Continuation-Passing Style and Strong Normalisation for Intuitionistic Sequent Calculi

José Espírito Santo[1], Ralph Matthes[2], and Luís Pinto[1]
{jes,luis}@math.uminho.pt matthes@irit.fr

[1] Departamento de Matemática, Universidade do Minho, Portugal
[2] C.N.R.S. and University of Toulouse III, France

Abstract. The intuitionistic fragment of the call-by-name version of Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus is isolated and proved strongly normalising by means of an embedding into the simply-typed λ-calculus. Our embedding is a continuation-and-garbage-passing style translation, the inspiring idea coming from Ikeda and Nakazawa's translation of Parigot's $\lambda\mu$-calculus. The embedding simulates reductions while usual continuation-passing-style transformations erase permutative reduction steps. For our intuitionistic sequent calculus, we even only need "units of garbage" to be passed. We apply the same method to other calculi, namely successive extensions of the simply-typed λ-calculus leading to our intuitionistic system, and already for the simplest extension we consider (λ-calculus with generalised application), this yields the first proof of strong normalisation through a reduction-preserving embedding.

1 Introduction

CPS (continuation-passing style) translations are a tool with several theoretical uses. One of them is an interpretation between languages with different type systems or logical infra-structure, possibly with corresponding differences at the level of program constructors and computational behavior. Examples are when the source language (but not the target language): (i) allows permutative conversions, possibly related to connectives like disjunction [4]; (ii) is a language for classical logic, usually with control operators [9, 10, 13]; (iii) is a language for type theory [1, 2] (extending (ii) to variants of pure type systems that have dependent types and polymorphism).

This article is about CPS translations for intuitionistic sequent calculi. The source and the target languages will differ neither in the reduction strategy (they will be both call-by-name) nor at the types/logic (they will be both based on intuitionistic implicational logic); instead, they will differ in the structural format of the type system: the source is in the sequent calculus format (with cut and left introduction) whereas the target is in the natural deduction format (with elimination/application). From a strictly logical point of view, this seems a new proof-theoretical use for double-negation translations.

Additionally, we insist that our translations simulate reduction. This is a strong requirement, not present, for instance in the concept of reflection of [23].

It seems to have been intended by [1], however does not show up in the journal version [2]. But it is, nevertheless, an eminently useful requirement if one wants to infer strong normalisation of the source calculus from strong normalisation of the simply-typed λ-calculus, as we do. In order to achieve simulation, we define continuation-and-garbage passing style (CGPS) translations, following an idea due to Ikeda and Nakazawa [13]. Garbage will provide room for observing reduction where continuation-passing alone would inevitably produce an identification, leading to failure of simulation in several published proofs for variants of operationalized classical logic, noted by [20] (the problem being β-reductions under vacuous μ-abstractions). As opposed to [13], in our intuitionistic setting garbage can be reduced to "units", and garbage reduction is simply erasing a garbage unit.

The main system we translate is the intuitionistic fragment of the call-by-name restriction of the $\overline{\lambda}\mu\tilde{\mu}$-calculus [3], here named $\lambda\mathbf{J}^{\mathbf{m}se}$. The elaboration of this system is interesting on its own. We provide a CPS and a CGPS translation for $\lambda\mathbf{J}^{\mathbf{m}se}$. We also consider other intuitionistic calculi, whose treatment can be easily derived from the results for $\lambda\mathbf{J}^{\mathbf{m}se}$. Among these is included, for instance, the λ-calculus with generalised application. For all these systems a proof of strong normalisation through a reduction-preserving embedding into the simply-typed λ-calculus is provided for the first time.

The article is organized as follows: Section 2 presents $\lambda\mathbf{J}^{\mathbf{m}se}$. Sections 3 and 4 deal with the CPS and the CGPS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$, respectively. Section 5 considers other intuitionistic calculi. Section 6 compares this work with related work and concludes.

2 An intuitionistic sequent calculus

In this section, we define and identify basic properties of the calculus $\lambda\mathbf{J}^{\mathbf{m}se}$. A detailed explanation of the connection between $\lambda\mathbf{J}^{\mathbf{m}se}$ and $\overline{\lambda}\mu\tilde{\mu}$ is left to the end of this section.

There are three classes of expressions in $\lambda\mathbf{J}^{\mathbf{m}se}$:

$$\begin{array}{lll} \text{(Terms)} & t, u, v & ::= x \mid \lambda x.t \mid \{c\} \\ \text{(Co-terms)} & l & ::= [] \mid u :: l \mid (x)c \\ \text{(Commands)} & c & ::= tl \end{array}$$

An *evaluation context* E is a co-term of the form $[]$ or $u :: l$. Terms can be variables (of which we assume a denumerable set ranged over by letters x, y, w, z), lambda-abstractions $\lambda x.t$ or coercions $\{c\}$ from commands to terms[1]. A *value* is a term which is either a variable or a lambda-abstraction. We use letter V to range over values. Co-terms provide means of forming lists of arguments, generalised arguments [14], or explicit substitutions. The latter two make use of the construction $(x)c$, a new binder that binds x in c. A command tl has a

[1] A version of $\lambda\mathbf{J}^{\mathbf{m}se}$ with implicit coercions would be possible but to the detriment of the clarity, in particular, of the reduction rule ϵ below.

double role: if l is of the form $(x)c$, tl is an explicit substitution; otherwise, tl is a general form of application.

In writing expressions, sometimes we add parentheses to help their parsing. Also, we assume that the scope of binders λx and (x) extends as far as possible. Usually we write only one λ for multiple abstraction.

In what follows, we reserve letter T ("term in a large sense") for arbitrary expressions. We write $x \notin T$ if x does not occur free in T. Substitution $[t/x]T$ of a term t for all free occurrences of a variable x in T is defined as expected, where it is understood that bound variables are chosen so that no variable capture occurs. Evidently, syntactic classes are respected by substitution, i. e., $[t/x]u$ is a term, $[t/x]l$ is a co-term and $[t/x]c$ is a command.

The calculus $\lambda\mathbf{J}^{\mathbf{m}se}$ has a form of sequent for each class of expressions:

$$\Gamma \vdash t : A \qquad \Gamma | l : A \vdash B \qquad \Gamma \xrightarrow{c} B$$

Letters A, B, C are used to range over the set of types (=formulas), built from a base set of type variables (ranged over by X) using the function type (that we write $A \supset B$). In sequents, contexts Γ are viewed as finite sets of declarations $x : A$, where no variable x occurs twice. The context $\Gamma, x : A$ is obtained from Γ by adding the declaration $x : A$, and will only be written if this yields again a valid context, i. e., if x is not declared in Γ.

The typing rules of $\lambda\mathbf{J}^{\mathbf{m}se}$ can be presented as follows, stressing the parallel between left and right rules:

$$\frac{}{\Gamma | [] : A \vdash A}\ LAx \qquad \frac{}{\Gamma, x : A \vdash x : A}\ RAx$$

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

$$\frac{\Gamma, x : A \xrightarrow{c} B}{\Gamma | (x)c : A \vdash B}\ LSel \qquad \frac{\Gamma \xrightarrow{c} A}{\Gamma \vdash \{c\} : A}\ RSel$$

$$\frac{\Gamma \vdash t : A \quad \Gamma | l : A \vdash B}{\Gamma \xrightarrow{tl} B}\ Cut$$

Besides admissibility of usual weakening rules, other forms of cut are admissible as typing rules for substitution for each class of expressions.

We consider the following base reduction rules on expressions:

$$\begin{array}{lll} (\beta)\ (\lambda x.t)(u :: l) \to u((x)tl) & & (\mu)\ (x)xl \to l, \text{ if } x \notin l \\ (\pi)\quad \{tl\}E \to t\,(l@E) & & (\epsilon)\ \ \{t[]\} \to t \\ (\sigma)\quad t(x)c \to [t/x]c, & & \end{array}$$

where, in general, $l@l'$ is a co-term that represents an "eager" concatenation of l and l', viewed as lists, and is defined as follows[2]:

$$[]@l' = l' \qquad (u :: l)@l' = u :: (l@l') \qquad ((x)tl)@l' = (x)t\,(l@l')$$

The one-step reduction relation $\to$ is inductively defined as the compatible closure of the reduction rules.

The reduction rules β, π and σ are relations on commands. The reduction rule μ (resp. ϵ) is a relation on co-terms (resp. terms). Rules β and σ generate and execute an explicit substitution, respectively. Rule π appends fragmented co-terms, bringing the term t of the π-redex $\{tl\}E$ closer to root position. Also, notice here the restricted form of the outer co-term E. This restriction characterizes call-by-name reduction [3]. A μ-reduction step has necessarily one of two forms: (i) $t(x)xl \to tl$, which is the execution of a linear substitution; (ii) $u :: (x)xl \to u :: l$, which is the simplification of a generalised argument. Finally, rule ϵ erases an empty list under $\{\_\}$. Notice that empty lists are important under (x). Another view of ϵ is as a way of undoing a sequence of two coercions: the "coercion" of a term t to a command $t[]$, immediately followed by coercion to a term $\{t[]\}$. By the way, $\{c\}[] \to c$ is a π-reduction step. Most of these rules have genealogy: see Section 5.

The $\beta\pi\sigma$-normal forms are obtained by constraining commands to one of the two forms $V[]$ or $x(u :: l)$, where V, u, l are $\beta\pi\sigma$-normal values, terms and co-terms respectively. The $\beta\pi\sigma\epsilon$-normal forms are obtained by requiring additionally that, in coercions $\{c\}$, c is of the form $x(u :: l)$ (where u, l are $\beta\pi\sigma\epsilon$-normal terms and co-terms respectively). $\beta\pi\sigma\epsilon$-normal forms correspond to the multiary normal forms of [24]. If we further impose μ-normality as in [24], then co-terms of the form $(x)x(u :: l)$ obey to the additional restriction that x occurs either in u or l.

Subject reduction holds for $\to$. This fact is established with the help of the admissible rules for typing substitution and with the help of yet another admissible form of cut for typing the append operator.

We offer now a brief analysis of critical pairs in $\lambda\mathbf{J}^{\mathbf{mse}}$ [3]. There is a self-overlap of π ($\{\{tl\}E\}E'$), there are overlaps between π and any of β ($\{(\lambda x.t)(u :: l)\}E$), σ ($\{t(x)c\}E$) and ϵ (the latter in two different ways from $\{t[]\}E$ and $\{\{tl\}[]\}$). Finally, μ overlaps with σ ($t(x)xl$ for $x \notin l$). The last three critical pairs are trivial in the sense that both reducts are identical. Also the other critical pairs are joinable in the sense that both terms have a common $\to^*$-reduct. We only show this for the first case: $\{tl\}E \to t(l@E)$ by π, hence also $\{\{tl\}E\}E' \to \{t(l@E)\}E' =: L$. On the other hand, a direct application of π

[2] Concatenation is "eager" in the sense that, in the last case, the right-hand side is not $(x)\{tl\}l'$ but, in the only important case that l' is an evaluation context E, its π-reduct. One immediately verifies $l@[] = l$ and $(l@l')@l'' = l@(l'@l'')$ by induction on l. Associativity would not hold with the lazy version of @. Nevertheless, one would get that the respective left-hand side reduces in at most one π-step to the right-hand side.

[3] For higher-order rewrite systems, see the formal definition in [18].

yields $\{\{tl\}E\}E' \to \{tl\}(E@E') =: R$. Thus the critical pair consists of the terms L and R. $L \to t((l@E)@E')$ and $R \to t(l@(E@E'))$, hence L and R are joinable by associativity of @.

Since the critical pairs are joinable, the relation $\to$ is locally confluent [18]. Thus, from Corollary 1 below and Newman's Lemma, $\to$ is confluent on typable terms.

$\lambda\mathbf{J}^{\mathbf{mse}}$ as the intuitionistic fragment of CBN $\overline{\lambda}\mu\tilde{\mu}$. An appendix to this article recalls the (call-by-name) restriction of Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus [3]. The reader should have in mind the non-standard naming of reduction rules.

Let $*$ be a fixed co-variable. The intuitionistic terms, co-terms and commands are generated by the grammar.

$$\begin{array}{lll} \text{(Terms)} & t, u, v ::= & x \mid \lambda x.t \mid \mu *.c \\ \text{(Co-terms)} & e ::= & * \mid u :: e \mid \tilde{\mu} x.c \\ \text{(Commands)} & c ::= & \langle t|e\rangle \end{array}$$

Terms have no free occurrences of co-variables. Each co-term or command has exactly one free occurrence of $*$. Sequents are restricted to have exactly one formula in the RHS. Therefore, they have the particular forms $\Gamma \vdash t : A$, $\Gamma|e : A \vdash * : B$ and $c : (\Gamma \vdash * : B)$. We omit writing the intuitionistic typing rules. Reduction rules read as for $\overline{\lambda}\mu\tilde{\mu}$, except for π and $\tilde{\mu}$:

$$(\pi) \quad \langle \mu *.c|E\rangle \to [E/*]c \qquad\qquad (\tilde{\mu}) \quad \mu *.\langle t|*\rangle \to t$$

Since $* \notin t$, $[E/*]t = t$. Let us spell out $[E/*]c$ and $[E/*]e$.

$$\begin{array}{ll} [E/*]\langle t|e\rangle = \langle t|[E/*]e\rangle & [E/*](u :: e) = u :: [E/*]e \\ [E/*]* = E & [E/*](\tilde{\mu}x.c) = \tilde{\mu}x.[E/*]c \end{array}$$

If we define rule π as $\langle \mu *.\langle t|e\rangle|E\rangle \to \langle t|[E/*]e\rangle$ and $[E/*](\tilde{\mu}x.\langle t|e\rangle) = \tilde{\mu}x.\langle t|[E/*]e\rangle$ we can avoid using $[E/*]c$ altogether.

The $\lambda\mathbf{J}^{\mathbf{mse}}$-calculus is obtained from the intuitionistic fragment as a mere notational variant. The co-variable $*$ disappears from the syntax. The co-term $*$ is written $[]$. $\{c\}$ is the coercion of a command to a term, corresponding to $\mu *.c$. This coercion is what remains of the μ binder in the intuitionistic fragment. Since there is no μ, there is little sense for the notation $\tilde{\mu}$. So we write $(x)c$ instead of $\tilde{\mu}x.c$. Reduction rule $\tilde{\mu}$ now reads $\{t[]\} \to t$ and is renamed as ϵ. Sequents $\Gamma|e : A \vdash * : B$ and $c : (\Gamma \vdash * : B)$ are written $\Gamma|e : A \vdash B$ and $\Gamma \xrightarrow{c} B$. Co-terms are ranged over by l (instead of e) and thought of as generalised lists. Finally, $[E/*]l$ is written $l@E$.

3 CPS for $\lambda\mathbf{J}^{\mathbf{mse}}$

We fix a ground type (some type variable) $\perp$. Then, $\neg A := A \supset \perp$, as usual in intuitionistic logic. While our calculus is strictly intuitionistic in nature, a double-negation translation nevertheless proves useful for the purposes of establishing

strong normalisation, as has been shown by de Groote [4] for disjunction with its commuting conversions. A type A will be translated to $\overline{A} = \neg\neg A^*$, with the type A^* defined by recursion on A (where the definition of $\overline{A}$ is used as an abbreviation): $X^* = X$; $(A \supset B)^* = \neg\overline{B} \supset \neg\overline{A}$. This symmetrically-looking definition of $(A \supset B)^*$ is logically equivalent to $\overline{A} \supset \neg\neg\overline{B}$. The additional double negation of $\overline{B}$ is needed to treat cuts with co-terms ending in $(x)c$ (that are already present as generalised application in $\lambda\mathbf{J}$, see Section 5).

The translation of all syntactic elements T will be presented in Plotkin's [21] colon notation $(T : K)$ for some term K taken from simply-typed λ-calculus. A term t of $\lambda\mathbf{J}^{\mathbf{mse}}$ will then be translated into the simply-typed λ-term

$$\bar{t} = \lambda k.(t : k)$$

with a "new" variable k. The definition of $(T : K)$ uses the definition of $\bar{t}$ as an abbreviation (the variables m, w are supposed to be "fresh"):

$$\begin{array}{rcl} (x : K) = xK & \qquad & ([] : K) = \lambda w.wK \\ (\lambda x.t : K) = K(\lambda wx.w\bar{t}) & & (u :: l : K) = \lambda w.w(\lambda m.m\,(l : K)\,\overline{u}) \\ (\{c\} : K) = (c : K) & & ((x)c : K) = \lambda x.(c : K) \end{array}$$

$$\begin{array}{rcl} (t[] : K) & = & (t : K) \\ (t(u :: l) : K) & = & (t : \lambda m.m\,(l : K)\,\overline{u}) \\ (t(x)c : K) & = & (\lambda x.(c : K))\bar{t} \end{array}$$

The translation obeys to the following typing:

$$\frac{\Gamma \vdash t : A \quad \overline{\Gamma} \vdash K : \neg A^*}{\overline{\Gamma} \vdash (t : K) : \bot} \qquad \frac{\Gamma \xrightarrow{c} A \quad \overline{\Gamma} \vdash K : \neg A^*}{\overline{\Gamma} \vdash (c : K) : \bot}$$

$$\frac{\Gamma | l : A \vdash B \quad \overline{\Gamma} \vdash K : \neg B^*}{\overline{\Gamma} \vdash (l : K) : \neg\overline{A}}$$

Only the first premise in all these three rules refers to $\lambda\mathbf{J}^{\mathbf{mse}}$, the other ones to simply-typed λ-calculus. $\overline{\Gamma}$ is derived from Γ by replacing every $x : C$ in Γ by $x : \overline{C}$. As a direct consequence (to be established during the proof of the above typings), type soundness of the CPS translation follows:

$$\Gamma \vdash_{\lambda\mathbf{J}^{\mathbf{mse}}} t : A \Longrightarrow \overline{\Gamma} \vdash_{\lambda} \bar{t} : \overline{A}$$

This CPS translation is also sound for reduction, in the sense that each reduction step in $\lambda\mathbf{J}^{\mathbf{mse}}$ translates to zero or more β-steps in λ-calculus. Because of the collapsing of some reductions, this result does not guarantee yet strong normalisation of $\lambda\mathbf{J}^{\mathbf{mse}}$.

Proposition 1. *If $t \to u$ in $\lambda\mathbf{J}^{\mathbf{mse}}$, then $\bar{t} \to_\beta^* \overline{u}$ in the λ-calculus.*

Proof. Simultaneously we prove $T \to T' \Longrightarrow (T : K) \to_\beta^* (T' : K)$ for T, T' terms, co-terms or commands. More specifically, at the base cases, the CPS translation does the following: identifies ϵ and π-steps; sends one μ-step into zero or more β-steps in λ-calculus; sends one β or σ-step into one or more β-steps in λ-calculus. □

4 CGPS for $\boldsymbol{\lambda \mathbf{J}^{\mathbf{mse}}}$

This is the central mathematical finding of the present article. It is very much inspired from a "continuation and garbage passing style" translation for Parigot's $\lambda\mu$-calculus, proposed by Ikeda and Nakazawa [13]. While they use garbage to overcome the problems of earlier CPS translations that did not carry β-steps to at least one β-step if they were under a vacuous μ-binding, as reported in [20], we ensure proper simulation of ϵ, π and μ. Therefore, we can avoid the separate proof of strong normalisation of permutation steps alone that is used in addition to the CPS in [4] (there in order to treat disjunction and not for sequent calculi as we do).

We use the type $\top$ for "garbage", i. e., terms that are carried around for their operational properties, not for denotational purposes. We only require the following from $\top$: There is a term $\mathsf{s}(\cdot) : \top \to \top$ such that $\mathsf{s}(x) \to_\beta^+ x$. This can, e. g., be realised by $\top := \bot \to \bot$ and $\mathsf{s}(\cdot) := \lambda x.(\lambda y.x)(\lambda z.z)$. We abbreviate $[t; u] := (\lambda x.t)u$ for some $x \notin t$. Then, $[t; u] \to_\beta t$, and $\Gamma \vdash t : A$ and $\Gamma \vdash u : B$ together imply $\Gamma \vdash [t; u] : A$.

The only change w. r. t. the type translation in CPS is that, now,

$$\overline{A} = \top \supset \neg\neg A^*$$

is used throughout, hence, again, $X^* = X$ and $(A \supset B)^* = \neg\overline{B} \supset \neg\overline{A}$.

We define the simply-typed λ-term $(T : G, K)$ for every syntactic construct T of $\lambda\mathbf{J}^{\mathbf{mse}}$ and simply-typed λ-terms G (for "garbage") and K. Then, the translation of term t is defined to be

$$\bar{t} = \lambda gk.(t : g, k)$$

with "new" variables g, k, that is again used as an abbreviation inside the recursive definition of $(T : G, K)$ as follows (the variables m, w are again "fresh"):

$$\begin{array}{rlrl}
(x : G, K) &= x\,\mathsf{s}(G)K & ([] : G, K) &= \lambda w.w\,\mathsf{s}(G)K \\
(\lambda x.t : G, K) &= [K(\lambda wx.w\bar{t}); G] & (u :: l : G, K) &= \lambda w.w\,\mathsf{s}(G)(\lambda m.m\,(l : G, K)\,\overline{u}) \\
(\{c\} : G, K) &= (c : G, K) & ((x)c : G, K) &= \lambda x.(c : G, K)
\end{array}$$

$$\begin{aligned}
(t[] : G, K) &= (t : \mathsf{s}(G), K) \\
(t(u :: l) : G, K) &= (t : \mathsf{s}(G), \lambda m.m\,(l : G, K)\,\overline{u}) \\
(t(x)c : G, K) &= (\lambda x.(c : G, K))\bar{t}
\end{aligned}$$

If one removes the garbage argument, one precisely obtains the CPS translation.

The translation obeys to the following typing:

$$\frac{\Gamma \vdash t : A \quad \overline{\Gamma} \vdash G : \top \quad \overline{\Gamma} \vdash K : \neg A^*}{\overline{\Gamma} \vdash (t : G, K) : \bot} \qquad \frac{\Gamma | l : A \vdash B \quad \overline{\Gamma} \vdash G : \top \quad \overline{\Gamma} \vdash K : \neg B^*}{\overline{\Gamma} \vdash (l : G, K) : \neg\overline{A}}$$

$$\frac{\Gamma \xrightarrow{c} A \quad \overline{\Gamma} \vdash G : \top \quad \overline{\Gamma} \vdash K : \neg A^*}{\overline{\Gamma} \vdash (c : G, K) : \bot}$$

For $\overline{\Gamma}$ see the previous section. Therefore (and to be proven simultaneously), the CGPS translation satisfies type soundness, i. e., $\Gamma \vdash t : A$ implies $\overline{\Gamma} \vdash \bar{t} : \overline{A}$.

Lemma 1.

1. $[t/x](T : G, K) = (T : [t/x]G, [t/x]K)$ *for* T *any* u, l *or* c *such that* $x \notin T$.
2. $[\overline{t}/x](T : G, K) \to^*_\beta ([t/x]T : [\overline{t}/x]G, [\overline{t}/x]K)$ *for* T *any* u, l *or* c.
3. G *and* K *are subterms of* $(T : G, K)$ *for* T *any* u, l *or* c.
4. $(l : G, K)\overline{t} \to^*_\beta (tl : G, K)$
5. $\lambda x.(xl : G, K) \to^+_\beta (l : G, K)$ *if* $x \notin l$.
6. *(a)* $(tl : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u}) \to^+_\beta (t\,(l@(u :: l')) : G, K)$
   (b) $(l : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u}) \to^+_\beta (l@(u :: l') : G, K)$

Proof. 1./2./3. Each one by simultaneous induction on terms, co-terms and commands. 4./5. Case analysis on l. 6. By simultaneous induction on l. □

If we remove the garbage argument in statements 6 and 5 of this lemma, we can no longer guarantee one or more β-steps in λ-calculus. In the first case we have identity, whereas in the second case we have zero or more β-steps in λ-calculus. These differences account for the gain of simulation of π and μ-steps, when moving from CPS to CGPS.

Theorem 1 (Simulation). *If* $t \to u$ *in* $\lambda\mathbf{J}^{\mathbf{mse}}$, *then* $\overline{t} \to^+_\beta \overline{u}$ *in the* λ*-calculus.*

Proof. Simultaneously we prove: $T \to T' \Longrightarrow (T : G, K) \to^+_\beta (T' : G, K)$ for T, T' terms, co-terms or commands. We illustrate the cases of the base rules.

Case β: $(\lambda x.t)(u :: l) \to u(x)tl$.

$$\begin{array}{rcll}((\lambda x.t)(u :: l) : G, K) & = & (\lambda x.t : \mathsf{s}(G), \lambda m.m(l : G, K)\overline{u}) & \\ & = & [(\lambda m.m(l : G, K)\overline{u})(\lambda w x.w\overline{t}); \mathsf{s}(G)] & \\ & \to^3_\beta & (\lambda x.(l : G, K)\overline{t})\overline{u} & \\ & \to^*_\beta & (\lambda x.(tl : G, K))\overline{u} & \text{(Lemma 1.4)} \\ & = & (u(x)tl : G, K) & \end{array}$$

Case π: $\{tl\}E \to t\,(l@E)$. Sub-case $E = []$.

$$\begin{array}{rcll}(\{tl\}[] : G, K) = (tl : \mathsf{s}(G), K) & \to^+_\beta & (tl : G, K) & \text{(Lemma 1.3/1.1)} \\ & = & (t\,(l@[]) : G, K). & \end{array}$$

Sub-case $E = u :: l'$.

$$\begin{array}{rcll}(\{tl\}(u :: l') : G, K) & = & (tl : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u}) & \\ & \to^+_\beta & (t\,(l@(u :: l')) : G, K) & \text{(Lemma 1.6)}\end{array}$$

Case σ: $t(x)c \to [t/x]c$.

$$\begin{array}{rcll}(t(x)c : G, K) & = & (\lambda x.(c : G, K))\overline{t} & \\ & \to_\beta & [\overline{t}/x](c : G, K) & \\ & \to^*_\beta & ([t/x]c : G, K) & \text{(Lemma 1.2)}\end{array}$$

Case μ: $(x)xl \to l$, if $x \notin l$.

$$((x)xl : G, K) = \lambda x.(xl : G, K) \to^+_\beta (l : G, K) \quad \text{(Lemma 1.5)}$$

Case ϵ: $\{t[]\} \to t$.

$$(\{t[]\} : G, K) = (t[] : G, K) = (t : \mathsf{s}(G), K) \to_\beta^+ (t : G, K)$$

The cases corresponding to the closure rule $t \to t' \Longrightarrow tl \to t'l$ (resp. $l \to l' \Longrightarrow tl \to tl'$) can be proved by case analysis on l (resp. $l \to l'$). The cases corresponding to the other closure rules follow by routine induction. □

Remark 1. Unlike the failed simulation by CPS reported in [20] that only occurred with the closure rules, the need for garbage in our translation is already clearly visible in the subcase $E = []$ for π and the case ϵ. But the garbage is also effective for the closure rules, where the most delicate rule is the translation of $t(u :: l)$ that mentions l and u only in the continuation argument K to t's translation. Lemma 1.3 is responsible for propagation of simulation. The structure of our garbage – essentially just "units of garbage" – can thus be easier than in the CGPS in [13] for $\lambda\mu$-calculus since there, K cannot be guaranteed to be a subterm of $(T : G, K)$, again because of the problem with void μ-abstractions. The solution of [13] for the most delicate case of application is to copy the K argument into the garbage. We do not need this in our intuitionistic calculi.

Corollary 1. *The typable terms of* $\lambda\mathbf{J}^{\mathbf{m}se}$ *are strongly normalising.*

Recalling our discussion in Section 2, we already could have inferred strong normalisation of $\lambda\mathbf{J}^{\mathbf{m}se}$ from that of $\overline{\lambda}\mu\tilde{\mu}$, which has been shown directly by Polonovski [22] using reducibility candidates and before by Lengrand's [16] embedding into a calculus by Urban that also has been proven strongly normalizing by the candidate method. Our proof is just by a syntactic transformation to simply-typed λ-calculus.

5 CGPS for other intuitionistic calculi

As a consequence of the results of the previous section, we obtain in this section the embedding, by a CGPS translation, of several intuitionistic calculi into the simply-typed λ-calculus. These intuitionistic calculi are successive extensions of the simply-typed λ-calculus that lead to $\lambda\mathbf{J}^{\mathbf{m}se}$, as illustrated in the diagram below, and include both natural deduction systems and other sequent calculi.

$$\lambda\mathbf{J}^{\mathbf{m}se} \xleftarrow{\ e\ } \lambda\mathbf{J}^{\mathbf{m}s} \xleftarrow{\ s\ } \lambda\mathbf{J}^{\mathbf{m}} \xleftarrow{\ m\ } \lambda\mathbf{J} \xleftarrow{\ J\ } \lambda$$

Each extension step adds both a new feature and a reduction rule to the preceding calculus. The following table summarizes these extensions.

calculus	reduction rules	feature added
λ	β	
$\lambda\mathbf{J}$	β, π	generalised application
$\lambda\mathbf{J}^{\mathbf{m}}$	β, π, μ	multiarity
$\lambda\mathbf{J}^{\mathbf{m}s}$	β, π, μ, σ	explicit substitution
$\lambda\mathbf{J}^{\mathbf{m}se}$	$\beta, \pi, \mu, \sigma, \epsilon$	empty lists of arguments

The scheme for naming systems and reduction rules intends to be systematic (and in particular explains the name $\lambda\mathbf{J}^{\mathbf{m}se}$).

The path between the two end-points of this spectrum visits and organizes systems known from the literature. $\lambda\mathbf{J}$ is a variant of the calculus ΛJ of [14]. $\lambda\mathbf{J}^{\mathbf{m}}$ is a variant of the system in [8]. $\lambda\mathbf{J}^{\mathbf{m}se}$ is studied in [7] under the name λ^{Gtz}. This path is by no means unique. Other intermediate systems could have been visited (like the multiary λ-calculus $\lambda^{\mathbf{m}}$, named λPh in [8]), had the route been a different one, i. e., had the different new features been added in a different order.

Each system $\mathcal{L} \in \{\lambda\mathbf{J}, \lambda\mathbf{J}^{\mathbf{m}}, \lambda\mathbf{J}^{\mathbf{m}s}\}$ embeds in the system immediately after it in this spectrum, in the sense of existing a mapping simulating reduction. Hence, strong normalisation is inherited from $\lambda\mathbf{J}^{\mathbf{m}se}$ all the way down to $\lambda\mathbf{J}$. Also, each $\mathcal{L} \in \{\lambda\mathbf{J}, \lambda\mathbf{J}^{\mathbf{m}}, \lambda\mathbf{J}^{\mathbf{m}s}\}$ has, by composition, an embedding $g_{\mathcal{L}}$ in $\lambda\mathbf{J}^{\mathbf{m}se}$. It can be shown that there is a CGPS translation of each $\mathcal{L}$ so that this CGPS translation is the composition of $g_{\mathcal{L}}$ with the CGPS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$. It follows that the CGPS translation of each $\mathcal{L}$ simulates reduction, that is, is an embedding of $\mathcal{L}$ in the λ-calculus. Let us see all this with some detail.

$\lambda\mathbf{J}$-calculus. The terms of $\lambda\mathbf{J}$ are generated by the grammar:

$$t, u, v ::= x \mid \lambda x.t \mid t(u, x.v)$$

Construction $t(u, x.v)$ is called generalised application. Following [14], $(u, x.v)$ is called a generalised argument; they will be denoted by the letters R and S. Typing rules for x and $\lambda x.t$ are as usual and omitted. The typing rule for generalised application is:

$$\frac{\Gamma \vdash t : A \supset B \quad \Gamma \vdash u : A \quad \Gamma, x : B \vdash v : C}{\Gamma \vdash t(u, x.v) : C}\ GApp$$

Reduction rules are as in [14], except that π is defined in the "eager" way:

$$(\beta)\ (\lambda x.t)(u, y.v) \to [[u/x]t/y]v \qquad (\pi)\ tRS \to t(R@S)$$

where the generalised argument $R@S$ is defined by recursion on R:

$$(u, x.V)@S = (u, x.VS) \qquad (u, x.tR')@S = (u, x.t(R'@S))\ ,$$

for V a value, i. e., a variable or a λ-abstraction. The operation @ is associative, which allows to join the critical pair of π with itself as before for $\lambda\mathbf{J}^{\mathbf{m}se}$. The other critical pair stems from the interaction of β and π and is joinable as well.

Strong normalisation of typable terms immediately follows from that of ΛJ in [15], but in the present article, we even get an embedding into λ.

In defining the embeddings m, s and e we omit the clauses for variables and λ-abstraction, because in these cases the embeddings are defined homomorphically. Although we won't use it, we recall the embedding $J : \lambda \to \lambda\mathbf{J}$ just for completeness: $J(tu) = J(t)(J(u), x.x)$.

$\lambda\mathbf{J}^{\mathbf{m}}$-calculus. We offer now a new, lighter, presentation of the system in [8]. The expressions of $\lambda\mathbf{J}^{\mathbf{m}}$ are given by the grammar:

$$\text{(Terms)}\ \ t,u,v ::= x \mid \lambda x.t \mid t(u,l) \qquad \text{(Co-terms)}\ \ l ::= u :: l \mid (x)v$$

The application $t(u,l)$ is both generalised and multiary. Multiarity is the ability of forming a chain of arguments, as in $t(u_1, u_2 :: u_3 :: (x)v)$. By the way, this term is written $t(u_1, u_2 :: u_3 :: [], (x)v)$ in the syntax of [8]. There are two kinds of sequents: $\Gamma \vdash t : A$ and $\Gamma | l : A \vdash B$. Typing rules are as follows:

$$\frac{\Gamma \vdash t : A \supset B \quad \Gamma \vdash u : A \quad \Gamma | l : B \vdash C}{\Gamma \vdash t(u,l) : C}\ GMApp$$

$$\frac{\Gamma, x : A \vdash v : B}{\Gamma | (x)v : A \vdash B}\ Sel \qquad \frac{\Gamma \vdash u : A \quad \Gamma | l : B \vdash C}{\Gamma | u :: l : A \supset B \vdash C}\ LIntro$$

We re-define reduction rules of [8] in this new syntax. Rule μ can now be defined as a relation on co-terms. Rule π is changed to the "eager" version, using letters R and S for generalised arguments, i. e., elements of the form (u,l).

$$\begin{array}{ll} (\beta_1)\ (\lambda x.t)(u,(y)v) \to [[u/x]t/y]v & (\pi) \qquad tRS \to t(R@S) \\ (\beta_2)\ (\lambda x.t)(u, v :: l) \to ([u/x]t)(v,l) & (\mu)\ (x)x(u,l) \to u :: l,\ \text{if } x \notin u,l \end{array}$$

$\beta = \beta_1 \cup \beta_2$. The generalised argument $R@S$ is defined with the auxiliary notion of the co-term $l@S$ that is defined by recursion on l by $(u :: l)@S = u :: (l@S)$, $((x)V)@S = (x)VS$, for V a value, and $((x)t(u,l))@S = (x)t(u, l@S)$. Then, define $R@S$ by $(u,l)@S = (u, l@S)$. Since the auxiliary operation @ can be proven associative, this also holds for the operation @ on generalised arguments. Apart from the usual self-overlapping of π that is joinable by associativity of @, there are critical pairs between β_i and π that are joinable. The last critical pair is between β_1 and μ and needs a β_2-step to be joined.

The embedding $m : \lambda\mathbf{J} \to \lambda\mathbf{J^m}$ is given by $m(t(u, x.v)) = m(t)(m(u), (x)m(v))$.

$\lambda\mathbf{J^{ms}}$-calculus. The expressions of $\lambda\mathbf{J^{ms}}$ are given by:

$$\text{(Terms)}\ \ t,u,v ::= x \mid \lambda x.t \mid tl \qquad \text{(Co-terms)}\ \ l ::= u :: l \mid (x)v$$

The construction tl has a double role: either it is a generalised and multiary application $t(u :: l)$ or it is an explicit substitution $t(x)v$. The typing rules for $u :: l$ and $(x)v$ are as in $\lambda\mathbf{J^m}$. Construction tl is typed by:

$$\frac{\Gamma \vdash t : A \quad \Gamma | l : A \vdash B}{\Gamma \vdash tl : B}\ Cut$$

The reduction rules are as follows:

$$\begin{array}{ll} (\beta)\ (\lambda x.t)(u :: l) \to u((x)tl) & (\sigma)\ t(x)v \to [t/x]v \\ (\pi) \quad (tl)(u :: l') \to t\,(l@(u :: l')) & (\mu)\ (x)xl \to l,\ \text{if } x \notin l \end{array}$$

where the co-term $l@l'$ is defined by $(u :: l)@l' = u :: (l@l')$, $((x)V)@l' = (x)Vl'$, for V a value, and $((x)tl)@l' = (x)t\,(l@l')$. Again, @ is associative and guarantees

the joinability of the critical pair of π with itself. The critical pairs between β and π and between σ and μ are joinable as for $\lambda\mathbf{J}^{\mathbf{mse}}$. The overlap between σ and π is bigger than in $\lambda\mathbf{J}^{\mathbf{mse}}$ since the divergence arises for $t((x)v)(u :: l)$ with v an arbitrary term whereas in $\lambda\mathbf{J}^{\mathbf{mse}}$, there is only a command at that place. Joinability is nevertheless easily established.

Comparing these reduction rules with those of $\lambda\mathbf{J}^{\mathbf{m}}$, there is only one β-rule, whose effect is to generate a substitution. There is a separate rule σ for substitution execution. The embedding $s : \lambda\mathbf{J}^{\mathbf{m}} \to \lambda\mathbf{J}^{\mathbf{ms}}$ is characterized by $s(t(u, l)) = s(t)(s(u) :: s(l))$.

Finally, let us compare $\lambda\mathbf{J}^{\mathbf{ms}}$ and $\lambda\mathbf{J}^{\mathbf{mse}}$. In the former, any term can be in the scope of a selection (x), whereas in the latter the scope of a selection is a command. But in the latter we have a new form of co-term $[]$. Since in $\lambda\mathbf{J}^{\mathbf{mse}}$ we can coerce any term t to a command $t[]$, we can translate $\lambda\mathbf{J}^{\mathbf{ms}}$ into $\lambda\mathbf{J}^{\mathbf{mse}}$, by defining $e((x)t) = (x)e(t)[]$. In fact, one has to refine this idea in order to get simulation of reduction. The embedding $e : \lambda\mathbf{J}^{\mathbf{ms}} \to \lambda\mathbf{J}^{\mathbf{mse}}$ obeys the following:

$$e(tl) = \{e(t)e(l)\} \qquad e((x)V) = (x)e(V)[] \qquad e((x)tl) = (x)e(t)e(l)$$

Proposition 2. *Each of the embeddings m, s and e simulates reduction.*

Proof. We just sketch the proof for e (the others are easier). We prove

$$t \to t' \Longrightarrow e(t) \to^+ e(t') \,\text{and}\, e((x)t) \to^+ e((x)t'), \;\; \text{for any}\, t, t' \in \lambda\mathbf{J}^{\mathbf{ms}}$$

simultaneously with $l \to l' \Longrightarrow e(l) \to^+ e(l')$. In particular, the following fact is used: $[e(t)/x]e(T) \to_\epsilon^* e([t/x]T)$, for T a term or a co-term. □

Since each of m, s and e preserves typability, it follows from Corollary 1 that:

Corollary 2. *The typable terms of $\lambda\mathbf{J}^{\mathbf{ms}}$, $\lambda\mathbf{J}^{\mathbf{m}}$ and $\lambda\mathbf{J}$ are strongly normalising.*

CGPS translations. We define CGPS translations for $\lambda\mathbf{J}^{\mathbf{ms}}$, $\lambda\mathbf{J}^{\mathbf{m}}$ and $\lambda\mathbf{J}$. The translation of types is unchanged. In each translation, we just show the clauses that are new.

1. For $\lambda\mathbf{J}^{\mathbf{ms}}$ one has (the first rule just replaces c by vl in the rule for $\lambda\mathbf{J}^{\mathbf{mse}}$):
$$\begin{aligned}(t(x)vl : G, K) &= (\lambda x.(vl : G, K))\overline{t}\\ (t(x)V : G, K) &= (\lambda x.(V : \mathsf{s}(G), K))\overline{t}\\ ((x)v : G, K) &= \lambda x.(v : \mathsf{s}(G), K)\end{aligned}$$
2. For $\lambda\mathbf{J}^{\mathbf{m}}$: $(t(u, l) : G, K) = (t : \mathsf{s}(G), \lambda m.m\,(l : G, K)\,\overline{u})$.
3. Finally, for $\lambda\mathbf{J}$: $(t(u, x.v) : G, K) = (t : \mathsf{s}(G), \lambda m.m\,(\lambda x.(v : \mathsf{s}(G), K))\,\overline{u})$.

These translations are coherent with the CGPS translation for $\lambda\mathbf{J}^{\mathbf{mse}}$:

Proposition 3. *Let $\mathcal{L} \in \{\lambda\mathbf{J}^{\mathbf{ms}}, \lambda\mathbf{J}^{\mathbf{m}}, \lambda\mathbf{J}\}$. Let $f_{\mathcal{L}}$ be the embedding of $\mathcal{L}$ in its immediate extension and let $g_{\mathcal{L}}$ be the embedding of $\mathcal{L}$ in $\lambda\mathbf{J}^{\mathbf{mse}}$. Then, for all $t \in \mathcal{L}$, $\overline{t} = \overline{f_{\mathcal{L}}(t)}$. Hence, for all $t \in \mathcal{L}$, $\overline{t} = \overline{g_{\mathcal{L}}(t)}$.*

Theorem 2 (Simulation). *Let $\mathcal{L} \in \{\lambda\mathbf{J}^{\mathbf{ms}}, \lambda\mathbf{J}^{\mathbf{m}}, \lambda\mathbf{J}\}$. If $t \to u$ in $\mathcal{L}$, then $\overline{t} \to_\beta^+ \overline{u}$ in the λ-calculus.*

Proof. By Propositions 2 and 3 and Theorem 1. □

6 Further remarks

This article provides reduction-preserving CGPS translations of $\lambda\mathbf{J}^{\mathbf{m}se}$ and other intuitionistic calculi, hence obtaining embeddings into the simply-typed λ-calculus and proving strong normalisation. As a by-product, the connections between systems like $\lambda\mathbf{J}$ and $\lambda\mathbf{J}^{\mathbf{m}}$ and the intuitionistic fragment of $\overline{\lambda}\mu\tilde{\mu}$ are detailed.

In the literature one finds strong normalisation proofs for sequent calculi [5, 6, 16, 17, 22, 25], but not by means of CPS translations; or CPS translations for natural deduction systems [1, 2, 4, 13, 19].

This article provides, in particular, a reduction-preserving CGPS translation for the lambda-calculus with generalised applications $\lambda\mathbf{J}$. [19] covers full propositional classical logic with general elimination rules and its intuitionistic implicational fragment corresponds to $\lambda\mathbf{J}$. However, [19] does not prove a simulation by CPS in our sense (permutative conversions are collapsed), so an auxiliary argument in the style of de Groote [4], involving a proof in isolation of SN for permutative conversions, is used.

In Curien and Herbelin's work [3, 11] one finds a CPS translation $(\_)^n$ of the call-by-name restriction of $\overline{\lambda}\mu\tilde{\mu}$. We compare $(\_)^n$ with our $\overline{(\_)}$. (i) $(\_)^n$ generalises Hofmann-Streicher translation [12]; $\overline{(\_)}$ generalises Plotkin's call-by-name CPS translation [21]. (ii) $(\_)^n$ does not employ the colon operator; $\overline{(\_)}$ does employ (we suspect that doing administrative reductions at compile time is necessary to achieve simulation of reduction); (iii) $(\_)^n$ is defined for expressions where every occurrence of $u :: l$ is of the particular form $u :: E$; no such restriction is imposed in the definition of $\overline{(\_)}$. (iv) at some points it is unclear what the properties of $(\_)^n$ are, but no proof of strong normalisation is claimed; the CGPS $\overline{(\_)}$ simulates reduction and thus achieves a proof of strong normalisation.

The results obtained extend to second-order calculi (this extension is omitted for space reasons). We plan to extend the technique of continuation-and-garbage passing to $\overline{\lambda}\mu\tilde{\mu}$ and to dependently-typed systems. We tried to extend the CGPS to CBN $\overline{\lambda}\mu\tilde{\mu}$, described in the appendix, but already for a CPS translation, we do not see how to profit from the continuation argument for the translation of co-terms and commands. Moreover, a special case of the rule we call π corresponds to the renaming rule $a(\mu b.M) \to [a/b]M$ of $\lambda\mu$-calculus. This rule is evidently not respected by the CGPS translation by Ikeda and Nakazawa [13] (nor by the CPS they recall) since the continuation argument K is omitted in the interpretation of the left-hand side but not in the right-hand side. So, new ideas or new restrictions will be needed.

Acknowledgements: We thank the referees for pointing out the work of Nakazawa and Tatsuta, whom we thank for an advanced copy of [19]. The first and third authors are supported by FCT through the Centro de Matemática da Universidade do Minho. The second author thanks for an invitation by that institution to Braga in October 2006. All authors are also supported by the European project TYPES.

References

1. G. Barthe, J. Hatcliff, and M. Sørensen. A notion of classical pure type system (preliminary version). In S. Brookes and M. Mislove, editors, *Proc. of the 30th Conf. on the Mathematical Foundations of Programming Semantics*, volume 6 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1997. 56 pp.
2. G. Barthe, J. Hatcliff, and M. Sørensen. Cps translations and applications: The cube and beyond. *Higher-Order and Symbolic Computation*, 12(2):125–170, 1999.
3. P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of 5th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP '00), Montréal*, pages 233–243. IEEE, 2000.
4. P. de Groote. On the strong normalisation of intuitionistic natural deduction with permutation-conversions. *Information and Computation*, 178:441–464, 2002.
5. A. Dragalin. *Mathematical Intuitionism.*, volume 67 of *Translations of Mathematical Monographs*. AMS, 1988.
6. 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.
7. J. Espírito Santo. Completing Herbelin's programme (in this volume).
8. J. Espírito Santo and L. Pinto. Permutative conversions in intuitionistic multiary sequent calculus with cuts. In M. Hofmann, editor, *Proc. of TLCA'03*, volume 2701 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, 2003.
9. M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. Reasoning with continuations. In *1st Symposium on Logic and Computer Science*, pages 131–141. IEEE, 1986.
10. T. Griffin. A formulae-as-types notion of control. In *ACM Conf. Principles of Programming Languages*, pages 47–58. ACM Press, 1990.
11. H. Herbelin. C'est maintenant qu'on calcule, 2005. Habilitation Thesis, Paris XI.
12. M. Hofmann and T. Streicher. Continuation models are universal for lambda-mu-calculus. In *Proc. of LICS 1997*, pages 387–395. IEEE, 1997.
13. S. Ikeda and K. Nakazawa. Strong normalization proofs by CPS-translations. *Information Processing Letters*, 99:163–170, 2006.
14. F. Joachimski and R. Matthes. Standardization and confluence for a lambda-calculus with generalized applications. In L. Bachmair, editor, *Proc. of Int. Conf. on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, 2000.
15. F. Joachimski and R. Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 42(1):59–87, 2003.
16. S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In B. Gramlich and S. Lucas, editors, *Post-proc. of the 3rd Workshop on Reduction Strategies in Rewriting and Programming (WRS'03)*, volume 86 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.
17. S. Lengrand, R. Dyckhoff, and J. McKinna. A sequent calculus for type theory. In Z. Ésik, editor, *Computer Science Logic, 20th Int. Workshop, CSL 2006, Proc.*, volume 4207 of *Lecture Notes in Computer Science*, pages 441–455. Springer Verlag, 2006.
18. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

19. K. Nakazawa and M. Tatsuta. Strong normalization of classical natural deduction with disjunctions. Submitted.
20. K. Nakazawa and M. Tatsuta. Strong normalization proof with CPS-translation for second order classical natural deduction. *Journal of Symbolic Logic*, 68(3):851–859, 2003. Corrigendum: vol. 68 (2003), no. 4, pp. 1415–1416.
21. G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.
22. E. Polonovski. Strong normalization of lambda-mu-mu-tilde with explicit substitutions. In I. Walukiewicz, editor, *Proc. of 7th Int. Conf. on Foundations of Software Sciences and Computation Structures (FoSSaCS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 423–437. Springer-Verlag, 2004.
23. A. Sabry and P. Wadler. A reflection on call-by-value. In *Proc. of ACM SIGPLAN Int. Conf. on Functional Programming ICFP 1996*, pages 13–24. ACM Press, 1996.
24. H. Schwichtenberg. Termination of permutative conversions in intuitionistic Gentzen calculi. *Theoretical Computer Science*, 212(1–2):247–260, 1999.
25. C. Urban and G. Bierman. Strong normalisation of cut-elimination in classical logic. In J.-Y. Girard, editor, *Proc. of TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*, pages 365–380. Springer-Verlag, 1999.

A The call-by-name $\overline{\lambda}\mu\tilde{\mu}$-calculus

In this appendix we recall the call-by-name restriction of $\overline{\lambda}\mu\tilde{\mu}$-calculus [3] (with the subtraction connective left out). Expressions are either terms, co-terms or commands and are defined by:

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

Variables (resp. co-variables) are ranged over by x, y, z (resp. a, b, c). An *evaluation context* E is a co-term of the form a or $u :: e$.

There is one kind of sequent per each syntactic class

$$\Gamma \vdash t : A | \Delta \qquad \Gamma | e : A \vdash \Delta \qquad c : (\Gamma \vdash \Delta)$$

Typing rules are as in [3]. We consider 5 reduction rules:

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

These are the reductions considered by Polonovski in [22], with three provisos. First, the β-rule for the subtraction connective is not included. Second, in the π-rule, the co-term involved is an evaluation context E; this is exactly what characterizes the call-by-name restriction of $\overline{\lambda}\mu\tilde{\mu}$ [3]. Third, the naming of the rules is non-standard. Curien and Herbelin (and Polonovski as well) name rules π and σ as μ, $\tilde{\mu}$, respectively. The name μ has moved to the rule called *se* in [22]. By symmetry, the rule called *sv* by Polonovski is now called $\tilde{\mu}$. The reason for this change is explained by the spectrum of systems in Section 5: the rule we now call π (resp. μ) is the most general form of the rule with the same name in the system $\lambda\mathbf{J}$ (resp. $\lambda\mathbf{J^m}$), and therefore its name goes back to [14] (resp. [8], actually back to [24]).