x

Curry-Howard for sequent calculus at last!

José Espírito Santo[1]

1 Centro de Matemática, Universidade do Minho, Portugal

Abstract

This paper tries to remove what seems to be the remaining stumbling blocks in the way to a full understanding of the Curry-Howard isomorphism for sequent calculus, namely the questions: What do variables in proof terms stand for? What is co-control and a co-continuation? How to define the dual of Parigot's mu-operator so that it is a co-control operator? Answering these questions leads to the interpretation that sequent calculus is a formal vector notation with first-class co-control. But this is just the "internal" interpretation, which has to be developed simultaneously with, and is justified by, an "external" one, offered by natural deduction: the sequent calculus corresponds to a bi-directional, agnostic (w.r.t. the call strategy), computational lambda-calculus. Next, the duality between control and co-control is studied and proved in the context of classical logic, where one discovers that the classical sequent calculus has a distortion towards control, and that sequent calculus is the de Morgan dual of natural deduction.

1998 ACM Subject Classification F.4.1.

Keywords and phrases co-control, co-continuation, vector notation, let-expression, formal substitution, context substitution, computational lambda-calculus, classical logic, de Morgan duality

Digital Object Identifier 10.4230/LIPIcs.TLCA.2015.x

1 Introduction

Despite the anathema *"From an algorithmic viewpoint, the sequent calculus has no* Curry-Howard isomorphism*, because of the multitude of ways of writing the same proof"* [9], more than two decades of research have been dedicated to extend the Curry-Howard isomorphism to the sequent calculus. In its purest form, this is the question: if systems of combinators correspond to Hilbert systems, and the ordinary λ-calculus corresponds to natural deduction, what variant of the λ-calculus does correspond to the sequent calculus? Many computational aspects have been shown to be relevant: pattern matching [2, 19], explicit substitutions[1, 18], abstract machines [4]. But we miss a clear-cut answer to the question in its pure form. The textbook [18] says that the sequent calculus corresponds to explicit substitutions, but it immediately admits "this is just the beginning of the story", as we will see yet again.

We might dismiss the question as closed: maybe the sequent calculus is too complex to admit such a clear-cut computational explanation. But we will not give up, because of the following reason: there are very basic questions, at the bottom of any attempt to understand the sequent calculus computationally, which remain barely uttered and scandalously unanswered; these question we may now give an answer; and this answer opens the way to the desired clear-cut interpretation of the sequent calculus. The questions are:

I. What does a variable stand for in a sequent calculus proof-term?
II. What is the related substitution operation?
III. What is co-control?


© J. Espírito Santo;
licensed under Creative Commons License CC-BY
13th International Conference on Typed Lambda Calculi and Applications (TLCA'15).
Editor: Thorsten Altenkirch; pp. 1–15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

What do we mean? Something very simple. Suppose the proof terms L_2 and L_3 represent derivations of $\Gamma \vdash A$ and $\Gamma, y : B \vdash C$ respectively; and suppose we now infer $\Gamma, x : A \supset B \vdash C$ by left-introduction, building a derivation represented by some term $\mathsf{L}(x, L_2, y.L_3)$. What does this x stand for? What can it be substituted for? Certainly not for a proof-term, as $\mathsf{L}(L_1, L_2, y.L_3)$ corresponds to no derivation.

Of course we can survive by avoiding (not answering) the questions. We may say $\mathsf{L}(x, L_2, y.L_3)$ is a particular form $\langle xL_2/y\rangle L_3$ of "explicit substitution", the general form of which corresponds to cut, we can write cut-elimination rules with this syntax, many of them consisting of permutations of substitutions, all this even without breaking strong normalization [18]. Left introduction is reified with a cut that will to be eliminated, interpreted as a substitution that will not be executed. After all this, we still don't know what that x stands for; and we are busy doing explicitly "substitution", but we do not really know what substitution we mean.

Variables in proof-terms correspond to formulas in the l.h.s. of sequents; and the explicit handling of formulas in the l.h.s. of sequents is typical of the sequent calculus. Now, we have a model of what the handling of formulas in the *r.h.s.* of sequents means computationally: it is the $\lambda\mu$-calculus, which proves that the r.h.s. handling is related to *control operation*. So, by mere formal duality, "co-control" operation has to be involved in the computational interpretation of the sequent calculus. But what is co-control? This is question III.

Toward the system. Of course, our starting point for a co-control operator is the $\tilde{\mu}$-operator of the $\overline{\lambda}\mu\tilde{\mu}$-calculus. But in $\overline{\lambda}\mu\tilde{\mu}$: variables are (and stand for) proof-terms - which allows one to represent left-introductions $\mathsf{L}(x, L_2, y.L_3)$ as certain cuts that cannot be eliminated; and the operational meaning of $\tilde{\mu}$ is given by a reduction rule that triggers an ordinary term-substitution.

We propose to integrate the $\tilde{\mu}$-operator in a system keeping the cut=redex paradigm, where the treatment of variables is very much like in the $\overline{\lambda}$-calculus [10]: variables are not terms, but rather show in a construction that Herbelin writes xl and sees as corresponding to the structural inference of contraction. We prefer to interpret this construction logically as an inference that makes a formula passive on the l.h.s. of the sequent, and computationally as the dual to the "naming" construction aM of the $\lambda\mu$-calculus. Additionally, in our system the operational meaning of $\tilde{\mu}$ is given by a rule that captures some sort of context, triggering some sort of "structural substitution", as in $\lambda\mu$. Variables will stand for that sort of context (in the same way as names in $\lambda\mu$ stand for evaluation contexts or continuations), and the new "structural substitution" is the related kind of substitution. But what sort of context and structural substitution? The co-control question sends us back to the questions I and II.

The answers come from recent work about the isomorphism between $\overline{\lambda}\mu\tilde{\mu}$ and natural deduction [17], where a context-like concept named co-context was introduced in the sequent calculus side. Here we adapt the concept to our intuitionistic setting and rename it *co-continuation*. This is the last ingredient of the sequent calculus we propose, named $\overline{\lambda\tilde{\mu}}$. The system is an extension of $\overline{\lambda}$ with first-class co-control.

Computational interpretation. Having obtained the system, what is its computational interpretation? First-class co-control is a part of the interpretation, but not the whole story - it is only the story that goes beyond $\overline{\lambda}$. Fortunately, all that is needed for the rest of the story is already in place. Surprisingly, we can harvest not one but actually two clear-cut interpretations. Amazingly, the alternative can be seen through the little construction xl.

The first interpretation, the *external* one, is through natural deduction. At the basis of it is the idea, going back to [10], that l represents an evaluation context or continuation (a concept derived from the λ-calculus or natural deduction syntax), and that xl represents a

fill instruction: fill x in the hole of the continuation represented by l. This interpretation, if developed along the lines of [7, 17], is the core of the isomorphism Θ between the system $\overline{\lambda\tilde{\mu}}$ and the natural deduction system (named $\underline{\lambda\mathsf{let}}$) that is designed hand-in-hand with it.

Notice the isomorphism holds at the levels of syntax and rewriting. For this reason, Θ is the ultimate term assignment, that completely side-steps the anathema cited above, and the ultimate realization of the idea, going back to Prawitz [15], of sequent calculus as a meta-system for natural deduction. But, more important, Θ is a computational interpretation, because the target system $\underline{\lambda\mathsf{let}}$ - we will see - has a clear computational meaning itself: it is a computational λ-calculus [12, 16] agnostic w.r.t. the CBN vs CBV alternative.

The second interpretation, the *internal* one, is through the "structural substitution" operation of $\overline{\lambda\tilde{\mu}}$: xl is a fill instruction, but l is the stuff to be filled in the hole of the co-continuation that will land at x. Here l is taken literally, as primitive syntax, not as a continuation. So, we need a good word to describe l computationally. It could be "list" [10] or "spine" [3], but the best is "*vector*", to suggest the (informal) "vector notation" of the λ-calculus [11]. This choice is part of a most needed re-interpretation of a 15-years-old technical result: there is a fragment of $\overline{\lambda}$, here renamed $\overrightarrow{\lambda}$, that is isomorphic to the λ-calculus [6, 3, 5], the isomorphism being essentially the map $\mathcal{P}$ from natural deduction to sequent calculus introduced by Prawitz [15]. But someone has to say loudly that $\overrightarrow{\lambda}$ is a *formal vector notation*. This is the re-interpretation.

Summarizing, we propose two computational interpretations of $\overline{\lambda\tilde{\mu}}$: formal vector notation with first-class co-control, developed in Section 3; and agnostic computational λ-calculus, developed in Section 4. Section 5 looks closely at the duality between control and co-control and extracts unforeseen consequences for logical duality and structural proof theory. Please bear in mind that the words of this introduction just give an approximation of what we want to say. Let the technical development that follows speak for itself.

2 Background

Outside Section 5, we just consider intuitionistic implicational logic. Formulas(=types) are given by: $A, B, C ::= X \mid A \supset B$. In typing systems, contexts Γ are sets of declarations $x : A$ with at most one declaration per variable. In term languages, meta-substitution is denoted with square brackets, as in $[N/x]M$.

2.1 Control operation

Parigot's $\lambda\mu$-calculus [13] is our model for the management of formulas and (co-)variables in the r.h.s. of sequents, when the possibility of a distingushed/active/selected formula exists - a model we wish to "dualize" to the l.h.s. of sequents.

Still we diverge from the original. Let $Q := [b]((\mu a.M)N_1 \cdots N_m)$. In the original formulation of $\lambda\mu$, the reduction of Q proceeds by m applications of "structural reduction", by which μ-abstraction consumes the arguments, one after the other, capturing contexts $[\cdot]N_i$ and triggering a *"structural substitution"*. After m such reduction steps, the resulting term $[b]\mu a.M'$ is reduced by a *renaming* rule, producing $[b/a]M'$. The same effect is obtained with a single, long-step, reduction rule for the μ-operator. Consider the context $\mathcal{C} = [b]([\cdot]N_1 \cdots N_m)$, hence $Q = \mathcal{C}[\mu a.M]$. Now apply the reduction rule $\mathcal{C}[\mu a.M] \to [\mathcal{C}/a]M$, where $[\mathcal{C}/a]M$ is

Figure 1 The $\overline{\lambda}\mu\tilde{\mu}$-calculus

(Terms)	t, u	$::=$	$x \mid \lambda x.t \mid \mu a.c$	(β)	$\langle \lambda x.t \mid u :: e \rangle$	$\to$	$\langle u \mid \tilde{\mu} x.\langle t \mid e \rangle \rangle$
(Co-terms)	e	$::=$	$a \mid u :: e \mid \tilde{\mu} x.c$	$(\tilde{\mu})$	$\langle t \mid \tilde{\mu} x.c \rangle$	$\to$	$[t/x]c$
(Commands)	c	$::=$	$\langle t \mid e \rangle$	(μ)	$\langle \mu a.c \mid e \rangle$	$\to$	$[e/a]c$

context substitution, in whose definition the critical case reads:[1]

$$[\mathcal{C}/a](aP) = \mathcal{C}[P'] \text{ where } P' = [\mathcal{C}/a]P \ . \tag{1}$$

In this formulation, the single reduction rule still says that the μ-operator captures $\mathcal{C}$, while the definition of context substitution says that aP is a *fill instruction*: fill the hole of the $\mathcal{C}$ that lands here with P.

This style with a single reduction rule is - see [17] - the exact reflection in natural deduction of the reduction rule for the μ-operator in the $\overline{\lambda}\mu\tilde{\mu}$-calculus. We are calling the latter reduction rule μ - see Fig. 1 - and so it is natural to call μ the corresponding natural deduction rule [2]. Beware that, sometimes, in $\lambda\mu$, μ names solely the "structural reduction" - which we may see as the control operation proper. In the style with a single rule, μ comprehends the whole control operation, and we will seek a single rule $\tilde{\mu}$ to comprehend the whole co-control operation - and find something different from the rule $\tilde{\mu}$ of $\overline{\lambda}\mu\tilde{\mu}$, despite the compelling symmetry of the latter.

2.2 Vector notation

The vector notation for the λ-calculus is the following definition of the λ-terms [11]:

$$M, N, P, Q \ ::= \ \lambda x.M \mid x\vec{N} \mid (\lambda x.M)N\vec{N}$$

According to this definition, λ-terms have three forms, which we call *first, second and third forms*. The advantage of this notation is that the head variable/redex is visible, and the β-normal forms are obtained by omitting the third form of terms in the above grammar.

This notation is informal, since many details are left unspecified. For instance: Are vectors a separate syntactic class, or do the second and third forms correspond to families of rules? How is substitution defined? Notice that, in the second form, it does not make sense to replace x by another term.

Let us introduce the *relaxed* vector notation:

$$M, N, P, Q \ ::= \ \lambda x.M \mid x\vec{N} \mid M\vec{N}$$

Some redundancy is now allowed, as the same ordinary λ-term can be represented in many ways. By analyzing the third form, we find four cases. One corresponds to the third form of the original vector notation, the other three may be simplified as follows:

$$\begin{array}{lrcl} (\epsilon) & M[] & = & M \\ (\pi_1) & (x\vec{Q})N\vec{N} & = & x(\vec{Q}N\vec{N}) \\ (\pi_2) & (P\vec{Q})N\vec{N} & = & P(\vec{Q}N\vec{N}) \end{array}$$

[1] Structural substitution is the particular case $[a([\cdot]N)/a]\_$ of context substitution.
[2] In [17] both rules are called σ_μ.

Figure 2 Typing rules of the $\overline{\lambda}\tilde{\mu}$-calculus

$$Ax\ \frac{}{\Gamma|A \vdash [] : A} \qquad Cut\ \frac{\Gamma \vdash t : A \quad \Gamma|A \vdash k : B}{\Gamma \vdash tk : B} \qquad Pass\ \frac{\Gamma, x : A|A \vdash k : B}{\Gamma, x : A \vdash x\hat{}k : B}$$

$$L\supset\ \frac{\Gamma \vdash u : A \quad \Gamma|B \vdash k : C}{\Gamma|A \supset B \vdash u :: k : C} \qquad R\supset\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B} \qquad Act\ \frac{\Gamma, x : B \vdash v : B}{\Gamma|A \vdash \tilde{\mu}x.v : B}$$

Here [] represents the empty vector. The simplifications were given names that will link them with the reduction rules of sequent calculus.

The third form of the original vector notation is a β-redex and hence it means something like "call the head function with the first argument of the vector". The simplifications that have arisen by relaxing the vector notation can be read as rules for vector bookkeeping: garbage collect an empty vector (ϵ), or append chained vector (π_i). Much more difficult is to recognize in the second form $x\vec{N}$ an instruction for action on the vector $\vec{N}$: this is visible in a more general system with co-control, like the sequent calculus we are about to introduce, where the base case for vectors is not just [].

3 Sequent calculus

3.1 The $\overline{\lambda}\tilde{\mu}$-calculus

The proof-expressions of $\overline{\lambda}\tilde{\mu}$ [3] are given by the following grammar:

$$\begin{array}{rrcl} \text{(Terms)} & t, u, v & ::= & \lambda x.t \,|\, x\hat{}k \,|\, tk \\ \text{(Generalized vectors)} & k & ::= & [] \,|\, \tilde{\mu}x.v \,|\, u :: k \end{array}$$

The typing rules are in Fig. 2. They handle two kinds of sequents: $\Gamma \vdash t : A$ and $\Gamma|A \vdash k : B$. The distinguished formula A in the latter is not exactly a "stoup" or a focused formula, because the operator $\tilde{\mu}x.t$ may select an arbitrary formula from the context Γ. The construction $x\hat{}k$ comes from the $\overline{\lambda}$-calculus, but here it forms a pair with $\tilde{\mu}x.t$: logically, these are a activation/passification pair, in the style of the $\lambda\mu$-calculus, but acting on the l.h.s. of sequents.

The notation $\tilde{\mu}x.t$ comes from $\overline{\lambda}\mu\tilde{\mu}$ [4]; but here, contrary to what happens in $\overline{\lambda}\mu\tilde{\mu}$, the reduction rule that defines the behavior of $\tilde{\mu}$ does not trigger a term-substitution, but rather a context-substitution, in the style of the above presentation of the μ-operator. The construction $x\hat{}k$ is easily recognized as the accompanying fill-instruction, and what remains is to pin down the right notion of context that $\tilde{\mu}$ will capture. The notion is this:

$$\mathcal{H} ::= x\hat{}[\cdot] \,|\, t([\cdot]) \,|\, \mathcal{H}[u :: [\cdot]]$$

These expressions are called *co-continuations*. Later we will argue they are logically dual to continuations.

The reduction rules of $\overline{\lambda}\tilde{\mu}$ are in Fig. 3. Let $\pi := \pi_1 \cup \pi_2$. The co-control rule $\tilde{\mu}$ triggers the context substitution $[\mathcal{H}/x]\_$, in whose definition the only non-routine case is:

$$[\mathcal{H}/x](x\hat{}k) = \mathcal{H}[k'] \text{ with } k' = [\mathcal{H}/x]k \ .$$

[3] We would have adopted the name $\overline{\lambda}\tilde{\mu}$ (to suggest $\overline{\lambda} + \tilde{\mu}$), had it not been already in use [4].

Figure 3 Reduction rules of $\overline{\lambda\tilde{\mu}}$

$$\begin{array}{llrcl@{\qquad}llrcl} (\beta) & (\lambda x.t)(u :: k) & \to & (u(\tilde{\mu}x.t))k & (\epsilon) & t[] & \to & t \\ (\tilde{\mu}) & \mathcal{H}[\tilde{\mu}x.t] & \to & [\mathcal{H}/x]t & (\pi_1) & (x\hat{}k)k' & \to & x\hat{}(k@k') \\ & & & & (\pi_2) & (tk)k' & \to & t(k@k') \end{array}$$

This equation gives the meaning of $x\hat{}k$: fill k in the hole of the $\mathcal{H}$ that will substitute x. The π_i-rules employ concatenation of generalized vectors $k@k'$, defined by the obvious equations $[]@k' = k'$ and $(u :: k)@k' = u :: (k@k')$, together with $(\tilde{\mu}x.t)@k' = \tilde{\mu}x.tk'$.

Rule $\tilde{\mu}$ eliminates all occurrences of the $\tilde{\mu}$-operator. The remaining rules eliminate all occurrences of cuts tk. So the $\beta\tilde{\mu}\epsilon\pi$-normal forms correspond to a well-known representation of β-normal λ-terms. There is a critical pair generated by rules $\tilde{\mu}$ and π. This is the *call-by-name vs call-by-value* dilemma [4].

Two particular cases of the reduction rule $\tilde{\mu}$ are:

$$(\rho) \qquad y\hat{}(\tilde{\mu}x.t) \to [y\hat{}[\cdot]/x]t \qquad\qquad (\sigma) \quad u(\tilde{\mu}x.t) \to [u([\cdot])/x]t$$

We let $\tau := \tilde{\mu}\backslash(\rho \cup \sigma)$. The particular case ρ may be called the *renaming* rule (as sometime does the "dual" rule in the $\lambda\mu$-calculus). Indeed, the particular case $[y\hat{}[\cdot]/x]t$ of context substitution is almost indistinguishable from a substitution operation that renames variables, since the critical case of its definition reads[4]

$$[y\hat{}[\cdot]/x](x\hat{}k) = y\hat{}k' \text{ with } k' = [y\hat{}[\cdot]/x]k \ .$$

If we wanted a set of small step $\tilde{\mu}$-rules, in the style of the original $\lambda\mu$-calculus, we would have taken ρ and σ, together with $u :: (\tilde{\mu}x.t) \to \tilde{\mu}x.[x\hat{}(u :: [\cdot])/x]t$. The particular case $[x\hat{}(u :: [\cdot])/x]t$ of context substitution gives a form of "structural substitution" dual to that found in $\lambda\mu$.

3.2 The proof theory of vector notation

We consider notorious fragments of $\overline{\lambda\tilde{\mu}}$. First we do a kind of reconstruction of $\overline{\lambda}$. Next we identify two subsystems of our version of $\overline{\lambda}$ that say something about vector notation.

The $\rho\tau$-normal-forms of $\overline{\lambda\tilde{\mu}}$ are given by:

$$t, u, v \ ::= \ \lambda x.t \,|\, x\hat{}l \,|\, tk \qquad k \ ::= \ l \,|\, \tilde{\mu}x.t \qquad l \ ::= \ [] \,|\, u :: l$$

These are the $\overline{\lambda}$-expressions, if we recognize $t(\tilde{\mu}x.v)$ as the "mid-cut", and if we ignore the other forms of explicit substitution or concatenation that are primitive in the original formulation of $\overline{\lambda}$. In terms of logical derivations, $\rho\tau$-normalization ends in the focused fragment LJT [10] of the sequent calculus (the fragment corresponding to $\overline{\lambda}$). In this sense $\rho\tau$-reduction is a focalization process. See the focalization theorem below, saying that $\rho\tau$-normal forms exist and are unique.

The $\tilde{\mu}$-rule is now restricted to the σ-rule, but the critical pair with π remains. This is solved by a syntactical trick, that chooses the call-by-name option: erase the case $\tilde{\mu}x.t$ from k's (so that there is no more a distinction between k's and l's, there is a single class of *vectors* ranger over by l), but break the cut tk into the two cases tl ("head-cut") and $t(\tilde{\mu}x.v)$.

[4] We say "almost" because, don't forget, variables are not expressions *per se*.

A π-redex $(x\hat{}l)l'$ or $(tl)l'$ is no longer a σ-redex, and a mid-cut $(y\hat{}l)\tilde{\mu}x.v$ or $(ul)\tilde{\mu}x.v$ can never be reduced by π (as it could in $\overline{\lambda\tilde{\mu}}$). This concludes the reconstruction of $\overline{\lambda}$.

This version of $\overline{\lambda}$ is equipped with β, σ, ϵ and π. We now consider the R-normal forms, with $R = \sigma$ or $R = \sigma \cup \epsilon \cup \pi$.

- In the first case, terms have the forms $\lambda x.t$, $x\hat{}l$ or tl, equipped with a rule β that corresponds to $\overline{\lambda}$'s β-rule followed by σ-normalization. The rules ϵ and π remain. Let us call this system $\overset{\hookrightarrow}{\lambda}$.
- In the second case, terms have the forms $\lambda x.t$, $x\hat{}l$ or $(\lambda x.t)(u :: l)$, equipped solely with a rule β that corresponds to $\overline{\lambda}$'s β-rule followed by $\sigma\epsilon\pi$-normalization. Let us call this system $\overrightarrow{\lambda}$.

Around 15 year ago [6, 3, 5], the system $\overrightarrow{\lambda}$ was identified and proved isomorphic to the ordinary λ-calculus, with the isomorphism comprising a bijection between the sets of terms and an isomorphism of β-reduction relations. In the author's opinion, this little technical fact has a tremendous importance for the Curry-Howard isomorphism that has never been recognized. First, $\overrightarrow{\lambda}$ has a clear and revealing computational interpretation: it is a *formal vector notation*. It is a concrete definition of the notation (it says what vectors are, how substitution is defined, etc.) in perfect correspondence with a logical calculus. Second, we can reconstruct from this interpretation the interpretation of full sequent calculus $\overline{\lambda\tilde{\mu}}$ by walking back the path that led from $\overline{\lambda\tilde{\mu}}$ to $\overrightarrow{\lambda}$. Clearly, $\overset{\hookrightarrow}{\lambda}$ is a formalization of the relaxed vector notation we introduced in Section 2; and, since $\overset{\hookrightarrow}{\lambda}$ is the fragment of normal forms of $\overline{\lambda\tilde{\mu}}$ w.r.t. the co-control rule $\tilde{\mu}$, $\overline{\lambda\tilde{\mu}}$ can be interpreted as a *formal, relaxed vector notation with first-class co-control*.

4 Natural deduction

The interpretation developed before is an internal and literal one: vectors are vectors; and co-control is understood in a formal way, as dual to control. We now develop a natural deduction system $\underline{\lambda\mathsf{let}}$ that is isomorphic to $\overline{\lambda\tilde{\mu}}$. The isomorphism gives a new, external interpretation, which recovers the view that (some) vectors are "evaluation contexts"[10]; it justifies the design of $\overline{\lambda\tilde{\mu}}$, namely its notion of $\mathcal{H}$ and reduction rule $\tilde{\mu}$; finally, it allows the transfer of properties among the two systems.

4.1 The $\underline{\lambda\mathsf{let}}$-calculus

The proof-expressions of the calculus are given by:

$$\begin{array}{llrcl} \text{(Terms)} & M, N, P & ::= & \lambda x.M \mid \mathsf{app}(H) \mid \mathsf{let}\, x := H \,\mathsf{in}\, P \\ \text{(Heads)} & H & ::= & x \mid \mathsf{hd}(M) \mid HN \end{array}$$

Notice that variables x and applications HN are not terms. A head $\mathsf{hd}(M)$ is called a *head term*.

The typing rules are in Fig. 4. They handle two kinds of sequents: $\Gamma \vdash M : A$ and $\Gamma \rhd H : A$. Four rules are standard, with the appropriate kind of sequent determined by the kind of expression being typed. The remaining two rules switch the kind of sequent, and are called *coercions*. However, despite the superficial impression, the two coercions are quite different, one being called weak and the other strong. Normalization will tell them apart radically.

A *normal* derivation is one without occurrences of *Let* and *SCoercion*.

Figure 4 Typing rules of $\underline{\lambda\mathsf{let}}$

$$\frac{}{\Gamma, x : A \rhd x : A}\ Hyp \qquad \frac{\Gamma \rhd H : A \quad \Gamma, x : A \vdash P : B}{\Gamma \vdash \mathsf{let}\, x := H \,\mathsf{in}\, P : B}\ Let \qquad \frac{\Gamma \rhd H : A}{\Gamma \vdash \mathsf{app}(H) : A}\ WCoercion$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ Intro \qquad \frac{\Gamma \rhd H : A \supset B \quad \Gamma \vdash N : A}{\Gamma \rhd HN : B}\ Elim \qquad \frac{\Gamma \vdash M : A}{\Gamma \rhd \mathsf{hd}(M) : A}\ SCoercion$$

Figure 5 Reduction rules of $\underline{\lambda\mathsf{let}}$

$$\begin{array}{rrcl}
(\mathsf{beta}) & \mathsf{hd}(\lambda x.M)N & \to & \mathsf{hd}(\mathsf{let}\, x := \mathsf{hd}(N) \,\mathsf{in}\, M) \\
(\mathsf{let}) & \mathsf{let}\, x := H \,\mathsf{in}\, P & \to & [H/x]P \\
(\mathsf{triv}) & \mathsf{app}(\mathsf{hd}(M)) & \to & M \\
(\mathsf{head}_1) & \mathsf{hd}(\mathsf{app}(H)) & \to & H \\
(\mathsf{head}_2) & \mathcal{K}[\mathsf{hd}(\mathsf{let}\, x := H \,\mathsf{in}\, P)] & \to & \mathsf{let}\, x := H \,\mathsf{in}\, \mathcal{K}[\mathsf{hd}(P)]
\end{array}$$

▶ **Theorem 1** (Subformula property). *Every formula (resp. every formula, including A) occurring in a normal derivation of $\Gamma \vdash M : A$ (resp. $\Gamma \rhd H : A$) is subformula of A or of some formula in Γ (resp. is a subformula of some formula in Γ).*

So, the weak coercion loses information regarding the subformula property, while the strong coercion potentially violates that property.

The reduction rules of $\underline{\lambda\mathsf{let}}$ are in Fig. 5. Rule let triggers ordinary substitution $[H/x]P$, while rule head_2 employs certain contexts that we call *continuations*:

$$\mathcal{K} ::= \mathsf{app}([\cdot]) \mid \mathsf{let}\, x := [\cdot] \,\mathsf{in}\, P \mid \mathcal{K}[[\cdot]N]$$

We single out two particular cases of let: ren, when $H = x$; and sub, when $H = \mathsf{hd}(M)$. We put $\mathsf{t} := \mathsf{let}\backslash(\mathsf{ren} \cup \mathsf{sub})$. Let $\mathsf{head} := \mathsf{head}_1 \cup \mathsf{head}_2$. Notice that rules beta and head_1 are relations on heads. The normal forms w.r.t. all reduction rules are given by:

$$M \quad ::= \quad \lambda x.M \mid \mathsf{app}(H) \qquad H \quad ::= \quad x \mid HN$$

That is, these normal forms are characterized by the absence of occurrences of lets and $\mathsf{hd}()$. Lets are eliminated by let whereas all the other rules concur to eliminate the coercion $\mathsf{hd}()$. So, $\mathsf{beta}, \mathsf{triv}, \mathsf{let}, \mathsf{head}$-reduction is *normalization*, that is, the reduction to a form corresponding to normal derivations.

4.2 Isomorphism

See Fig. 6 for the map $\Theta : \overline{\lambda\tilde{\mu}} \longrightarrow \underline{\lambda\mathsf{let}}$. There is actually a function $\Theta : \overline{\lambda\tilde{\mu}}\text{-}Terms \longrightarrow \underline{\lambda\mathsf{let}}\text{-}Terms$, together with an auxiliary function $\Theta : \underline{\lambda\mathsf{let}}\text{-}Heads \times \overline{\lambda\tilde{\mu}}\text{-}Vectors \longrightarrow \underline{\lambda\mathsf{let}}\text{-}Terms$. Let $\Theta(t) = M$, $\Theta(u_i) = N_i$ and $\Theta(v) = P$. The idea is to map, say, $t(u_1 :: u_2 :: \tilde{\mu}x.v)$ to $\mathsf{let}\, x := hd(M)N_1N_2 \,\mathsf{in}\, P$, and $x\hat{}(u_1 :: u_2 :: [])$ to $\mathsf{app}(xN_1N_2)$: left-introductions are replaced by applications, inverting the associativity of non-abstractions.

▶ **Theorem 2** (Isomorphism). *Map Θ is a sound bijection between the set of $\overline{\lambda\tilde{\mu}}$-terms and the set of $\underline{\lambda\mathsf{let}}$-terms (whose inverse Ψ is shown in Fig. 7). Moreover, let R be rule β (resp. $\tilde{\mu}$, ϵ, π) of $\overline{\lambda\tilde{\mu}}$, and let R' be rule* beta *(resp.* let, triv, head*) of $\underline{\lambda\mathsf{let}}$. Then, $t \to_R t'$ in $\overline{\lambda\tilde{\mu}}$ iff $\Theta t \to_{R'} \Theta t'$ in $\underline{\lambda\mathsf{let}}$.*

Figure 6 Map $\Theta : \overline{\lambda\tilde{\mu}} \longrightarrow \underline{\lambda\mathsf{let}}$

$$\begin{array}{rclcrcl} \Theta(\lambda x.t) & = & \lambda x.\Theta t & & \Theta(H,[]) & = & \mathsf{app}(H) \\ \Theta(x\hat{}k) & = & \Theta(x,k) & & \Theta(H,\tilde{\mu}x.t) & = & \mathsf{let}\, x := H \,\mathsf{in}\, \Theta t \\ \Theta(tk) & = & \Theta(hd(\Theta t),k) & & \Theta(H,u::k) & = & \Theta(H\Theta u,k) \end{array}$$

Figure 7 Map $\Psi : \underline{\lambda\mathsf{let}} \longrightarrow \overline{\lambda\tilde{\mu}}$

$$\begin{array}{rclcrcl} \Psi(\lambda x.M) & = & \lambda x.\Psi M & & \Psi(x,k) & = & x\hat{}k \\ \Psi(\mathsf{app}(H)) & = & \Psi(H,[]) & & \Psi(\mathsf{hd}(M),k) & = & (\Psi M)k \\ \Psi(\mathsf{let}\, x := H \,\mathsf{in}\, P) & = & \Psi(H,\tilde{\mu}x.\Psi P) & & \Psi(HN,k) & = & \Psi(H,(\Psi N)::k) \end{array}$$

The real action of the isomorphism happens in the translation of non-abstractions. Every non-abstraction $\underline{\lambda\mathsf{let}}$-term has the form $\Theta(H,k)$, and $\Psi\Theta(H,k) = \Psi(H,k)$. Every non-abstraction $\overline{\lambda\tilde{\mu}}$-term has the form $\Psi(H,k)$, and $\Theta\Psi(H,k) = \Theta(H,k)$. So non-abstractions have the form $\Theta(H,k)$ (natural deduction) or $\Psi(H,k)$ (sequent calculus), and the isomorphism action between them is just to interchange Θ and Ψ in these expressions.

Ψ can be extended to continuations, establishing a bijection with vectors: $\Psi(\mathsf{app}([\cdot])) = []$, $\Psi(\mathsf{let}\, x := [\cdot] \,\mathsf{in}\, P) = \tilde{\mu}x.\Psi P$ and $\Psi(\mathcal{K}[[\cdot]N]) = \Psi(N) :: \Psi(\mathcal{K})$. As we knew, continuations are typed "on the left": the sequent calculus rules for typing vectors are derived typing rules in natural deduction for typing continuations.

Similarly, Θ can be extended to co-continuations, establishing a bijection with heads: $\Theta(x\hat{}[\cdot]) = x$, $\Theta(t([\cdot]) = \mathsf{hd}(\Theta t)$ and $\Theta(\mathcal{H}[u :: [\cdot]]) = \Theta(\mathcal{H})\Theta u$. This tells us how to type co-continuations [17]: the natural deduction rules for typing heads are derived typing rules in sequent calculus for typing co-continuations, and so co-continuations are typed "on the right".

Let us call Θ the inverse of the bijection between continuations and vectors. Then it is easy to prove that $\Theta(H,k) = \Theta(k)[H]$. This tells us that non-abstractions in $\overline{\lambda\tilde{\mu}}$ are fill instructions, and that Θ executes these instructions. For instance, $\Theta(tk) = \Theta(\mathsf{hd}(\Theta t),k) = \Theta(k)[\mathsf{hd}(\Theta t)]$; so tk means "fill $\mathsf{hd}(M)$ in the hole of continuation $\mathcal{K}$", with $M = \Theta t$ and $\mathcal{K} = \Theta(k)$; and $\Theta(tk)$ is the result of such filling. Similarly, $x\hat{}k$ means "fill x in the hole of $\mathcal{K}$". So Θ realizes again the idea, going back to Prawitz [15], that sequent calculus derivations are instructions for building natural deduction proofs.

4.3 Forgetfulness

The *forgetful map* $|\cdot|$ translates $\underline{\lambda\mathsf{let}}$-expressions to λ-terms by erasing occurrences of the coercion $\mathsf{hd}(\cdot)$, forgetting the distinction between terms and heads, de-sugaring let-expressions (*i.e.* translating them as β-redexes), and mapping $|\mathsf{app}(H)| = I|H|$, where $I = \lambda x.x$. The following results about $\overline{\lambda\tilde{\mu}}$ are proved through the analysis of this simple translation of the isomorphic calculus $\underline{\lambda\mathsf{let}}$.

▶ **Theorem 3** (Strong normalization). *Every typable term of $\overline{\lambda\tilde{\mu}}$ (resp. of $\underline{\lambda\mathsf{let}}$) is $\beta\epsilon\tilde{\mu}\pi$-SN (resp.* beta triv let head*-SN).*

▶ **Theorem 4** (Focalization). *Every term of $\overline{\lambda\tilde{\mu}}$ has a unique $\rho\tau$-normal form, which is a $\overline{\lambda}$-term (representing a LJT-proof).*

4.4 Computational interpretation

We argue that $\underline{\lambda\mathsf{let}}$ is a *bidirectional, agnostic, computational* λ*-calculus*. The word "bidirectional" comes from [14], where the organization of a typing system for λ-terms with two kinds of sequents (for synthesis, like $\Gamma \vdash M : A$, and for checking, like $\Gamma \rhd H : A$) is already found. The word "agnostic" comes from [19] and means coexistence or superimposition of call-by-name (CBN) and call-by-value (CBV).

Let us go back to Fig. 5. Rule β generates a let-expression, which can be executed by the separate rule let. Let-expressions enjoy "associativity": the particular case of head_2 where $\mathcal{K} = \mathsf{let}\, y := [\cdot]\, \mathsf{in}\, Q$ reads $\mathsf{let}\, y := \mathsf{hd}(\mathsf{let}\, x := H\, \mathsf{in}\, P)\, \mathsf{in}\, Q \to \mathsf{let}\, x := H\, \mathsf{in}\, \mathsf{let}\, y := \mathsf{hd}(P)\, \mathsf{in}\, Q$. In addition, there is a pair of reduction rules to cancel a sequence of two coercions. So we might view $\underline{\lambda\mathsf{let}}$ as a sort of computational λ-calculus [12, 16] where rule let does not require the computation of a value prior to substitution triggering, and where a pair of trivial reduction rules (triv and head_1) eliminates odd accumulations of coercions caused by a clumsy syntax.

However, this is not the right view. head_1 works together with head_2 to reduce every non-abstraction term to one of the forms $\mathcal{K}[x]$ or $\mathcal{K}[\mathsf{hd}(\lambda x.M)]$. This is a hint of what we see next: all reduction rules of $\underline{\lambda\mathsf{let}}$ have quite clear roles in CBV and CBN computation, and through these roles we will understand how different rules triv and head_1 are.

Let us make a technical point. Rule head_1 is a relation on heads. As with all other reduction rules, head_1 generates by compatible closure a relation $\to_{\mathsf{head}_1}$ on heads and another on terms. The relation $\to_{\mathsf{head}_1}$ on terms would have been the same, had we taken the rule head_1 as the relation on terms $\mathcal{K}[\mathsf{hd}(\mathsf{app}(H))] \to \mathcal{K}[H]$. A similar remark applies to beta. In the discussion of CBN and CBV that follows, we take head_1 and beta in their alternative formulation, so that it makes sense to speak about head_1- or beta-reduction at root position of a term.

CBN and CBV are defined through priorities among reduction rules [4]:

- CBV strategy: reduction at root position of a closed, non-abstraction term with priority given to head.
- CBN strategy: reduction at root position of a closed, let-free, non-abstraction term with priority given to triv.

We will give an alternative characterization of these strategies. For CBN we need the rule $\mathcal{K}[\mathsf{hd}(\lambda x.M)N] \to \mathcal{K}[\mathsf{hd}([\mathsf{hd}(N)/x]M)]$, which we call $CBN - \mathsf{beta}$, and is obtained by beta followed by let.

▶ **Theorem 5** (Agnosticism). *The following is an equivalent description of the CBN and CBV strategies. In this description, "reduction" means root-position reduction of a closed, non-abstraction term. We assume additionally that the initial term is let-free.*

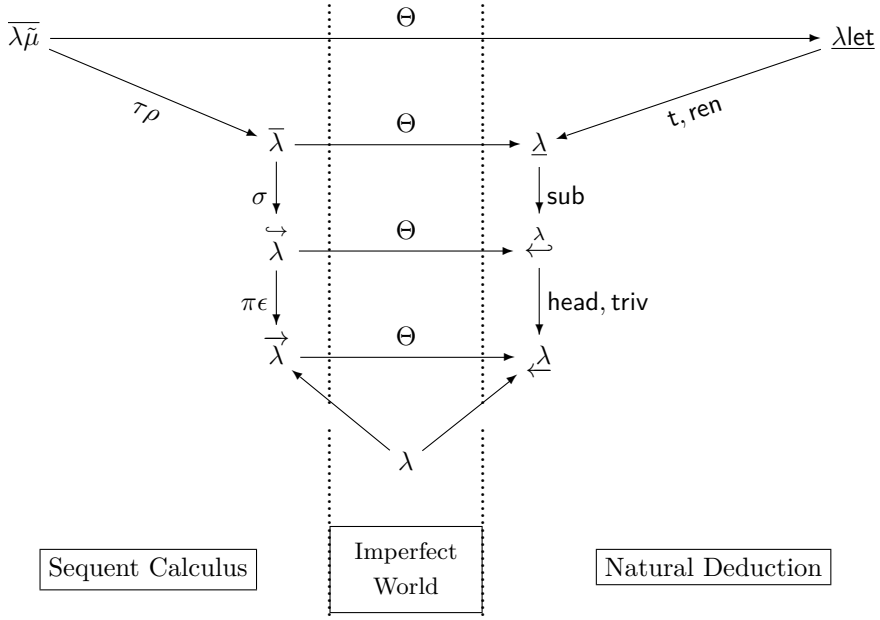
- *CBV. Do* head*-reduction as long as possible, until the term becomes either a* beta*,* let*, or* triv *redex. In the two first cases, reduce and restart; in the last case, reduce to return the computed abstraction.*
- *CBN. Do* head*-reduction as long as possible, until the term becomes either a* beta *or* triv *redex. In the first case, reduce (with* $CBN - \mathsf{beta}$*) and restart; in the last case, reduce to return the computed abstraction.*

This description is not in terms of priorities, but rather reveals the shared organization of the computation and the roles of the different reduction rules, which are the same in both strategies - see Fig. 8. The shared organization, in turn, shows how "superimposed" CBV and CBN are in the system, in other words, how agnostic the system is.

Figure 8 Agnosticism: the shared organization of CBN and CBV strategies

		CBV	CBN
iteration	pre-processing	$\mathsf{head}_1 + \mathsf{head}_2$	head_1
of the computation cycle	real computation	beta + let	beta; let
return		triv	triv

Figure 9 The sequent calculus/natural deduction mirror.



4.5 Epilogue

An easy consequence of the isomorphism Θ is that the space of calculi in the sequent calculus format has a mirror image in natural deduction. See Fig. 9 for a roadmap. The λ-calculus is displayed in the "imperfect world" - the terminology is inspired in Remark 2.1. of [4].

The two computational interpretations of sequent calculus are collected in Fig. 10. The external interpretation works like this: in Fig. 10 the shown interpretation is that of $\underline{\lambda\mathsf{let}}$; $\overline{\lambda\tilde{\mu}}$ is a language of instructions for $\underline{\lambda\mathsf{let}}$ (recall Section 4.2); the behavior of $t \in \overline{\lambda\tilde{\mu}}$ is the isomorphic behavior of $\Theta t \in \underline{\lambda\mathsf{let}}$ written in the language of instructions.

Why is co-control almost invisible in natural deduction? Why does it boil down to the low-profile rule let, which is just a substitution triggering rule? The explanation is in the good old associativity of "applicative terms" [10]. In natural deduction, the control operator shows up in the hidden part of an applicative terms, like $(\mu a.M)N_1 \cdots N_m$. Since we want to get to this M, we collect the outer stuff in the context-like structure $\mathcal{K} = [\cdot]N_1 \cdots N_m$ and trigger a context-substitution $[\mathcal{K}/a]M$. But, in natural deduction, the co-control operator is disguised in let-expressions $\mathsf{let}\, x := HN_1 \cdots N_m \,\mathsf{in}\, P = \Theta(HN_1 \cdots N_m, \tilde{\mu}x.P)$, and so is already at the surface. So we may proceed by ordinary substitution $[HN_1 \cdots N_m/x]P$.

Figure 10 Curry-Howard for sequent calculus

sequent calculus	$\overline{\lambda\tilde{\mu}}$	internal interpretation	external interpretation
	t	λ-term in formal vector notation with first-class co-control	bi-directional, agnostic, computational λ-term
right intro.	$\lambda x.t$	λ-abstraction	λ-abstraction
contraction	$x\hat{}k$	2nd form of vector notation fill k in the co-continuation x	$\mathcal{K}[x]$
cut	tk	3rd form of vector notation	$\mathcal{K}[\mathsf{hd}(M)]$
	k	generalized vector	continuation $\mathcal{K}$
axiom	$[]$	empty vector	$\mathsf{app}([\cdot])$
left selection	$\tilde{\mu}x.t$	co-control operator	$\mathsf{let}\, x := [\cdot]\, \mathsf{in}\, P$
left intro.	$u :: k$	vector constructor	$\mathcal{K}[[\cdot]N]$
cut elim. + + focalization	red. rules		
key-step	β	function call	function call
left-sel. elim.	$\tilde{\mu}$	co-control operation	subst. triggering (let)
focalization	τ	co-control operation proper	subst. triggering (t)
focalization	ρ	subst. triggering (renaming)	subst. triggering (ren)
right-perm.	σ	subst. triggering	subst. triggering (sub)
right-perm.	ϵ	erasure of empty vector	return (triv)
left-perm.	π	append of iterated vectors	re-associate (head)

5 Duality

Let us check that co-control, as formulated in $\overline{\lambda\tilde{\mu}}$, is dual to control. We will expand $\overline{\lambda\tilde{\mu}}$ to a self-dual system where control and co-control are each other's dual. This is achieved with three steps.

First step: to unify $\overline{\lambda\tilde{\mu}}$ and $\underline{\lambda\mathsf{let}}$. Some hints are at the end of Section 4.2, the idea comes from [7]. Every non-abstraction term of $\overline{\lambda\tilde{\mu}}$ has the form $\Psi(H, k)$. So we unify $x\hat{}k$ and tk as $\overline{\Psi}(H, k)$ and allow in $\overline{\lambda\tilde{\mu}}$ a new syntactic class $H ::= x \mid \mathsf{hd}(t)$. Every non-abstraction term of $\underline{\lambda\mathsf{let}}$ has the form $\Theta(H, k)$. Se we unify $\mathsf{app}(H)$ and $\mathsf{let}\, x := H\, \mathsf{in}\, P$ as $\underline{\Theta}(H, k)$ and allow in $\underline{\lambda\mathsf{let}}$ a new syntactic class $k ::= [] \mid \tilde{\mu}x.P$. Next let us unify $\overline{\Psi}(H, K)$ and $\underline{\Theta}(H, k)$ as $\chi(H|k)$. After this we realize that $\overline{\lambda\tilde{\mu}}$ and $\underline{\lambda\mathsf{let}}$ are partial views of the same system (the former lacks HN, the latter lacks $u :: k$). So let t and M range over the same set of proof terms. Continuations are internalized and coincide with vectors, co-continuations are internalized and coincide with heads, context-substitution is internalized as ordinary substitution. We work modulo $\chi(HN|k) = \chi(H|N :: k)$, which abstracts the single difference between $\overline{\lambda\tilde{\mu}}$ and $\underline{\lambda\mathsf{let}}$.

Second step: to add control. We introduce the class of "commands" $c ::= \chi(H|k)$, and a non-abstraction term is now $\mu a.c$. Continuations are now given by $k ::= a \mid \tilde{\mu}x.c \mid u :: k$. Sequents have full r.h.s's: for instance, $\Gamma|k : A \vdash \Delta$. Logically, we moved to classical logic.

Third step: to complete the duality. We add the dual implication $A - B$, and the class of co-terms $r ::= \tilde{\lambda}a.r \mid \tilde{\mu}x.c$. The place left vacant in the grammar of continuations by the move of $\tilde{\mu}x.c$ is occupied by the new construction $\tilde{\mathsf{hd}}(r)$. The full suite of sequents is:

$$\Gamma \vdash t : A|\Delta \qquad \Gamma|r : A \vdash \Delta \qquad \Gamma \rhd H : A|\Delta \qquad \Gamma|k : A \rhd \Delta \qquad c : (\Gamma \vdash \Delta)$$

Figure 11 The unified calculus

$$
\begin{array}{rrcl}
\text{(Terms)} & t,u,M,N & ::= & \lambda x.t \mid \mu a.c \\
\text{(Co-terms)} & r,s & ::= & \tilde{\lambda} a.r \mid \tilde{\mu} x.c \\
\text{(Co-continuations)} & H & ::= & x \mid \mathsf{hd}(M) \mid HN \mid H \,\tilde{::}\, r \\
\text{(Continuations)} & k & ::= & a \mid \tilde{\mathsf{hd}}(r) \mid r \tilde{\frown} k \mid u :: k \\
\text{(Commands)} & c & ::= & \chi(H|k)
\end{array}
$$

$$
\begin{array}{rrcl}
(\beta) & \chi(\mathsf{hd}(\lambda x.t)|u :: k) & \to & \chi(\mathsf{hd}(u)|\tilde{\mathsf{hd}}(\tilde{\mu} x.\chi(\mathsf{hd}(t)|k))) \\
(\tilde{\beta}) & \chi(H \,\tilde{::}\, s|\tilde{\mathsf{hd}}(\tilde{\lambda} a.r)) & \to & \chi(\mathsf{hd}(\mu a.\chi(H|\tilde{\mathsf{hd}}(r)))|\tilde{\mathsf{hd}}(s)) \\
(\mu) & \chi(\mathsf{hd}(\mu a.c)|k) & \to & [k/a]c \\
(\tilde{\mu}) & \chi(H|\tilde{\mathsf{hd}}(\tilde{\mu} x.c)) & \to & [H/x]c
\end{array}
$$

$$
\begin{array}{rrcl}
(\cong) & \chi(HN|k) & = & \chi(H|N :: k) \\
(\cong) & \chi(H \,\tilde{::}\, r|k) & = & \chi(H|r \tilde{\frown} k)
\end{array}
$$

We now easily write the constructors for the inference rules relative to $A - B$, just by dualizing those of implication: the already seen $\tilde{\lambda} a.r$ (left introduction), the co-continuation $H \,\tilde{::}\, r$ (right introduction), and the continuation $r \tilde{\frown} k$ (elimination, on the left!). The full system is given in Fig. 11. The typing rules are omitted due to space limitations, but writing them down is now just routine.

The classical, de Morgan/Gentzen duality is the duality between hypotheses and conclusions, l.h.s. and r.h.s. of sequents, conjunction and disjunction (if these were present), $A \supset B$ and $B - A$. Gentzen praised LK for its exhibiting of this duality [8]. Let us denote it by $(\tilde{\cdot})$, *justement*. At the level of types $A \,\tilde{\supset}\, B = \tilde{B} - \tilde{A}$ and vice-versa. Co-terms are dual of terms, and vice-versa. The same for co-continuations and continuations. The notation of constructions and the naming of reduction rules self-explains how $(\tilde{\cdot})$ operates. Commands are self-dual: $\chi(\tilde{H}|k) = \chi(\tilde{k}|\tilde{H})$. The unified system is self-dual, at the level of typing and reduction. For instance, $\Gamma \vdash t : A|\Delta$ iff $\tilde{\Delta}|\tilde{t} : \tilde{A} \vdash \tilde{\Gamma}$, etc.

Given the process of construction of the unified system, it is clear the latter has a fragment that is a sequent calculus: forbid HN and $H \,\tilde{::}\, r$ and get rid of class $H ::= x \mid \mathsf{hd}(t)$ by expanding the two cases of commands: $x\hat{}k := \chi(x|k)$ and $tk := \chi(\mathsf{hd}(t)|k)$. The result SC is not a self-dual system: its dual is the natural-deduction fragment ND of the unified system, obtained thus: forbid $u :: k$ and $r \tilde{\frown} k$ and get rid of k's by expanding the two cases of commands: $aH := \chi(H|a)$ and the new $\chi(H|\tilde{\mathsf{hd}}(r))$. In other words: de Morgan/Gentzen duality transforms sequent calculus SC into natural deduction ND, and vice-versa; and this is just a partial view of the self-duality of the unified system. Notice that the duality between SC and ND links HN with $r \tilde{\frown} k$ (and $H \,\tilde{::}\, r$ with $u :: k$), whereas the isomorphism between the two systems (internalized as equations in the unified system) links HN with $N :: k$ (and $H \,\tilde{::}\, r$ with $r \tilde{\frown} k$).

So SC is not self-dual, but $\overline{\lambda}\mu\tilde{\mu}$ seemingly is [4]. This is a symptom of something. Look again at Fig. 1. Despite its compelling symmetry, $\overline{\lambda}\mu\tilde{\mu}$ (like SC) has a huge distortion towards control. Why? Commands in $\overline{\lambda}\mu\tilde{\mu}$, we may say, have the form $\langle H|e\rangle$ with $H ::= \mathsf{hd}(t)$. Hence, the constructors for e's have no dual in the class of H's: the class of e's is fully there, the class of H's is residually there. We seem to see a duality between terms t and "co-terms" e, but here the word "co-terms" is a misnomer. "Co-terms" e's are continuations, rightly captured by the μ-operator; then, either we see terms as the "dual" of continuations, and let them be captured by the $\tilde{\mu}$-operator, but then the latter, although "dual" to the μ-operator, is not a co-control operator; or the $\tilde{\mu}$-operator, if it is to be a co-control operator,

should capture, not terms, but co-continuations, a missing kind of expression, which is also typed "on the right"; and the true co-terms are another missing kind of expressions typed "on the left". Notice that the distortion in $\overline{\lambda}\mu\tilde{\mu}$ has nothing to do with the fact that the dual of implication is not included in Fig. 1; if it were, one would add one constructor to the grammar of terms and its "dual" to the grammar of "co-terms", preserving the original "duality" [4], but still failing to achieve true duality, for the same reasons.

What did we learn? There is nothing wrong with SC or $\overline{\lambda}\mu\tilde{\mu}$, in their not being self-dual. What happens is that the classical sequent calculus, despite its symmetry, is unable to capture the duality between control and co-control, because the latter requires the full extent of the de Morgan/Gentzen duality, which also involves natural deduction, and is captured only in the unified system.

6 Conclusions

Contributions. On a higher-level, this paper has two main contributions. The first is the intended one, about the Curry-Howard isomorphism for sequent calculus. If systems of combinators correspond to Hilbert systems, and the ordinary λ-calculus corresponds to natural deduction, what variant of the λ-calculus does correspond to the sequent calculus? We propose a clear-cut answer, which turns out to be a coin with two faces: sequent calculus corresponds to a formal vector notation with first-class co-control; and to a bi-directional, agnostic, computational λ-calculus.

The second contribution concerns structural proof theory. We knew from our past experience [7, 17] that sequent calculus has to be developed hand-in-hand with natural deduction. And this was confirmed here in many ways. For instance, co-continuations correspond to a primitive syntactic class in natural deduction, from where one can import, say, the typing rules. Also, things look very different in the other side of the sequent-calculus-vs-natural-deduction mirror, different to the point of un-recognizability. For instance, co-control is almost invisible in natural deduction.

The surprise came when we considered the classical, unified, self-dual system, where we learned three things: (i) the de Morgan/Gentzen duality comprehends the duality between control and co-control, as long as we unify sequent calculus and natural deduction; (ii) sequent calculus and natural deduction are de Morgan dual, and this duality is a partial view of the self-duality of the unified system; (iii) the classical sequent calculus (the champion of symmetry) is biased towards control - because it is just a sequent calculus.

On the technical level, the main contribution is the formulation of co-control. The formulation is entirely based on the identification of the concept of co-continuation. This is the entity variables in sequent calculus proof terms stand for, and with which one may formulate the $\tilde{\mu}$-operator as a co-control operator, dualizing the behavior of the μ-operator. We showed the meaning of co-control in natural deduction, and how co-control subsumes a form of focalization. Finally, it is also noteworthy: (i) The analysis contained in the agnosticism theorem of the superimposition of CBN and CBV present in $\underline{\lambda\mathsf{let}}$ (and $\overline{\lambda\tilde{\mu}}$); (ii) The treatment of the logical operation $A - B$ contained in the unified system.

Related and future work. The author apologizes for the title of this paper, if the reader finds it exaggerated. True, the author believes something simultaneously new and very basic was said here about the computational interpretation of the sequent calculus. On the other hand, this contribution corresponds a small step from the wisdom accumulated before. Specifically, our proposal starts from the following ingredients: the $\overline{\lambda}$-calculus [10], the $\tilde{\mu}$-operator [4], the vector notation [11], the technical result about formal vector notation

[6, 3, 5], together with the previous work by the author [7, 17].

It is clear why co-control has been unnoticed in the theory and practice of programming: in a syntax with the natural deduction format, co-control control corresponds to a low-profile substitution triggering rule. Co-control, as such, is only visible in a syntax with the sequent calculus format. Now, such kind of syntax is usually regarded as a machine representation. Therefore, it is natural to ask whether co-control is relevant in the theory and practice of functional languages implementation. This question deserves further investigation.

Acknowledgements. The author was supported by Fundação para a Ciência e Tecnologia through project PEst-OE/MAT/UI0013/2014.

References

1. H. Barendregt and S. Ghilezan. Lambda terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10(1):121–134, 2000.
2. S. Cerrito and D. Kesner. Pattern matching as cut elimination. In *Proceedings of 14th annual IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 98–108, 1999.
3. I. Cervesato and F. Pfenning. A linear spine calculus. *J. Log. Compt.*, 13(5):639–688, 2003.
4. 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.
5. R. Dyckhoff and C. Urban. Strong normalisation of Herbelin's explicit substitution calculus with substitution propagation. *Journal of Logic and Computation*, 13(5):689–706, 2003.
6. 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.
7. José Espírito Santo. The λ-calculus and the unity of structural proof theory. *Theory of Computing Systems*, 45:963–994, 2009.
8. G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1969.
9. J-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. C.U.P., 1989.
10. H. Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.
11. F. Joachimski and R. Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Arch. for Math. Logic*, 42:59–87, 2003.
12. E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.
13. 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.
14. B. Pierce and D. Turner. Local type inference. In *Proc. of POPL'98*, pages 252–265. ACM, 1998.
15. D. Prawitz. *Natural Deduction*. Almquist and Wiksell, Stockholm, 1965.
16. A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, 1997.
17. José Espírito Santo. Towards a canonical classical natural deduction system. *Ann. Pure Appl. Logic*, 164(6):618–650, 2013.
18. M. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Elsevier, 2006.
19. Noam Zeilberger. On the unity of duality. *Ann. Pure Appl. Logic*, 153(1-3):66–96, 2008.