$-calculus with the idea of "coercion calculus" due to Cervesato-Pfenning, accommodating let-expressions in a surprising way: they expand Parigot's syntactic class of named terms.
This calculus aims to be the simultaneous answer to three problems. The first problem is the lack of a canonical natural deduction system for classical logic. $\underline{\lambda}\mu\mathsf{let}$ is not yet another classical calculus, but rather a canonical reflection in natural deduction of the impeccable treatment of classical logic by sequent calculus. The second problem is the lack of a formalization of the usual semantics of $\overline{\lambda}\mu\tilde{\mu}$-calculus, that explains co-terms and cuts as, respectively, contexts and hole-filling instructions. The mentioned isomorphism is the required formalization, based on the precise notions of context and hole-expression offered by $\underline{\lambda}\mu\mathsf{let}$. The third problem is the lack of a robust process of "read-back" into natural deduction syntax of calculi in the sequent calculus format, that affects mainly the recent proof-theoretic efforts of derivation of λ-calculi for call-by-value. An isomorphic counterpart to the Q-subsystem of $\overline{\lambda}\mu\tilde{\mu}$-calculus is derived, obtaining a new λ-calculus for call-by-value, combining control and let-expressions.

1 Introduction

In the early days of proof theory, Gentzen [9] refined the de Morgan duality between conjuntion and disjunction by defining the sequent calculus LK, a symmetric proof system for classical logic exhibiting a duality, at the level of proofs, between hypothesis and conclusion. Recently Curien and Herbelin [3] extended the Curry-Howard correspondence to LK and showed, by means of the $\overline{\lambda}\mu\tilde{\mu}$-calculus, that classical logic also contains a duality, at the level of cut elimination, between call-by-name (CBN) and call-by-value (CBV) computation.

$\overline{\lambda}\mu\tilde{\mu}$ is remarkably elegant and simple, but not self-sufficient. For several reasons, it would be desirable to have a complementary systems for natural deduction. First, because the computational intuitions about $\overline{\lambda}\mu\tilde{\mu}$ are expressed in terms of a never-formalized natural deduction notation. Co-terms and cuts (="commands") of $\overline{\lambda}\mu\tilde{\mu}$ are interpreted, respectively, as "contexts" and (the instruction of? the result of?) "hole filling", where "contexts" are expressions with a hole from a never-specified language. Second, because $\overline{\lambda}\mu\tilde{\mu}$ dispenses with functional application, but it is rather natural and useful to "read back" to a notation where such construction is available [3, 12].

However, classical natural deduction suffers from several problems of *design* and *dimension*. First, classical natural deduction is often defined as an intuitionistic natural deduction system supplemented with some classical inference principle. Prawitz [20] admits that *"this is perhaps not the most natural procedure from the classical logic point of view"*, as it does not reflect the de Morgan symmetry at the level of proofs; and already Gentzen observed that there is no canonical choice as to what inference principle to add. Computationally, and speaking now about the implicational fragment, this means that the λ-calculus may be extended with a variety of control operators: for instance $\mathcal{C}$, $\varDelta$, or $\mathtt{call-cc}$, corresponding to the principles double-negation elimination, *reductio ad absurdum*, and Peirce's law, respectively [8, 10, 21].

Second, the problem of dimension is that, in retrospect, the standard natural deduction systems are CBN, and all attempts to define classical CBV systems in natural deduction style [16, 3, 22, 11, 12] do not show an explanation of the proof-theoretical issues involved, and a rationale for the hidden CBV side of natural deduction. For instance, let-expressions are unavoidable in CBV λ-calculi, but no proof-theoretical understanding of them is offered. Naively, it is often thought that let-expressions form terms and are typed with the cut rule (of sequent calculus).

In this paper we introduce a new natural deduction system $\underline{\lambda}\mu\mathsf{let}$ for classical logic, presented as an extension of Parigot's $\lambda\mu$-calculus [17], and equipped with let-expressions. Like $\lambda\mu$, $\underline{\lambda}\mu\mathsf{let}$ does not depart from some intuitionistic system, but instead manipulates multiple conclusions. But the main design principle, of course not shared by $\lambda\mu$, is to obtain a system *isomorphic* to $\overline{\lambda}\mu\tilde{\mu}$, in order to have, in the natural deduction side, a system as faithful as LK to the dualities of classical logic. So, $\underline{\lambda}\mu\mathsf{let}$ comes with a sound bijection $\varTheta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$ at the level of the sets of proofs, that is also an isomorphism at the level of normalisation relations; and this isomorphism ensures that $\underline{\lambda}\mu\mathsf{let}$ is not yet another calculus, but rather a canonical classical natural deduction system.

As a proof system, $\underline{\lambda}\mu\mathsf{let}$ has an inference rule named *primitive substitution*, which is the typing rule for let-expressions. Primitive substitution is different from cut because its left premiss can be the conclusion of an elimination. When this is not the case, the primitive substitution is a mere *explicit substitution*[1][1].

[1] At the level of expressions, the corresponding particular case of let-expressions is called explicit substitution too. So we have the Curry-Howard pair (primitive substitution/let-expression), with particular case (explicit substitution/explicit sub-

This is the proof-theoretical understanding of let-expressions put forward by this paper.

$\underline{\lambda}\mu\mathsf{let}$ is a "coercion calculus" [2,6,7], with its syntax carefully organized into syntactic classes. It has a class of *statements*, extending Parigot's named terms. Surprisingly, this is where let-expressions live. $\underline{\lambda}\mu\mathsf{let}$ also has a class of *hole expression*, where applications live, suitable to be filled in the hole of *contexts*. These are a derived syntactic class, as usual in natural deduction, and consist of statements with a hole in the left end.

Such ingredients allow us to give a formalization, via Θ, of the usual semantics of $\overline{\lambda}\mu\tilde{\mu}$. We derive yet another syntactic notion in $\underline{\lambda}\mu\mathsf{let}$, that of *contextual*. Contextuals stand to contexts as numerals 0, $s(0)$, $s(s(0))$, etc. stand to numbers, and are therefore *instructions* for building contexts; and they are manipulated formally in $\overline{\lambda}\mu\tilde{\mu}$ as co-terms. Similarly, "commands" of $\overline{\lambda}\mu\tilde{\mu}$ are *instructions* of hole-filling, but the *result* of those instructions are the statements of $\underline{\lambda}\mu\mathsf{let}$. The isomorphism $\Theta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$ boils down to the execution of such instructions.

Θ is also the tool for doing "read-back" into natural deduction systematically. We use it to reflect in natural deduction the simple and elegant treatment of CBN and CBV computation offered by $\overline{\lambda}\mu\tilde{\mu}$ [3]. That is, we find in $\underline{\lambda}\mu\mathsf{let}$ the right definitions of CBN and CBV reductions or subsystems when the appropriate restrictions of Θ from the known counterparts in $\overline{\lambda}\mu\tilde{\mu}$ can be established. In particular, we find new CBV λ-calculi in natural deduction syntax by reflecting, through Θ, the known CBV fragments of $\overline{\lambda}\mu\tilde{\mu}$. Surprisingly, the resulting calculi have escaped the recent efforts in the literature [3,22,11,12] for obtaining through similar proof-theoretical means λ-calculi for CBV combining control operators and let-expressions in a logically founded way.

Structure of the paper. The paper is organized as follows. Section 2 recalls $\overline{\lambda}\mu\tilde{\mu}$. Section 3 presents $\underline{\lambda}\mu\mathsf{let}$. Section 4 proves the isomorphism $\overline{\lambda}\mu\tilde{\mu} \cong \underline{\lambda}\mu\mathsf{let}$ and explains the semantics of $\overline{\lambda}\mu\tilde{\mu}$ and the proof-theoretical foundation of let-expressions, explicit substitutions, and named expressions. Section 5 investigates CBN and CBV in $\underline{\lambda}\mu\mathsf{let}$, as well as "read-back" into natural deduction. Section 6 summarizes the paper, reviews the literature, and suggests future work.

2 Background

In this section we fix notation and recall Curien-Herbelin's $\overline{\lambda}\mu\tilde{\mu}$ [3].

Notations. In λ-calculi for classical logic, like $\lambda\mu$ or $\overline{\lambda}\mu\tilde{\mu}$, there are variables and co-variables. Variables (resp. co-variables) are always ranged by x, y, z (resp. a, b, c). Meta-substitution is denoted with square brackets $[\_/x]\_$. Similarly for other forms of meta-substitution used in λ-calculi for classical logic, like "structural" substitution. Explicit substitution is denoted with angle brackets $\langle\_/x\rangle\_$. All calculi in this paper assume Barendregt's variable convention (in particular we take renaming of bound variables or co-variables and avoidance of capture for granted).

stitution). To avoid confusion, we refer to any meta-*operation* of substitution as "meta-substitution".

Fig. 1. Typing rules for $\overline{\lambda}\mu\tilde{\mu}$

$$\frac{}{\Gamma|a : A \vdash a : A, \Delta}\;LAx \qquad \frac{}{\Gamma, x : A \vdash x : A|\Delta}\;RAx$$

$$\frac{\Gamma \vdash u : A|\Delta \quad \Gamma|e : B \vdash \Delta}{\Gamma|u :: e : A \supset B \vdash \Delta}\;LIntro \qquad \frac{\Gamma, x : A \vdash t : B|\Delta}{\Gamma \vdash \lambda x.t : A \supset B|\Delta}\;RIntro$$

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

$$\frac{\Gamma \vdash t : A|\Delta \quad \Gamma|e : A \vdash \Delta}{\langle t|e\rangle : (\Gamma \vdash \Delta)}\;Cut$$

A *value* is a variable or λ-abstraction, and usually is denoted V or W. If $\boldsymbol{N} = N_1, \cdots, N_m$ $(m \geq 0)$, then $M\boldsymbol{N}$ denotes $MN_1 \cdots N_m$, that is, $(\cdots(MN_1)\cdots N_m))$, and $\boldsymbol{N} :: e$ denotes $N_1 :: \cdots :: N_m :: e$, that is, $(N_1 :: \cdots :: (N_m :: e)\cdots))$.

Types (=formulas) are ranged over by A, B, C and generated from type variables using the "arrow" (=implication), written $A \supset B$.

$\overline{\lambda}\mu\tilde{\mu}$-calculus. Expressions are either terms, co-terms or commands, and are defined by the following grammar:

$$t, u ::= x \,|\, \lambda x.t \,|\, \mu a.c \qquad e ::= a \,|\, u :: e \,|\, \tilde{\mu}x.c \qquad c ::= \langle t|e\rangle$$

There is one kind of sequent per each syntactic class $\Gamma \vdash t : A|\Delta$, $\Gamma|e : A \vdash \Delta$, and $c : (\Gamma \vdash \Delta)$. In the first two kinds, the displayed formula A is *selected*. Typing rules are given in Figure 1. A typable term is a term t such that $\Gamma \vdash t : A|\Delta$ is derivable, for some Γ, Δ, A. Similarly for co-terms e and commands c.

In addition to three ordinary substitution operators, there are three *co-substitution* operators $[e/a]c$, $[e/a]u$, and $[e/a]e'$. We use the abbreviations

$$\langle t/x\rangle c := \langle t|\tilde{\mu}x.c\rangle \qquad \langle e/a\rangle c := \langle \mu a.c|e\rangle \tag{1}$$

called *explicit substitution* and *explicit co-substitution*, respectively.[2]

We consider 5 reduction rules:

$$\begin{array}{llll}
(\beta) & \langle \lambda x.t|u :: e\rangle \to \langle u/x\rangle\langle t|e\rangle & & \\
(\sigma) & \langle t/x\rangle c \to [t/x]c & (\eta_{\tilde{\mu}}) & \tilde{\mu}x.\langle x|e\rangle \to e, \text{ if } x \notin e \\
(\pi) & \langle e/a\rangle c \to [e/a]c & (\eta_{\mu}) & \mu a.\langle t|a\rangle \to t, \text{ if } a \notin t
\end{array}$$

The reduction rules usually named $\tilde{\mu}$ and μ are here renamed σ and π, respectively, and written with the (co)substitution abbreviations. The rules η_μ and $\eta_{\tilde{\mu}}$ are considered *e.g.* in [19][3]. There is a critical pair, called the *CBN-CBV dilemma*:

[2] It is useful to recall that Parigot's named terms are derived in $\overline{\lambda}\mu\tilde{\mu}$ as $a(t) := \langle t|a\rangle$.

[3] Curiously, if we omit the η-like rules, we do not see any of μ or $\tilde{\mu}$ in the reduction rules, when the (co-)substitution abbreviations (1) are used. The notation (1) emphasizes

$$\begin{tikzcd}[\mu a.c/x]c' & \langle \mu a.c|\tilde{\mu}x.c'\rangle \arrow[l, "\sigma"'] \arrow[r, "\pi"'] & [\tilde{\mu}x.c'/a]c\end{tikzcd}$$

According to [3], avoiding this dilemma is the principle for the definition of the CBN and CBV fragments of $\overline{\lambda}\mu\tilde{\mu}$. See section 5 for more on this.

3 The natural deduction system $\underline{\lambda}\mu\mathsf{let}$

As we present $\underline{\lambda}\mu\mathsf{let}$, we compare informally with Parigot's $\lambda\mu$ [17].

Primitive syntax. Expressions of $\underline{\lambda}\mu\mathsf{let}$ are defined by the following grammar:

$$\begin{array}{lrl} \text{(Terms)} & M, N, P ::= & x \mid \lambda x.M \mid \mu a.S \\ \text{(Hole Expressions)} & H ::= & \mathsf{h}(M) \mid HN \\ \text{(Statements)} & S ::= & a(H) \mid \mathsf{let}\, x = H \,\mathsf{in}\, S \end{array}$$

Terms are either variables, λ-abstractions $\lambda x.M$, or μ-abstractions $\mu a.S$ whose body is a statement S. *Statements* are either *named expressions* of the form $a(H)$, or *let-expressions* $\mathsf{let}\, x = H \,\mathsf{in}\, S$. Hole expressions H are either *coercions* $\mathsf{h}(M)$, or applications HN[4]. Informally, every statement has one of two forms $a((\mathsf{h}(M)N_1 \cdots N_m))$ or $\mathsf{let}\, x = (\mathsf{h}(M)N_1 \cdots N_m) \,\mathsf{in}\, S$, with $m \geq 0$. So, not only $\mathsf{h}(M)$ means M coerced to a hole expression, but it also signals the *head* term of a statement. In $\lambda\mu$ there are neither hole expressions, nor let-expressions. Applications are terms and statements are just named terms $a(M)$.

Typing system The typing system of $\underline{\lambda}\mu\mathsf{let}$, given in Fig. 2, derives three kinds of sequents, one for each syntactic class:

$$\Gamma \vdash M : A|\Delta \qquad \Gamma \rhd H : A|\Delta \qquad S : (\Gamma \vdash \Delta) \ .$$

The first and third kinds (*term sequents* and *statement sequents*, resp.) are familiar from $\lambda\mu$. If we disregard the distinction between the first two kinds of sequents, then the first five typing rules in Fig. 2 are exactly those of $\lambda\mu$, and the coercion rule is a trivial repetition rule. So, up to the substitution rule, we have a refinement of the typing system of $\lambda\mu$, that is, a classical natural deduction system where sequents have to be chosen of the appropriate kind, and containing an extra-rule for coercing between two different kinds of sequents.

The final rule, called primitive substitution, or just substitution, is also standard, apart from the fact that sequents have to be chosen of the appropriate kind. The connection and the difference relatively to sequent calculus' cut rule will become clear after the proof of $\overline{\lambda}\mu\tilde{\mu} \cong \underline{\lambda}\mu\mathsf{let}$.

that β, σ, and π are about generation and execution of explicit (co-)substitution. The execution itself, by σ or π, is in one go, by calling meta-operations.

[4] Hole expressions are indeed expressions that go into the hole of contexts: see below. Ordinary application between two terms is derivable: $MN := \mu a.a(\mathsf{h}(M)N)$; so are Parigot's named terms: $a(M) := a(\mathsf{h}(M))$

Fig. 2. Typing rules for $\underline{\lambda}\mu\mathsf{let}$

$$\frac{}{\Gamma, x : A \vdash x : A|\Delta}\ \mathit{Assumption} \qquad \frac{\Gamma \vdash M : A|\Delta}{\Gamma \rhd \mathsf{h}(M) : A|\Delta}\ \mathit{Coercion}$$

$$\frac{\Gamma \rhd H : A \supset B|\Delta \quad \Gamma \vdash N : A|\Delta}{\Gamma \rhd HN : B|\Delta}\ \mathit{Elim} \qquad \frac{\Gamma, x : A \vdash M : B|\Delta}{\Gamma \vdash \lambda x.M : A \supset B|\Delta}\ \mathit{Intro}$$

$$\frac{\Gamma \rhd H : A|\Delta, a : A}{a(H) : (\Gamma \vdash \Delta, a : A)}\ \mathit{Pass} \qquad \frac{S : (\Gamma \vdash \Delta, a : A)}{\Gamma \vdash \mu a.S : A|\Delta}\ \mathit{Act}$$

$$\frac{\Gamma \rhd H : A|\Delta \quad S : (\Gamma, x : A \vdash \Delta)}{\mathsf{let}\, x = H\, \mathsf{in}\, S : (\Gamma \vdash \Delta)}\ \mathit{Subst}$$

A *typable* term is a term M such that $\Gamma \vdash M : A|\Delta$ is derivable, for some Γ, Δ, A. Similarly for hole expressions H and statements S.

Derived syntax. *Explicit substitution* in $\underline{\lambda}\mu\mathsf{let}$ is the following abbreviation:

$$\langle N/x \rangle S := \mathsf{let}\, x = \mathsf{h}(N)\, \mathsf{in}\, S$$

Another *derived* syntactical concept of $\underline{\lambda}\mu\mathsf{let}$, crucial for the definition of reduction rules and for the comparison with $\overline{\lambda}\mu\tilde{\mu}$, is that of *context*[5]. A context is an expression of the two possible forms ($m \geq 0$):

$$a(([\,]N_1 \cdots N_m)) \qquad \mathsf{let}\, x = ([\,]N_1 \cdots N_m)\, \mathsf{in}\, S \tag{2}$$

So, a context is a statement with a "hole" $[\,]$ in a position where a hole expression H is expected. Let $\mathcal{E}[\,]$ range over contexts, and $\mathcal{E}[H]$ denote the statement obtained by filling the hole of $\mathcal{E}[\,]$ with H. Such expressions are generated by the following algebra $\mathbb{E}$: two constants $a([\,])$ and $\mathsf{let}\, x = [\,]\, \mathsf{in}\, S$ and an operation that sends $\mathcal{E}[\,]$ to $\mathcal{E}[[\,]N]$. We introduce the signature of this algebra:

$$\begin{array}{rl} a([\,]) & a \\ \mathsf{let}\, x = [\,]\, \mathsf{in}\, S & \tilde{\mu}x.S \\ \mathcal{E}[] \mapsto \mathcal{E}[[\,]N] & N :: \_ \end{array}$$

The closed terms of the free algebra with the same signature are called *contextuals* and defined by the following grammar[6]:

[5] Rocheteau [22] *extends* $\lambda\mu$ with certain contexts. We follow the style of the natural deduction systems of [7], where contexts are not primitive.

[6] The notation is of course chosen to match that of $\overline{\lambda}\mu\tilde{\mu}$, but while in $\overline{\lambda}\mu\tilde{\mu}$ "contexts" (that is, co-terms) belong to the primitive syntax, context(ual)s in $\underline{\lambda}\mu\mathsf{let}$ are a derived concept. The exact connection between contexts in $\overline{\lambda}\mu\tilde{\mu}$ and $\underline{\lambda}\mu\mathsf{let}$, which justifies the choice of notation, will be established later, after the isomorphism between the two systems is proved.

$$\text{(Contextuals)} \quad \mathcal{E} ::= a \mid \tilde{\mu}x.S \mid N :: \mathcal{E}$$

In the same way as the numeral $s(s(0))$ denotes the number 2, the contextual $N :: N' :: a$ (resp. $N :: \tilde{\mu}x.S$) denotes the context $a(([\,]NN'))$ (resp. $\mathsf{let}\, x = [\,]N \,\mathsf{in}\, S$). Since there is an intended interpretation of the signature (the algebra $\mathbb{E}$), each $\mathcal{E}$ is associated with a unique context. Since $\mathbb{E}$ generates the set of contexts, any context is denoted by some $\mathcal{E}$. So, we can *identify* contextuals with contexts (as we can identify numbers with numerals once the usual interpretation of 0 and s is fixed). For instance, hole filling $\mathcal{E}[H]$ can now be defined by recursion:

$$\begin{aligned} a[H] &= a(H) \\ (\tilde{\mu}x.S)[H] &= \mathsf{let}\, x = H \,\mathsf{in}\, S \\ (N :: \mathcal{E})[H] &= \mathcal{E}[HN] \end{aligned} \tag{3}$$

In addition to meta-substitution for variables, there are three operations of meta-substitution for co-variables $[\mathcal{E}/a]M$, $[\mathcal{E}/a]H$, and $[\mathcal{E}/a]S$, defined by a simultaneous recursion, all of whose clauses are homomorphic, but the crucial one:

$$[\mathcal{E}/a](a(H)) = \mathcal{E}[H'] \qquad \text{where } H' = [\mathcal{E}/a]H.$$

For instance: (i) $[N :: \mathcal{E}/a](a(\mathsf{h}(M))) = \mathcal{E}[\mathsf{h}(M')N]$, with $M' = [N :: \mathcal{E}/a]M$; so, $[N :: \mathcal{E}/a]\_$ is a form of "structural substitution" as found in $\lambda\mu$. (ii) $[b/a](a(H)) = b([b/a]H)$; $[b/a]\_$ is a renaming operation, also found in $\lambda\mu$. [7]

Reduction rules. Some of the reduction rules of $\underline{\lambda}\mu\mathsf{let}$ will act on the head of statements. We use contexts as a device for bringing to surface such heads, which normally are buried under a sequence of arguments. For instance, if S is $\mathsf{let}\, x = \mathsf{h}(M)N_1 \cdots N_m \,\mathsf{in}\, S'$, then $S = \mathcal{E}[\mathsf{h}(M)]$, where $\mathcal{E} = N_1 :: \cdots N_m :: \tilde{\mu}x.S'$.

The reduction rules of $\underline{\lambda}\mu\mathsf{let}$ are as follows:

$$\begin{array}{llll} (\beta) & \mathcal{E}[\mathsf{h}(\lambda x.M)N] \to \langle N/x\rangle\mathcal{E}[\mathsf{h}(M)] & & \\ (\sigma) & \langle N/x\rangle S \to [N/x]S & (\eta_\mu) & \mu a.a(\mathsf{h}(M)) \to M,\ a \notin M \\ (\pi) & \mathcal{E}[\mathsf{h}(\mu a.S)] \to [\mathcal{E}/a]S & (\eta_{\mathsf{let}}) & \mathsf{let}\, x = H \,\mathsf{in}\, \mathcal{E}[\mathsf{h}(x)] \to \mathcal{E}[H],\ x \notin \mathcal{E} \end{array}$$

By *normalisation* we mean $\beta\pi\sigma$-reduction.

Rule β generates a substitution, which is executed *implicitly* by a separate rule (σ). So, it would be perhaps more appropriate to call $\langle N/x\rangle S$ a "delayed" substitution. Rule π plays in $\underline{\lambda}\mu\mathsf{let}$ a role similar to the role played by rules μ and ρ in $\lambda\mu$, being the union of two rules:

[7] Informally, the connection with "structural substitution" is as follows ($\boldsymbol{N}$ of length $m \geq 0$):

$$\begin{aligned} [\boldsymbol{N} :: b/a]\_ &= [b(\bullet\boldsymbol{N})/a\bullet]\_ \\ [\boldsymbol{N} :: \tilde{\mu}x.S/a]\_ &= [\mathsf{let}\, x = \bullet\boldsymbol{N} \,\mathsf{in}\, S/a\bullet]\_ \end{aligned}$$

Fig. 3. CBN/CBV dilemma in $\underline{\lambda}\mu\mathsf{let}$

$$\begin{tikzcd}
{[\mu a.S/x]S'} & \mathsf{let}\,x = \mathsf{h}(\mu a.S)\,\mathsf{in}\,S' \arrow[l, "\sigma"'] \arrow[r, "\pi"'] & {[\tilde{\mu}x.S'/a]S}
\end{tikzcd}$$

$$b((\mathsf{h}(\mu a.s)\boldsymbol{N})) \to [\boldsymbol{N} :: b/a]S \tag{4}$$
$$\mathsf{let}\,x = \mathsf{h}(\mu a.S)\boldsymbol{N}\,\mathsf{in}\,S' \to [\boldsymbol{N} :: \tilde{\mu}x.S'/a]S \ \ , \tag{5}$$

with $\boldsymbol{N}$ of length $m \geq 0$. If $m = 0$ in (4), then we have a version of the "renaming" rule ρ. If $m > 0$ in (5), then a subexpression of the form $\mathsf{h}(\mu a.S)N$ exists, but, in contrast to rule μ of $\lambda\mu$, the whole statement of which $\mathsf{h}(\mu a.S)$ is the head is transformed in a single π-step. Rule η_μ is similar to the rule with same name in $\lambda\mu$. Rule η_{let} has no counterpart in $\lambda\mu$ because the latter has no let-expressions.

Properties. Strong normalisation of typable expressions of $\underline{\lambda}\mu\mathsf{let}$ will be a consequence of isomorphism with $\overline{\lambda}\mu\tilde{\mu}$, to be proved in the next section.

If we take $\boldsymbol{N}$ of length 0 in (5), the redex is also a σ-redex. Hence, like $\overline{\lambda}\mu\tilde{\mu}$, $\underline{\lambda}\mu\mathsf{let}$ has a critical pair between π and σ that breaks confluence: see Fig. 3. Later (see Section 5) we will discuss fragments of $\underline{\lambda}\mu\mathsf{let}$ that are isomorphic to confluent fragments of $\overline{\lambda}\mu\tilde{\mu}$, and therefore confluent themselves.

The $\beta\pi\sigma$-normal forms are given by:

$$\begin{array}{rcl}
M_{nf}, N_{nf} & ::= & x \mid \lambda x.M_{nf} \mid \mu a.S_{nf} \\
H_{nf} & ::= & \mathsf{h}(x) \mid H_{nf}N_{nf} \\
S_{nf} & ::= & a(\mathsf{h}(\lambda x.M_{nf})) \\
& \mid & a(H_{nf}) \mid \mathsf{let}\,x = H_{nf}N_{nf}\,\mathsf{in}\,S_{nf}
\end{array}$$

At the level of derivations, the *normality criterion* is:

- The left premiss of every substitution is the conclusion of an elimination;
- The premiss of a coercion is never the conclusion of an activation; moreover if a coercion is the main premiss of an elimination, then its premiss is an assumption.

Theorem 1 (Subformula property). *In a derivation of $\Gamma \vdash M_{nf} : A|\Delta$, all formulas are subformulas of Γ, A, or Δ.* [8]

Proof: Similar claims for S_{nf} and H_{nf} are proved by simultaneous induction; but, crucially, the claim for H_{nf} is stronger: the type A is a subformula of Γ. $\square$

Discussion. At first sight, $\underline{\lambda}\mu\mathsf{let}$ is a complex system. For instance, contexts are derived syntax, but the expressions that are filled in the hole of contexts

[8] We say that A is a subformula of Γ (resp. Δ) if A is subformula of some formula in Γ (resp. Δ).

are primitive. However, we seek, not what we would like natural deduction to be, but what natural deduction is, if it is to be isomorphic to $\overline{\lambda}\mu\tilde{\mu}$. Consider another example: reduction rules of $\underline{\lambda}\mu\mathsf{let}$. We might regret their complex formulation, with manipulation of contexts $\mathcal{E}$. But isn't $\underline{\lambda}\mu\mathsf{let}$ supposed to be a control calculus? The rule π is particularly revealing: it should express the features of μ-abstraction as a control operator; but, in $\overline{\lambda}\mu\tilde{\mu}$, π is *prima facie* a substitution execution rule; only its isomorphic counterpart in $\underline{\lambda}\mu\mathsf{let}$ reveals the control operation.

$\underline{\lambda}\mu\mathsf{let}$ is a "coercion calculus". Simplifying matters, there is a coercion calculus in [2, 6], whose syntax has the typical separation into several classes

$$M, N, P ::= x \,|\, \lambda x.M \,|\, \{H\} \qquad H ::= \mathsf{h}(M) \,|\, HN \ ,$$

reflecting in natural deduction a fragment of intuitionistic sequent calculus. In addition to $\mathsf{h}(M)$, there is a backward coercion $\{H\}$, which had to be developed in [7] to a substitution construction $\{H/x\}P$, in order to reflect full intuitionistic sequent calculus. $\underline{\lambda}\mu\mathsf{let}$ represents a further elaboration of the same construction, capable of capturing full classical sequent calculus.[9]

Regarding the the proof of the subformula property, in $\underline{\lambda}\mu\mathsf{let}$ inspection of inference rules falls short[10], but nevertheless a proof by induction on normal forms is possible and straightforward. Prawitz [20] proves only a "slightly weakened subformula property" [25], that requires a preliminary analysis of the structure of normal derivations, by which "branches" are shown to have elimination and introduction parts. A similar structure is *built in* every derivation of $\underline{\lambda}\mu\mathsf{let}$, as a consequence of its organization as a coercion calculus.

4 Isomorphism

In this section mappings $\varTheta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$ and $\varPsi : \underline{\lambda}\mu\mathsf{let} \to \overline{\lambda}\mu\tilde{\mu}$ are defined and analised. In particular, they establish $\overline{\lambda}\mu\tilde{\mu} \cong \underline{\lambda}\mu\mathsf{let}$. As a corollary, strong normalisation for $\underline{\lambda}\mu\mathsf{let}$ follows. Next we show why $\varTheta$ is a semantics of $\overline{\lambda}\mu\tilde{\mu}$, and explain the proof-theory of let-expressions and named expressions.

Mappings $\varPsi$ and $\varTheta$. Let $\varPsi(M) = t$, $\varPsi(N_i) = u_i$ and $\varPsi(S) = c$. The idea behind $\varPsi : \underline{\lambda}\mu\mathsf{let} \longrightarrow \overline{\lambda}\mu\tilde{\mu}$ is to map statements as follows:

$$\mathsf{let}\, x = \mathsf{h}(M)N_1 \cdots N_m \,\mathsf{in}\, S \mapsto \langle t | u_1 :: \cdots :: u_m :: \tilde{\mu}x.c\rangle \tag{6}$$
$$a((\mathsf{h}(M)N_1 \cdots N_m)) \mapsto \langle t | u_1 :: \cdots :: u_m :: a\rangle \tag{7}$$

The idea behind $\varTheta : \overline{\lambda}\mu\tilde{\mu} \longrightarrow \underline{\lambda}\mu\mathsf{let}$ is the translation of commands obtained by reverting these mappings, with $\varTheta(t) = M$, $\varTheta(u_i) = N_i$ and $\varTheta(c) = S$. See Fig. 4. Observe that, in (6) and (7), each occurrence of application H_iN_i is replaced by an occurrence of left introduction $u_i :: e_i$. Conversely for $\varTheta$. Soundness of $\varTheta$ and $\varPsi$ is routine. As a consequence, $\varTheta$ and $\varPsi$ preserve typability.

[9] Observe the progression: $\{H\} := \{H/x\}x$ and $\{H/x\}P := \mu a.\mathsf{let}\, x = H \,\mathsf{in}\, a(\mathsf{h}(P))$.

[10] Inspection of inference rules is also insufficient for establishing the subformula property for $\overline{\lambda}\mu\tilde{\mu}$, because some instances of *Cut* are not eliminable.

Fig. 4. Mappings $\Psi : \underline{\lambda}\mu\mathsf{let} \to \overline{\lambda}\mu\tilde{\mu}$ and $\Theta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$

$\Psi(x) = x$	$\Theta(x) = x$
$\Psi(\lambda x.M) = \lambda x.\Psi M$	$\Theta(\lambda x.t) = \lambda x.\Theta t$
$\Psi(\mu a.S) = \mu a.\Psi S$	$\Theta(\mu a.c) = \mu a.\Theta c$
$\Psi(a(H)) = \Psi(H,a)$	$\Theta\langle t\|e\rangle = \Theta(\mathsf{h}(\Theta t), e)$
$\Psi(\mathsf{let}\, x = H \,\mathsf{in}\, S) = \Psi(H, \tilde{\mu}x.\Psi S)$	$\Theta(H,a) = a(H)$
$\Psi(\mathsf{h}(M), e) = \langle \Psi M\|e\rangle$	$\Theta(H, \tilde{\mu}x.c) = \mathsf{let}\, x = H \,\mathsf{in}\, \Theta c$
$\Psi(HN, e) = \Psi(H, \Psi N :: e)$	$\Theta(H, u :: e) = \Theta(H\Theta u, e)$

Theorem 2 (Isomorphism). *Mappings Ψ and Θ are mutually inverse bijections between the set of $\overline{\lambda}\mu\tilde{\mu}$-terms and the set of $\underline{\lambda}\mu\mathsf{let}$-terms. Moreover, for $R = \beta$ (resp. $R = \sigma, \pi, \eta_\mu, \eta_{\tilde{\mu}}$), and $R' = \beta$ (resp. $R' = \sigma, \pi, \eta_\mu, \eta_{\mathsf{let}}$):*

1. *$t \to_R t'$ in $\overline{\lambda}\mu\tilde{\mu}$ iff $\Theta t \to_{R'} \Theta t'$ in $\underline{\lambda}\mu\mathsf{let}$.*
2. *$M \to_{R'} M'$ in $\underline{\lambda}\mu\mathsf{let}$ iff $\Psi M \to_R \Psi M'$ in $\overline{\lambda}\mu\tilde{\mu}$.*

Corollary 1 (SN). *Every typable expression of $\underline{\lambda}\mu\mathsf{let}$ is $\beta\sigma\pi\eta_\mu\eta_{\mathsf{let}}$-SN.*

Proof: SN holds of $\overline{\lambda}\mu\tilde{\mu}$ [19], Θ and Ψ preserve typability, and $\overline{\lambda}\mu\tilde{\mu} \cong \underline{\lambda}\mu\mathsf{let}$. $\square$

Semantics of $\overline{\lambda}\mu\tilde{\mu}$. The choice of notation for contexts in $\underline{\lambda}\mu\mathsf{let}$ imposes the following trivial extensions of Ψ and Θ to contexts and co-terms:

$$\begin{array}{cc} \Psi a = a & \Theta a = a \\ \Psi(\tilde{\mu}x.S) = \tilde{\mu}x.\Psi S & \Theta(\tilde{\mu}x.c) = \tilde{\mu}x.\Theta c \\ \Psi(N :: \mathcal{E}) = \Psi N :: \Psi\mathcal{E} & \Theta(u :: e) = \Theta u :: \Theta e \end{array} \tag{8}$$

We can identify each context $\mathcal{E}$ of $\underline{\lambda}\mu\mathsf{let}$ with a function of type $\underline{\lambda}\mu\mathsf{let} - HoleExpressions \to \underline{\lambda}\mu\mathsf{let} - Statements$; it is the function that sends H to $\mathcal{E}[H]$ (hence $\mathcal{E}(H) = \mathcal{E}[H]$). Now let e be a co-term of $\overline{\lambda}\mu\tilde{\mu}$ and consider $\Theta(\_, e) : \underline{\lambda}\mu\mathsf{let} - HoleExpressions \to \underline{\lambda}\mu\mathsf{let} - Statements$. By induction on e one proves (simply by unfolding the definitions of $\Theta(H, e)$ in Fig. 4 and $\mathcal{E}[H]$ in (3)) that

$$\Theta(e)[H] = \Theta(H, e) \ . \tag{9}$$

Hence, $\Theta(\_, e)$ and Θe are the same function.

So, $\Theta(\_, a)$ and the $\underline{\lambda}\mu\mathsf{let}$-context denoted a are the same function; if $S = \Theta c$, then $\Theta(\_, \tilde{\mu}x.c)$ and the $\underline{\lambda}\mu\mathsf{let}$-context denoted $\tilde{\mu}x.S$ are the same function; finally, if $\Theta u = N$ and $\Theta e = \mathcal{E}$, then $\Theta(\_, u :: e)$ and the $\underline{\lambda}\mu\mathsf{let}$-context denoted $N :: \mathcal{E}$ are the same function. This justifies the choice of notation for contexts in $\underline{\lambda}\mu\mathsf{let}$.

The formalization of the intuitive *semantics* of $\overline{\lambda}\mu\tilde{\mu}$, offered by $\underline{\lambda}\mu\mathsf{let}$, is the mapping $\theta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$ defined homomorphically on terms, like Θ; that sends co-terms to contexts homomorphically, as in (8); and defined by $\theta\langle t|e\rangle = \theta(e)[\mathsf{h}(\theta t)]$ on commands. The whole action of θ is concentrated on the translation of $\langle t|e\rangle$, that reads "fill $\mathsf{h}(\theta t)$ in the hole of the context θe".

Corollary 2 (Semantics of $\overline{\lambda}\mu\tilde{\mu}$). *$\theta t = \Theta t$ and $\theta e = \Theta e$ and $\theta c = \Theta c$.*

Proof: Trivial induction on t, e, and c. The case $c = \langle t|e\rangle$ follows from (9). $\square$

So, the proof-theoretical mapping Θ, that replaces left-introductions by eliminations, is the semantics θ; and the latter is an isomorphism by Theorem 2.

Proof-theory of let-expressions and named expressions. Recall the typing systems of $\overline{\lambda}\mu\tilde{\mu}$ and $\underline{\lambda}\mu\mathsf{let}$ (Figs. 1 and 2). We explain the difference between cut in $\overline{\lambda}\mu\tilde{\mu}$ and primitive substitution in $\underline{\lambda}\mu\mathsf{let}$, thereby clarifying the proof-theoretical status of let-expressions. We also argue that explicit substitutions and Parigot's named terms are, in some sense, neutral w.r.t. the sequent calculus/natural deduction difference. [11]

We say, for the sake of this discussion, that a cut $\langle t|e\rangle$ is of type I (resp. II) if it has the shape of the r.h.s. of (6) (resp. (7)). Through Θ, cuts of type I correspond to let-expressions/primitive substitutions $\mathsf{let}\, x = H \,\mathsf{in}\, S$, whereas cuts of type II correspond to the naming construction $a(H)$. [12]

In contrast to the right premiss e of the cut of type I $\langle t|e\rangle$, the right premiss S of the primitive substitution $\mathsf{let}\, x = H \,\mathsf{in}\, S$ can never be the conclusion of left-introduction (similar remark already made by Negri and von Plato [15], page 172). But, as a compensation, the left premiss H is not limited to be morally a term $\mathsf{h}(M)$ (as is the left premiss t in $\langle t|e\rangle$), it can be a sequence of eliminations (as already remarked in [7] for the intuitionistic case). So, cuts of type I (in sequent calculus) are more general on the right premiss, while substitutions (in natural deduction) are more general on the left premiss. But there is a kind of common case, when $e = \tilde{\mu}x.c$ (so e is not a left introduction), and $H = \mathsf{h}(M)$ (so H is not an elimination). Then the cut of type I has the form $\langle t/x\rangle c = \langle t|\tilde{\mu}x.c\rangle$ and the primitive substitution has the form $\langle N/x\rangle S = \mathsf{let}\, x = N \,\mathsf{in}\, S$, the form of an explicit substitution.

Similarly, Parigot's named terms can be seen as the common case of cuts of type II and the construction $a(H)$: such case reads $a(t) = \langle t|a\rangle$ ($e = a$ is not a left introduction) and $a(M) = a(\mathsf{h}(M))$ ($H = \mathsf{h}(M)$ is not an elimination).

5 Call-by-name and call-by-value

In this section we analyse CBN and CBV in $\underline{\lambda}\mu\mathsf{let}$. We do "read-back" [3, 12] in a systematic fashion, reflecting into $\underline{\lambda}\mu\mathsf{let}$, through Θ, the definitions of CBN and CBV reductions and fragments of $\overline{\lambda}\mu\tilde{\mu}$. Finally, we analyse the largest CBV fragment of $\underline{\lambda}\mu\mathsf{let}$ obtained.

CBN and CBV reduction. In $\overline{\lambda}\mu\tilde{\mu}$, CBN and CBV reduction is defined by giving priority to either σ or π, respectively, in the CBN/CBV dilemma. Recall the CBN-CBV dilemma of $\underline{\lambda}\mu\mathsf{let}$ (Fig. 3). In $\underline{\lambda}\mu\mathsf{let}$ we define:

CBV reduction: σ is restricted to the case $\mathsf{let}\, x = \mathsf{h}(V) \,\mathsf{in}\, S' \to [V/x]S'$.

CBN reduction: π is restricted to the case $\mathcal{E}_n[\mathsf{h}(\mu a.S)] \to [\mathcal{E}_n/a]S$, where *CBN contexts* are given by: $\mathcal{E}_n ::= a \,|\, N :: \mathcal{E}_n$.

With the first restriction, the σ-reduction in Fig. 3 becomes forbidden; with the second, it is the π-reduction in Fig. 3 which becomes blocked.

[11] Recall the particular cases $\langle M/x\rangle S \mapsto \langle t/x\rangle c$ and $a(M) \mapsto a(t)$ of (6) and (7), resp.

[12] In intuitionistic logic [7], cuts correspond only to primitive substitution.

Fig. 5. CBN and CBV fragments

$$\begin{tikzcd}
\overline{\lambda}\mu\tilde{\mu} \arrow[r, leftrightarrow, "\Theta, \Psi"] \arrow[d, dash, dashed] & \underline{\lambda}\mu\mathsf{let} \arrow[d, dash, dashed] \\
\overline{\lambda}\mu\tilde{\mu}_{\mathsf{T}} \arrow[r, leftrightarrow, "\Theta, \Psi"] \arrow[d, dash, dashed] & \underline{\lambda}\mu\mathsf{let}_{\mathsf{T}} \arrow[d, dash, dashed] \\
\overline{\lambda}\mu \arrow[r, leftrightarrow, "\Theta, \Psi"] & \underline{\lambda}\mu
\end{tikzcd}
\qquad
\begin{tikzcd}
\overline{\lambda}\mu\tilde{\mu} \arrow[r, leftrightarrow, "\Theta, \Psi"] \arrow[d, dash, dashed] & \underline{\lambda}\mu\mathsf{let} \arrow[d, dash, dashed] \\
\overline{\lambda}\mu\tilde{\mu}_{\mathsf{Q}} \arrow[r, leftrightarrow, "\Theta, \Psi"] \arrow[d, dash, dashed] & \underline{\lambda}\mu\mathsf{let}_{\mathsf{Q}} \arrow[d, dash, dashed] \\
\overline{\lambda}\tilde{\mu} \arrow[r, leftrightarrow, "\Theta, \Psi"] & \underline{\lambda}\mathsf{let}
\end{tikzcd}$$

CBN and CBV fragments. $\overline{\lambda}\mu\tilde{\mu}$ contains a CBN fragment $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{T}}$ and a CBV fragment $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{Q}}$, closed for CBN and CBV reduction, respectively. In addition, $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{T}}$ contains itself a fragment $\overline{\lambda}\mu$ close to $\lambda\mu$, whereas $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{Q}}$ contains itself a fragment $\overline{\lambda}\tilde{\mu}$ which can be "read-back" as a CBV λ-calculus. All this comes from [3].

The following result can only be given here through a diagramatic summary. The proof is by suitable adaptations of Theorem 2.

Theorem 3 (Read-back). *There are CBN fragments* $\underline{\lambda}\mu\mathsf{let}_{\mathsf{T}}$*,* $\underline{\lambda}\mu$*, and CBV fragments* $\underline{\lambda}\mu\mathsf{let}_{\mathsf{Q}}$*,* $\underline{\lambda}\mathsf{let}$ *of* $\underline{\lambda}\mu\mathsf{let}$ *such that appropriate restrictions of* Θ*,* Ψ *establish the isomorphisms illustrated in Fig. 5.* [13]

In $\overline{\lambda}\mu\tilde{\mu}$ the T (resp. Q) subsystem is defined by requiring the right (resp. left) premiss of the left-introduction rule to be a co-value (resp.value). This is remarkably elegant. In $\underline{\lambda}\mu\mathsf{let}$ we have:

T subsystem: obtained by restricting let-expressions to explicit substitutions.

Q subsystem: obtained by restricting HN to HV.

These characterisations of the largest CBN and CBV fragments do not exhibit so clearly the duality of CBN/CBV [3], but nevertheless are either familiar (CBV case [18, 24]) or insightful (CBN case). In fact, the collapse of let-expressions into explicit substitutions becomes an explanation for the fact that traditional natural deduction is CBN.[14]

CBV λ-calculus. The read-back results depicted in Fig. 5 cannot be found in [3, 11]: in *op. cit* they were either not attempted, or not formalized, or not successful. The case of CBV is surprising. There were a number of recent attempts to obtain a CBV λ-calculus (in natural deduction syntax) with a formulation validated by a good correspondence with $\overline{\lambda}\mu\tilde{\mu}$ [22, 11, 12].[15] And yet, nothing like

[13] The naming of systems is as follows. Symbols μ, T, and Q are invariant, when moving between sequent calculus and natural deduction. The remaining symbols obey the correspondence $\overline{\lambda}/\underline{\lambda}$ and $\tilde{\mu}/\mathsf{let}$.

[14] Traditional natural deduction goes even further, by allowing only implicit (*i.e.* meta) substitution.

[15] Such proof-theoretical approach contrasts with the development from [14] to [24], that put forward the computational λ-calculus as the paradigmatic CBV λ-calculus.

Fig. 6. $\underline{\lambda}\mu\mathsf{let}_{\mathsf{Q}}$: a CBV fragment of $\underline{\lambda}\mu\mathsf{let}$

$V ::=$	$x \mid \lambda x.M$	(β)	$\mathcal{E}_v[\mathsf{h}(\lambda x.M)V] \to \langle V/x\rangle\mathcal{E}_v[\mathsf{h}(M)]$
$M, N ::=$	$V \mid \mu a.S$	(σ)	$\langle V/x\rangle S \to [V/x]S$
$H ::=$	$\mathsf{h}(M) \mid HV$	(π)	$\mathcal{E}_v[\mathsf{h}(\mu a.S)] \to [\mathcal{E}_v/a]S$
$S ::=$	$a(H) \mid \mathsf{let}\, x = H \,\mathsf{in}\, S$	(η_μ)	$\mu a.a(\mathsf{h}(M)) \to M,\ a \notin M$
		(η_{let})	$\mathsf{let}\, x = H \,\mathsf{in}\, \mathcal{E}_v[\mathsf{h}(x)] \to \mathcal{E}_v[H],\ x \notin \mathcal{E}_v$

the natural deduction fragment $\underline{\lambda}\mu\mathsf{let}_{\mathsf{Q}}$ that corresponds to $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{Q}}$ was obtained. $\underline{\lambda}\mu\mathsf{let}_{\mathsf{Q}}$ is given in Fig. 6, where *CBV contexts* are: $\mathcal{E}_v ::= a \mid \tilde{\mu}x.S \mid V :: \mathcal{E}_v$.

Rocheteau's definition of CBV [22] does not agree with Ong-Stewart's [16][16]. The attempt to obtain the counterpart of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{Q}}$ in [11] admittedly failed [17]. Finally, the system in [12] places applications and let-expressions in the class of terms. As a result, there is a set of "structural" reduction rules in this system devoted to eliminate expressions which in $\underline{\lambda}\mu\mathsf{let}$ (and therefore in $\overline{\lambda}\mu\tilde{\mu}$) are already ruled out by the organization of the syntactic classes. For instance, one such rule is the "associativity" of let-expressions, that reduces a let whose actual parameter is another let. Such expression does not exist in $\underline{\lambda}\mu\mathsf{let}$.

6 Final remarks

Summary. The system $\underline{\lambda}\mu\mathsf{let}$ is not claimed to be as elegant as $\overline{\lambda}\mu\tilde{\mu}$, but its study is rich and fruitful, summarized in the following high-level lessons:

1. The syntax of natural deduction is non-trivial. Issues: the correct separation between what is primitive and what is derived; the correct organization into syntactic categories of the primitive syntax; only then let-expressions can be added.
2. There is a correspondence between cut in sequent calculus and primitive substitution in natural deduction; let-expressions and primitive substitution are in Curry-Howard correspondence; let-expressions generalize explicit substitutions, and are not exclusive of CBV.
3. It is the amalgamation of let-expressions and explicit substitutions that makes a natural deduction system CBN.
4. The isomorphism $\Theta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$ is a semantics that makes precise why coterms and commands of $\overline{\lambda}\mu\tilde{\mu}$ are "contexts" and "hole-filling" instructions.
5. The isomorphism $\Theta : \overline{\lambda}\mu\tilde{\mu} \to \underline{\lambda}\mu\mathsf{let}$ is a formal recipe for the "read-back" into natural deduction; in particular, is a recipe for the systematic generation of CBV λ-calculi in natural deduction style.

[16] Let $M = (\mu b.S')(\mu a.S)$ and N be the CBV reduct of M. According to Rocheteau, N is $\mu a.[a((\mu b.S')\bullet)/a(\bullet)]S$, whereas, according to Ong-Stewart, N (as well as the CBN reduct of M) is $\mu b.[b(\bullet(\mu a.S))/b(\bullet)]S'$.

[17] It is the calculus called $\lambda^{\eta}_{\mu_{let}} - \eta_{\mu}^{let-app}$ in [11].

Related work. In the body of the paper detailed comparison was made between the proposed system $\underline{\lambda}\mu\mathsf{let}$ and both $\lambda\mu$ and $\overline{\lambda}\mu\tilde{\mu}$ [17, 3], as well as between the CBV fragments of $\underline{\lambda}\mu\mathsf{let}$ and other proposals for CBV λ-calculi recently appeared in the literature [22, 11, 12].

In the recent studies about the correspondence between sequent calculus and natural deduction, one approach is to identify fragments of sequent calculus isomorphic to natural deduction. The initial result is $\lambda_H \cong \lambda$ of [5]. Other contributions of this kind are in [3, 13], although no isomorphism is claimed. The present paper belongs to another approach [26, 7], which pursues extensions of natural deduction isomorphic to full sequent calculus. The isomorphism $\overline{\lambda}\mu\tilde{\mu} \cong \underline{\lambda}\mu\mathsf{let}$ extends to classical logic the intuitionistic $\lambda^{\mathsf{Gtz}} \cong \lambda_{\mathsf{Nat}}$ of [7].

Dyckhoff and Lengrand [4] prove an equational correspondence [23] between LJQ (the Q subsystem of intuitionistic sequent calculus) and Moggi's computational λ-calculus [14]. An isomorphism is to be expected between LJQ and the intuitionistic, Q subsystem of $\underline{\lambda}\mu\mathsf{let}$.

Moggi [14] explained the difference between substitution and let-expressions as the difference between the *composition principles* of two different but related categories: some category with a monad and the corresponding Kleisli category. In the present paper we explain that cut and let are composition principles of two different but related proof-systems: sequent calculus and natural deduction.

Future work. This paper did not aim to contribute to the theory of CBV, but instead to produce, through a proof-theoretical analysis, new calculi for the future investigation of that theory. In spite of differing from other CBV λ-calculi in the literature, the new calculus $\underline{\lambda}\mu\mathsf{let}_{\mathsf{Q}}$, being isomorphic to $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{Q}}$, validates the usual cps semantics [3]. The difference may be telling, however, if we consider reductions instead of equations, and operational aspects like standardization and abstract machines.

Acknowledgments: The author is supported by FCT, through Centro de Matemática, Universidade do Minho. Paul Taylor's macros were used for typesetting diagrams.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
2. I. Cervesato and F. Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.
3. P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000.
4. R. Dyckhoff and S. Lengrand. Call-by-value lambda calculus and LJQ. *Journal of Logic and Computation*, 17:1109–1134, 2007.
5. J. Espírito Santo. Revisiting the correspondence between cut-elimination and normalisation. In *Proceedings of ICALP'2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 600–611. Springer-Verlag, 2000.

6. J. Espírito Santo. An isomorphism between a fragment of sequent calculus and an extension of natural deduction. In M. Baaz and A. Voronkov, editors, *Proceedings of LPAR'02*, volume 2514 of *Lecture Notes in Artificial Intelligence*, pages 354–366. Springer-Verlag, 2002.
7. J. Espírito Santo. The λ-calculus and the unity of structural proof theory. *Theory of Computing Systems*, 45:963–994, 2009.
8. M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. Reasoning with continuations. In *1st Symposium on Logic and Computer Science*. IEEE, 1986.
9. G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1969.
10. T. Griffin. A formulae-as-types notion of control. In *ACM Conf. Principles of Programming Languages*. ACM Press, 1990.
11. H. Herbelin. C'est maintenant qu'on calcule, 2005. Habilitation Thesis.
12. H. Herbelin and S. Zimmermann. An operational account of call-by-value minimal and classical lambda-calculus in "natural deduction" form. In *Proceedings of Typed Lambda Calculi and Applications'09*, volume 5608 of *Lecture Notes in Computer Science*, pages 142–156. Springer-Verlag, 2009.
13. K. Kikuchi. Call-by-name reduction and cut-elimination in classical logic. *Annals of Pure and Applied Logic*, 153:38–65, 2008.
14. E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.
15. S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge, 2001.
16. C-H.L. Ong and C.A. Stewart. A curry-howard foundation for functional computation with control. In *Proc. of Symposium on Principles of Programming Languages (POPL'97)*, pages 215–217. ACM Press, 1997.
17. M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classic natural deduction. In *Int. Conf. Logic Prog. Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer Verlag, 1992.
18. G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.
19. E. Polonovski. Strong normalization of lambda-mu-mu-tilde with explicit substitutions. In Igor (Ed.) Walukiewicz, editor, *Proc. of 7th Int. Conference on Foundations of Software Sciences and Computation Structures (FoSSaCS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 423–437. Springer-Verlag, 2004.
20. D. Prawitz. *Natural Deduction. A Proof-Theoretical Study*. Almquist and Wiksell, Stockholm, 1965.
21. N. Rehof and M. Sorensen. The λ_Δ-calculus. In *TACS'94*, volume 789 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.
22. J. Rocheteau. $\lambda\mu$-calculus and duality: call-by-name and call-by-value. In J. Giesl (Ed.), editor, *Proc. of RTA 2005*, volume 3467 of *Lecture Notes in Computer Science*, pages 204–218. Springer-Verlag, 2005.
23. A. Sabry and M. Felleisen. Reasoning about programms in continuation-passing-style. *LISP and Symbolic Computation*, 6(3/4):289–360, 1993.
24. A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, 1997.
25. A. Troelstra and H. Schwitchtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.
26. J. von Plato. Natural deduction with general elimination rules. *Annals of Mathematical Logic*, 40(7):541–567, 2001.