CONTINUATION-PASSING STYLE AND STRONG NORMALISATION FOR INTUITIONISTIC SEQUENT CALCULI [*]

JOSÉ ESPÍRITO SANTO [a], RALPH MATTHES [b], AND LUÍS PINTO [c]

[a,c] Departamento de Matemática, Universidade do Minho, Portugal
  e-mail address: {jes,luis}@math.uminho.pt

[b] Institut de Recherche en Informatique de Toulouse (IRIT), C.N.R.S. and University of Toulouse III (Paul Sabatier), 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 strictly 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. The results obtained extend to second and higher-order calculi.

CONTENTS

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 [6]; (ii) is a language for classical logic, usually with control operators [13, 16, 20]; (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 strictly simulate reduction. This is a strong requirement, not present, for instance in the concept of reflection of [34]. 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 strict simulation, we define continuation-and-garbage passing style (CGPS) translations, following an idea due to Ikeda and Nakazawa [20]. Garbage will provide room for observing reduction where continuation-passing alone would inevitably produce an identification, leading to failure of strict simulation in several published proofs for variants of operationalized classical logic, noted by [29] (the problem being β-reductions under vacuous μ-abstractions). As opposed to [20], 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 [5], 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 an extended version of the conference contribution of the same authors [12]. It is organized as follows: Section 2 presents $\lambda\mathbf{J}^{\mathbf{m}se}$. Section 3 compares $\lambda\mathbf{J}^{\mathbf{m}se}$ with

other systems, and obtains as a by-product confluence of $\lambda\mathbf{J}^{\mathbf{m}se}$. Sections 4 deals with the C(G)PS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$ and its subsystems. Section 5 extends the results to systems F, F^ω and intuitionistic higher-order logic. 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 next section.

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

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

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 [21], or explicit substitutions. A co-term of the form $(x)c$ binds variable x in c and provides the generalised application facility. Operationally it can be thought of as "substitute for x in c". A co-term of the form $[\,]$ or $u :: l$ is called an *evaluation context* and is denoted by E. An evaluation context of the form $u :: l$ allows for multiary applications, and when passed to a term it indicates that after consumption of argument u computation should carry on with arguments in l. $[\,]$ marks the end of an evaluation context and compensates the impossibility of writing $(x)x$.

A command tl has a 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.

$$
\begin{array}{llclllcl}
[t/x]x & = & t & & [t/x][\,] & = & [\,] \\
[t/x]y & = & y \text{ if } x \neq y & & [t/x](u :: l) & = & [t/x]u :: [t/x]l \\
[t/x](\lambda y.u) & = & \lambda y.[t/x]u & & [t/x]((y)c) & = & (y)[t/x]c \\
[t/x]\{c\} & = & \{[t/x]c\} & & [t/x](ul) & = & [t/x]u[t/x]l
\end{array}
$$

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:

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

---

[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.

Figure 1: Typing rules of $\lambda\mathbf{J}^{\mathbf{m}se}$

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

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

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

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

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. Similarly, Γ, Δ is the union of Γ and Δ, and assumes that the sets of variables declared in Γ and Δ are disjoint. We can think of a term (resp. co-term) as an annotation for a selected formula in the *rhs* (resp. *lhs*). Commands annotate sequents generated as a result of logical cuts, where there is no selected formula on the *rhs* or *lhs*; as such we write them on top of the sequent arrow.

The typing rules of $\lambda\mathbf{J}^{\mathbf{m}se}$ are presented in Figure 1, stressing the parallel between left and right rules.

The following other forms of cut are admissible as typing rules for substitution for each class of expressions:

$$\frac{\Gamma\vdash t:A \quad \Gamma,x:A\vdash u:B}{\Gamma\vdash [t/x]u:B} \qquad \frac{\Gamma\vdash t:A \quad \Gamma,x:A|l:B\vdash C}{\Gamma|[t/x]l:B\vdash C}$$

$$\frac{\Gamma\vdash t:A \quad \Gamma,x:A\xrightarrow{c}B}{\Gamma\xrightarrow{[t/x]c}B}$$

We also have the usual weakening rules: If a sequent with context Γ is derivable and Γ is replaced by a context Γ' that is a superset of Γ, then also this sequent is derivable.

We consider the following base reduction rules on expressions:

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

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 term closure of the reduction rules, by adding the following closure rules to the above initial cases of $\to$:

$$
\begin{aligned}
t \to t' &\implies \lambda x.t \to \lambda x.t', \quad tl \to t'l, \quad t :: l \to t' :: l, \\
l \to l' &\implies u :: l \to u :: l', \quad tl \to tl', \\
c \to c' &\implies (x)c \to (x)c', \quad \{c\} \to \{c'\}.
\end{aligned}
$$

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 [5]. A μ-reduction step that is not at the root 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. Rule μ undoes the sequence of inference steps consisting in un-selecting a formula and giving it the name x, followed by immediate selection of the same formula. The proviso $x \notin l$ guarantees that no contraction was involved. 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 3.2.

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 [35]. If we further impose μ-normality as in [35], 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 $\to$, i.e., the following rules are admissible:

$$
\frac{\Gamma \vdash t : A \quad t \to t'}{\Gamma \vdash t' : A} \qquad
\frac{\Gamma | l : A \vdash B \quad l \to l'}{\Gamma | l' : A \vdash B} \qquad
\frac{\Gamma \xrightarrow{c} B \quad c \to c'}{\Gamma \xrightarrow{c'} B}
$$

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:

$$
\frac{\Gamma | l : A \vdash B \quad \Gamma | l' : B \vdash C}{\Gamma | l@l' : A \vdash C}
$$

We offer now a brief analysis of critical pairs in $\lambda \mathbf{J}^{\mathbf{m}se}$ [3].

---

[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 [27].

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 in two different ways from $t(x)xl$ for $x \notin l$ and $(x)(x(y)c)$ for $x \notin c$. The last four 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 $\rightarrow^*$-reduct. We only show this for the first case: $\{tl\}E' \rightarrow t(l@E')$ by π, hence also

$$\{\{tl\}E'\}E \rightarrow \{t(l@E')\}E =: L.$$

On the other hand, a direct application of π yields

$$\{\{tl\}E'\}E \rightarrow \{tl\}(E'@E) =: R.$$

Thus the critical pair consists of the terms L and R. $L \rightarrow t((l@E')@E)$ and $R \rightarrow t(l@(E'@E))$, hence L and R are joinable by associativity of $@$.

We remark that the first three critical pairs (like the one just shown) are of a particularly simple nature: The forking term is of the form $\{c\}E$ with c any of the command redexes, i. e., a left-hand side of β, π or σ. The L term is obtained by reducing c to the respective right-hand side c' of that rule, and the R term comes from applying π at the root. Since c' is again a command, $L = \{c'\}E$ can be reduced by π to a term L'. The decisive feature of $@$ is that $R \rightarrow L'$ by an instance of the rule $c \rightarrow c'$ where the co-term part l of $c = tl$ is replaced by $l@E$.

Since the critical pairs are joinable, the relation $\rightarrow$ is locally confluent [27]. Thus, from Corollary 4.5 below and Newman's Lemma, $\rightarrow$ is confluent on typable terms. Confluence on all terms is proved in the next section.

3. Comparison with other systems

In this section we show that $\lambda\mathbf{J}^{\mathbf{m}se}$ can be generated "from above" - being the intuitionistic fragment of the call-by-name restriction of Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus; and "from below" - being the end-point of a spectrum of successively more general intuitionistic systems, starting from the ordinary λ-calculus. This latter result, by showing that the systems in the spectrum are subsystems of $\lambda\mathbf{J}^{\mathbf{m}se}$, will allow us to adapt easily the result about $\lambda\mathbf{J}^{\mathbf{m}se}$ to (new) results about its subsystems. In addition, we will obtain, as a by-product, a proof of confluence for $\lambda\mathbf{J}^{\mathbf{m}se}$ even for the untypable terms.

3.1. $\lambda\mathbf{J}^{\mathbf{m}se}$ as the intuitionistic fragment of CBN $\overline{\lambda}\mu\tilde{\mu}$.
After a recapitulation of a call-by-name version of $\overline{\lambda}\mu\tilde{\mu}$-calculus, we restrict it to the intuitionistic case and rediscover $\lambda\mathbf{J}^{\mathbf{m}se}$.

3.1.1. *The call-by-name $\overline{\lambda}\mu\tilde{\mu}$-calculus.*
Here, we recall the Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus [5]. More precisely, we only consider implication (i. e., we do not include the subtraction connective) and we present the call-by-name restriction of the system.

Expressions are either terms, co-terms or commands and are defined by the following grammar:

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

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 $u :: e$.

Figure 2: Typing rules of CBN $\overline{\lambda}\mu\tilde{\mu}$

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

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

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

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

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 given in Figure 2.

There are 6 substitution operations altogether:

$$[t/x]c \qquad [t/x]u \qquad [t/x]e \qquad [e/a]c \qquad [e/a]u \qquad [e/a]e'$$

We consider 5 reduction rules:

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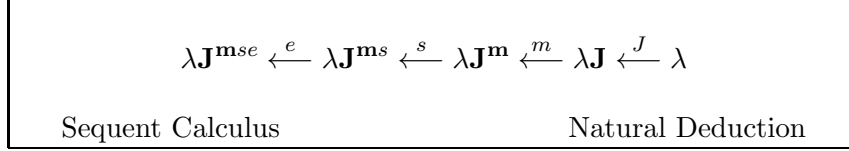\end{array}
$$

These are the reductions considered by Polonovski in [33], 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 $\overline{\lambda}\mu\tilde{\mu}$ [5]. 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 [33]. By symmetry, the rule called *sv* by Polonovski is now called $\tilde{\mu}$. The reason for this change is explained below by the spectrum of systems in Section 3.2: 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 [21] (resp. [10], actually back to [35]).

3.1.2. *The intuitionistic fragment of CBN $\overline{\lambda}\mu\tilde{\mu}$.* The following description is in the style of Section 2.13 of Herbelin's habilitation thesis [18].

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

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

Figure 3: A spectrum of intuitionistic calculi

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

Sequent Calculus   Natural Deduction

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 $\tilde{\mu}$:

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

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

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

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 $\lambda \mathbf{J}^{\mathbf{m}se}$-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 $l@E$.

3.2. **A spectrum of intuitionistic calculi.** The calculus $\lambda \mathbf{J}^{\mathbf{m}se}$ can also be explained as the end product of successive extensions of the simply-typed λ-calculus through several intuitionistic calculi, as illustrated in Fig. 3, which includes both natural deduction systems and sequent calculi other than $\lambda \mathbf{J}^{\mathbf{m}se}$.

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 [21]. $\lambda \mathbf{J}^{\mathbf{m}}$ is a variant of

Figure 4: Typing rules of $\lambda\mathbf{J}$

$$\frac{}{\Gamma, x : A \vdash x : A} \; Ax \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B} \; Intro$$

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

the system in [10]. $\lambda\mathbf{J}^{\mathbf{m}se}$ is studied in [9] 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 [10]), had the route been a different one, i.e., had the different new features been added in a different order. The reader is referred to the literature for the specific motivations underlying the introduction of the intermediate systems $\lambda\mathbf{J}$, $\lambda\mathbf{J}^{\mathbf{m}}$, and $\lambda\mathbf{J}^{\mathbf{m}s}$. Here, their interest lies in being the successive systems obtained by the addition, in a specific order, of the features exhibited by $\lambda\mathbf{J}^{\mathbf{m}se}$.

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 allowing a mapping that strictly simulates 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 $\lambda\mathbf{J}^{\mathbf{m}se}$. Let us see all this with some detail.

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

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

Construction $t(u, x.v)$ is called generalised application. Following [21], $(u, x.v)$ 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 the new rule is that of generalised application, given in Figure 4.

Reduction rules are as in [21], except that π is defined in the "eager" way:

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

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

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

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 [22], but in the present article, we even get an embedding into λ.

Although we won't use it, we recall the embedding $J : \lambda \to \lambda\mathbf{J}$ just for completeness:

$$\begin{aligned} J(x) &= x \\ J(\lambda x.t) &= \lambda x.J(t) \\ J(tu) &= J(t)(J(u), x.x) \end{aligned}$$

Figure 5: Typing rules of $\lambda \mathbf{J^m}$

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

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

$$\frac{\Gamma, x : A \vdash v : B}{\Gamma | (x)v : A \vdash B} \, Sel$$

3.2.2. $\lambda \mathbf{J^m}$-*calculus.* We offer now a new, lighter, presentation of the system in [10]. The expressions of $\lambda \mathbf{J^m}$ are given by the grammar:

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

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 [10]. There are two kinds of sequents: $\Gamma \vdash t : A$ and $\Gamma | l : A \vdash B$. Typing rules are given in Figure 5.

We re-define reduction rules of [10] 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).

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

$\beta = \beta_1 \cup \beta_2$. 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

$$
\begin{array}{rcl}
(u :: l)@S & = & u :: (l@S) \\
((x)V)@S & = & (x)VS, \quad \text{for } V \text{ a value} \\
((x)t(u, l))@S & = & (x)t(u, l@S)
\end{array}
$$

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

$$
\begin{array}{rcl}
m(x) & = & x \\
m(\lambda x.t) & = & \lambda x.m(t) \\
m(t(u, x.v)) & = & m(t)(m(u), (x)m(v))
\end{array}
$$

Figure 6: Typing rules of $\lambda\mathbf{J^{ms}}$: $GMApp$ of $\lambda\mathbf{J^m}$ is generalized to Cut

$$\frac{}{\Gamma, x : A \vdash x : A} \; Ax \qquad \frac{\Gamma \vdash t : A \quad \Gamma | l : A \vdash B}{\Gamma \vdash tl : B} \; Cut$$

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

$$\frac{\Gamma, x : A \vdash v : B}{\Gamma | (x)v : A \vdash B} \; Sel$$

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

(Terms) $\quad t, u, v \quad ::= \quad x \, | \, \lambda x.t \, | \, tl \qquad$ (Co-terms) $\quad l \quad ::= \quad u :: l \, | \, (x)v$

The construction tl has a double role: either it is a generalised and multiary application $t(u :: l)$ or it is an explicit substitution $t(x)v$. See Figure 6 for the typing rules.

The reduction rules are as follows:

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

where the co-term $l@l'$ is defined by

$$\begin{array}{rcl}
(u :: l)@l' & = & u :: (l@l') \\
((x)V)@l' & = & (x)Vl', \quad \text{for } V \text{ a value} \\
((x)tl)@l' & = & (x)t\,(l@l')
\end{array}$$

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

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

$$\begin{array}{rclcrcl}
s(x) & = & x & & s(u :: l) & = & s(u) :: s(l) \\
s(\lambda x.t) & = & \lambda x.s(t) & & s((x)v) & = & (x)s(v) \\
s(t(u, l)) & = & s(t)(s(u) :: s(l))
\end{array}$$

Finally, let us compare $\lambda\mathbf{J^{ms}}$ and $\lambda\mathbf{J^{mse}}$. 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 $\lambda\mathbf{J^{mse}}$ we can coerce any term t to a command $t[]$, we can translate $\lambda\mathbf{J^{ms}}$ into $\lambda\mathbf{J^{mse}}$, by defining $e((x)t) = (x)e(t)[]$. In fact, one has to refine this idea in order to get strict simulation of reduction. The embedding $e : \lambda\mathbf{J^{ms}} \to \lambda\mathbf{J^{mse}}$ is defined as

$$\begin{array}{rclcrcl}
e(x) & = & x & & e(u :: l) & = & e(u) :: e(l) \\
e(\lambda x.t) & = & \lambda x.e(t) & & e((x)V) & = & (x)e(V)[] \\
e(tl) & = & \{e(t)e(l)\} & & e((x)tl) & = & (x)e(t)e(l)
\end{array}$$

Proposition 3.1. *Each of the embeddings m, s and e preserves typability and types and strictly simulates reduction.*

Proof. Preservation of typability and types is immediate by induction on typing derivations. For strict simulation, we prove by induction

(i): $t \to t' \implies m(t) \to^+ m(t')$, for any $t, t' \in \lambda\mathbf{J}$

(ii): $t \to t' \implies s(t) \to^+ s(t')$, for any $t, t' \in \lambda\mathbf{J^m}$

(iii): $t \to t' \implies e(t) \to^+ e(t')$ and $e((x)t) \to^+ e((x)t')$, for any $t, t' \in \lambda\mathbf{J^{ms}}$

which for $f \in \{s, e\}$ requires simultaneous proof of: $l \to l' \implies f(l) \to^+ f(l')$. We show only some details of the proof of (iii). (The other statements have simpler proofs.) In the cases where $t \to_R t'$ (resp. $l \to l'$), with $R \in \{\beta, \pi, \mu\}$, in $\lambda\mathbf{J^{ms}}$, we have $e(t) \to_R e(t')$ and $e((x)t) \to_R e((x)t')$ (resp. $e(l) \to_R e(l')$) in $\lambda\mathbf{J^{mse}}$. The proof relative to π-steps requires commutation of the embedding with the append operator, that is requires the identity: $e(l@l') = e(l)@e(l')$, for any $l, l' \in \lambda\mathbf{J^{ms}}$. For σ-steps the situation is different: one σ-step in $\lambda\mathbf{J^{ms}}$ gives rise to one σ-step in $\lambda\mathbf{J^{mse}}$ but also, possibly, to π and ϵ steps. We consider below the base case of σ-reduction. The following two observations are needed:

(1) $(y)e(t)[] \to_\pi^* e((y)t)$, for any $t \in \lambda\mathbf{J^{ms}}$ and any variable y;

(2) $[e(t)/x]e(u) \to_\pi^* e([t/x]u)$, for any $t, u \in \lambda\mathbf{J^{ms}}$ and any variable x.

In the first observation, one can say more specifically that no π-step is required if t is a value and otherwise, if t is a command, exactly one π-step of the form $\{c\}[] \to c$ is needed (with c a command). The second observation uses the first and is proved simultaneously with its analogue for co-terms.

Let us then consider the case where we have the reduction $t(x)v \to [t/x]v$ in $\lambda\mathbf{J^{ms}}$. We concentrate on the sub-case $v = V$. (The other sub-case, where $v = t_0 l_0$, is similar.)

$$
\begin{aligned}
e(t(x)V) &= \{e(t)(x)e(V)[]\} \\
&\to_\sigma \{[e(t)/x](e(V)[])\} \\
&= \{[e(t)/x]e(V)[]\} \\
&\to_\epsilon [e(t)/x]e(V) \\
&\to_\pi^* e([t/x]V) \qquad \text{(Observation (2) above)}
\end{aligned}
$$

Now we need to prove: $e((y)t(x)v) \to^+ e((y)[t/x]v)$. We consider the possible forms of V. Sub-sub-case $V = x$.

$$
\begin{aligned}
e((y)t(x)x) &= (y)e(t)(x)x[] \\
&\to_\sigma (y)e(t)[] \\
&\to_\pi^* e((y)t) \qquad \text{(Observation (1) above)} \\
&= e((y)[t/x]x)
\end{aligned}
$$

Sub-sub-case $V = z$, with z a variable distinct of x:

$$
\begin{aligned}
e((y)t(x)z) &= (y)e(t)(x)z[] \\
&\to_\sigma (y)z[] \\
&= e((y)[t/x]z)
\end{aligned}
$$

Sub-sub-case $V = \lambda z.u$:

$$
\begin{aligned}
e((y)t(x)\lambda z.u) &= (y)e(t)(x)e(\lambda z.u)[] \\
&\to_\sigma (y)[e(t)/x]e(\lambda z.u)[] \\
&\to_\pi^* (y)e([t/x](\lambda z.u))[] \qquad \text{(Observation (2) above)} \\
&= e((y)[t/x](\lambda z.u)) \qquad ([t/x](\lambda z.u) \text{ is a value)}
\end{aligned}
$$

□

3.3. **Confluence.** For many purposes, it should suffice to have local confluence, which we do have for all the systems of this article, since in all of them, the critical pairs are joinable. Hence, thanks to Newman's lemma, all systems are confluent on typable terms since they are strongly normalizing, as shown in the later sections. We also believe that the usual methods that show the diamond property for properly defined notions of parallel reduction would yield confluence of all our systems. The aim of this section is to give indirect proofs for the systems of the spectrum, by inheriting confluence that is already known.

Firstly, we argue about confluence of $\lambda\mathbf{J}$ and $\lambda\mathbf{J^m}$. Secondly, we define and study a mapping from $\lambda\mathbf{J^{ms}}$ to $\lambda\mathbf{J^m}$. Thirdly, we apply the "interpretation method" to obtain confluence also of $\lambda\mathbf{J^{ms}}$. Finally, we do the same for $\lambda\mathbf{J^{ms}}$ and $\lambda\mathbf{J^{mse}}$ in order to infer confluence of $\lambda\mathbf{J^{mse}}$.

Confluence of $\lambda\mathbf{J}$ can be obtained from confluence of the original system $\lambda\mathbf{J}$ in [21] where π is *lazy*. Below we call $\widehat{\pi}$ the original lazy version of π, which reduces $t(u, x.v)S$ only to $t(u, x.vS)$ (for v a value V, there is no difference between π and $\widehat{\pi}$). Confluence for $\to_{\beta\pi}$ is obtained from confluence of $\to_{\beta\widehat{\pi}}$ in the same way as in [11] confluence of $\to_{\beta\pi'}$ is obtained from $\to_{\beta\widehat{\pi}}$, where π' is yet another variant of π.

Theorem 3.2. $\to_{\beta\pi}$ *in* $\lambda\mathbf{J}$ *is confluent.*

Proof. Assume $t \to^*_{\beta\pi} t_1$ and $t \to^*_{\beta\pi} t_2$. Then, also $t \to^*_{\beta\widehat{\pi}} t_1$ and $t \to^*_{\beta\widehat{\pi}} t_2$, and by confluence of $\to_{\beta\widehat{\pi}}$ there exists t_3 such that $t_1 \to^*_{\beta\widehat{\pi}} t_3$ and $t_2 \to^*_{\beta\widehat{\pi}} t_3$. The facts

(1) $t' \to^*_\pi \pi(t')$, for all t' in $\lambda\mathbf{J}$,
(2) $t' \to^*_{\beta\widehat{\pi}} t''$ implies $\pi(t') \to^*_{\beta\pi} \pi(t'')$, for all t', t'' in $\lambda\mathbf{J}$,

where notation $\pi(t')$ represents the π normal form of term t' (definable by recursion on t', using a very eager form of generalised application [21]), allow to conclude that t_1, t_2 both $\beta\pi$-reduce to $\pi(t_3)$.   □

What has been said above for $\lambda\mathbf{J}$ can be recast for $\lambda\mathbf{J^m}$, and confluence of $\to_{\beta\pi\mu}$ obtained from confluence of $\to_{\beta\widehat{\pi}\mu}$ [11]. In $\lambda\mathbf{J^m}$, the lazy π rule reads $tRS \to t(R\widehat{@}S)$, where $(u, l)\widehat{@}S = (u, l\widehat{@}S)$, and $(u :: l)\widehat{@}S = u :: (l\widehat{@}S)$ and $((x)t)\widehat{@}S = (x)tS$.

Theorem 3.3. $\to_{\beta\pi\mu}$ *in* $\lambda\mathbf{J^m}$ *is confluent.*

Proof. The proof above holds if β is replaced by $\beta\mu$. In particular, we have

(1) $t' \to^*_\pi \pi(t')$, for all t' in $\lambda\mathbf{J^m}$,
(2) $t' \to^*_{\beta\widehat{\pi}\mu} t''$ implies $\pi(t') \to^*_{\beta\pi\mu} \pi(t'')$, for all t', t'' in $\lambda\mathbf{J^m}$.   □

Now consider confluence of $\lambda\mathbf{J^{ms}}$. In this case, we cannot rely on a previous result of confluence for some variant of the system. Instead, we will lift the confluence result of [11] to $\lambda\mathbf{J^{ms}}$. First, we define a mapping $(\_)^\dagger : \lambda\mathbf{J^{ms}} \to \lambda\mathbf{J^m}$ in Figure 7.

Proposition 3.4.

(1) *For all* $t \in \lambda\mathbf{J^{ms}}$, $t \to^*_\sigma s(t^\dagger)$.
(2) *If* $t \to u$ *in* $\lambda\mathbf{J^{ms}}$, *then* $t^\dagger \to^*_{\beta\widehat{\pi}\mu} u^\dagger$ *in* $\lambda\mathbf{J^m}$.

Figure 7: Translation of $\lambda \mathbf{J}^{\mathbf{m}s}$ into $\lambda \mathbf{J}^{\mathbf{m}}$

$$
\begin{aligned}
x^\dagger &= x \\
(\lambda x.t)^\dagger &= \lambda x.t^\dagger \\
(t(x)v)^\dagger &= [t^\dagger/x]v^\dagger \\
(t(u :: l))^\dagger &= t^\dagger(u^\dagger, l^\dagger) \\
((x)v)^\dagger &= (x)v^\dagger \\
(u :: l)^\dagger &= u^\dagger :: l^\dagger
\end{aligned}
$$

Proof. 1. The claim is proved together with the similar claim for $l \in \lambda \mathbf{J}^{\mathbf{m}s}$ by simultaneous induction on t and l.

2. The claim is proved together with the similar claim for $l \to l'$ in $\lambda \mathbf{J}^{\mathbf{m}s}$. The proof is by simultaneous induction on $t \to u$ and $l \to l'$. The proof uses the following facts:

(i) $(\lambda x.t^\dagger)(u^\dagger, l^\dagger) \to_\beta [u^\dagger/x](tl)^\dagger$.

(ii) $(tl_1)^\dagger(u^\dagger, l_2{}^\dagger) \to_{\widehat{\pi}}^* (t(l_1@(u :: l_2)))^\dagger$ and $l_1{}^\dagger \widehat{@}(u^\dagger, l_2{}^\dagger) \to_{\widehat{\pi}}^* (l_1@u :: l_2)^\dagger$.

(iii) $[t^\dagger/x]v^\dagger = ([t/x]v)^\dagger$.

(iv) $(x)(xl)^\dagger \to_\mu^= l^\dagger$, if $x \notin l$.[4]

(i) and (iv) are proved by case analysis of l. (ii) is proved by induction on l_1. (iii) is proved together with $[t^\dagger/x]l^\dagger = ([t/x]l)^\dagger$ by simultaneous induction on v and l.[5] $\qquad\square$

Theorem 3.5. $\to_{\beta\pi\sigma\mu}$ *in* $\lambda \mathbf{J}^{\mathbf{m}s}$ *is confluent.*

Proof. Suppose $t \to_{\beta\pi\sigma\mu}^* t_i$, $i = 1, 2$, in $\lambda \mathbf{J}^{\mathbf{m}s}$. By part 2 of Proposition 3.4, $t^\dagger \to_{\beta\widehat{\pi}\mu}^* t_i{}^\dagger$ in $\lambda \mathbf{J}^{\mathbf{m}}$. By confluence [11], there is $u \in \lambda \mathbf{J}^{\mathbf{m}}$ such that $t_i{}^\dagger \to_{\beta\widehat{\pi}\mu}^* u$. By property 2 in the proof of Theorem 3.3, we get $t_i{}^\dagger \to_{\beta\pi\mu}^* \pi(u)$. By the properties of mapping $s : \lambda \mathbf{J}^{\mathbf{m}} \to \lambda \mathbf{J}^{\mathbf{m}s}$, we get $s(t_i{}^\dagger) \to_{\beta\pi\sigma\mu}^* s(\pi(u))$. We close the diagram in $\lambda \mathbf{J}^{\mathbf{m}s}$ because $t_i \to_\sigma^* s(t_i{}^\dagger)$. $\qquad\square$

Finally we consider confluence of $\lambda \mathbf{J}^{\mathbf{m}se}$. We will lift confluence of $\lambda \mathbf{J}^{\mathbf{m}s}$. First, we define a mapping $(\_)^\circ : \lambda \mathbf{J}^{\mathbf{m}se} \to \lambda \mathbf{J}^{\mathbf{m}s}$ in Figure 8 whose intuitive idea is that, in some sense, $\lambda \mathbf{J}^{\mathbf{m}se}$ is a subsystem of $\lambda \mathbf{J}^{\mathbf{m}s}$ – precisely the subsystem where selection is restricted to the cases $(x)x$ and $(x)tl$.

Proposition 3.6.

(1) *For all* $t \in \lambda \mathbf{J}^{\mathbf{m}se}$, $e(t^\circ) \to_\mu^* t$.

(2) *If* $t \to u$ *in* $\lambda \mathbf{J}^{\mathbf{m}se}$, *then* $t^\circ \to^+ u^\circ$ *in* $\lambda \mathbf{J}^{\mathbf{m}s}$.

Proof. Claim 1 is proved together with the similar claim $e(l^\circ) \to_\mu^* l$, by simultaneous induction on t and l.

Claim 2 for μ and ϵ is a direct verification. Since there are no commands in $\lambda \mathbf{J}^{\mathbf{m}s}$, one would have to study always two versions of β, π and σ: once inside braces $\{\}$, once bound by (y). However, since all three rules have the form $t_1 l_1 \to t_2 l_2$, it suffices to

---

[4] $\to_{\overline{\overline{R}}}$ denotes the reflexive closure of $\to_R$.

[5] In $\lambda \mathbf{J}^{\mathbf{m}}$ one has to use $\widehat{\pi}$ and not π for statement (2) to hold. Consider the $\lambda \mathbf{J}^{\mathbf{m}s}$-terms $v_0 = t_0(u_0 :: (x)(t_1(z)z))(u :: k)$ and $v_1 = t_0(u_0 :: (x)t_1(z)z(u :: k))$. Then $v_0 \to_\pi v_1$ but $v_0{}^\dagger \to_\pi v_1{}^\dagger$ fails.

Figure 8: Embedding of $\lambda\mathbf{J^{mse}}$ into $\lambda\mathbf{J^{ms}}$

$$
\begin{aligned}
x^\circ &= x \\
(\lambda x.t)^\circ &= \lambda x.t^\circ \\
\{tl\}^\circ &= t^\circ l^\circ \\
[]^\circ &= (x)x \\
((x)tl)^\circ &= (x)t^\circ l^\circ \\
(u :: l)^\circ &= u^\circ :: l^\circ
\end{aligned}
$$

Figure 9: Description of μ-normalisation function in $\lambda\mathbf{J^{mse}}$

$$
\begin{aligned}
\mu x &= x \\
\mu(\lambda x.t) &= \lambda x.\mu t \\
\mu\{tl\} &= \{\mu t\, \mu l\} \\
\mu[] &= [] \\
\mu((x)tl) &= \mu l \ (\text{if } t = x \text{ and } x \notin l) \\
\mu((x)tl) &= (x)\mu t\mu l \ (\text{otherwise}) \\
\mu(u :: l) &= \mu u :: \mu l
\end{aligned}
$$

verify $t_1^\circ l_1^\circ \to^+ t_2^\circ l_2^\circ$ for them. For σ, we also need the facts $([t/x]u)^\circ = [t^\circ/x]u^\circ$ and $([t/x]l)^\circ = [t^\circ/x]l^\circ$, and for the non-nil case of π, the fact $l^\circ@(u_1 :: l_1)^\circ \to_\mu (l@u_1 :: l_1)^\circ$ is proved by induction on l. □

The first statement of the previous proposition is an obstacle to an immediate application of the "interpretation method", because the μ-reduction goes in the wrong direction. We overcome this by observing that, as a consequence of $e(t^\circ) \to_\mu^* t$, we have $t \to_\mu^* \mu(e(t^\circ))$. (Here μ is the function that assigns the μ-normal form of an expression. Clearly, reduction rule μ alone is terminating and locally confluent, hence confluent.) So, in the proof of confluence (Theorem 3.8 below) there will be an extra step relying on the properties of mapping μ, which is explicitly given in Figure 9.

Proposition 3.7. In $\lambda\mathbf{J^{mse}}$, if $t \to u$, then $\mu t \to^* \mu u$.

Proof. The claim is proved together with the similar claim for $l \to l'$ by simultaneous induction on $t \to u$ and $l \to l'$. The proof makes use of the following facts: (i) $(x)\mu(t)\mu(l) \to_\mu^= \mu((x)tl)$; (ii) commutation of mapping μ with substitution; (iii) commutation of mapping μ with append. Fact (i) is immediate from definition. Facts (ii) and (iii) are proved by easy inductions. □

Theorem 3.8. $\to_{\beta\pi\sigma\mu\epsilon}$ in $\lambda\mathbf{J^{mse}}$ is confluent.

Proof. Suppose $t \to^*_{\beta\pi\sigma\mu\epsilon} t_i$, $i = 1, 2$, in $\lambda\mathbf{J^{m}}^{se}$. By part 2 of Proposition 3.6, $t^\circ \to^*_{\beta\pi\sigma\mu} t_i^\circ$ in $\lambda\mathbf{J^{m}}^s$. By confluence (Theorem 3.5), there is $u \in \lambda\mathbf{J^{m}}^s$ such that $t_i^\circ \to^*_{\beta\pi\sigma\mu} u$. By the properties of mapping $e : \lambda\mathbf{J^{m}}^s \to \lambda\mathbf{J^{m}}^{se}$, we get $e(t_i^\circ) \to^*_{\beta\pi\sigma\mu\epsilon} e(u)$. Proposition 3.7 yields $\mu(e(t_i^\circ)) \to^*_{\beta\pi\sigma\mu\epsilon} \mu(e(u))$. We close the diagram in $\lambda\mathbf{J^{m}}^{se}$ because $t_i \to^*_\mu \mu(e(t_i^\circ))$. $\square$

Notice that we might have inferred confluence of $\lambda\mathbf{J^{m}}^{se}$ of that of the call-by-name $\overline{\lambda}\mu\tilde{\mu}$-calculus, presented in Section 3.1.1: if this calculus is confluent, then its intuitionistic fragment is confluent as well since it has just the same rules on a subset of terms, co-terms and commands that is closed under reduction. Finally, its isomorphic copy $\lambda\mathbf{J^{m}}^{se}$ would be confluent as well. However, we are not aware of a proof of confluence of our version of call-by-name $\overline{\lambda}\mu\tilde{\mu}$-calculus: the calculus considered in [25] does not have the rules μ and $\tilde{\mu}$, has a more restrictive notion of evaluation contexts and imposes σ-reduction immediately following applications of β. As mentioned above, we would expect that the standard direct proof methods would be applicable to establish confluence of all of the systems considered in this section.

4. CGPS TRANSLATIONS

In this section we define a CPS translation for $\lambda\mathbf{J^{m}}^{se}$ into the simply-typed λ-calculus and show how it fails to provide a strict simulation of reduction. Next we refine the CPS translation to a CGPS translation of $\lambda\mathbf{J^{m}}^{se}$ and show that strict simulation of reduction is obtained. Strong normalisation for $\lambda\mathbf{J^{m}}^{se}$ follows. Finally, we adapt the CGPS translation to the subsystems of $\lambda\mathbf{J^{m}}^{se}$.

4.1. **CPS translation for $\lambda\mathbf{J^{m}}^{se}$.** We assume the reader is familiar with simply-typed lambda-calculus (we write $A \supset B$ for the function type $A \to B$ and $\to_\beta$ for the one-step reduction relation). Fix a ground type (some type variable) $\bot$. Then, $\neg A := A \supset \bot$, 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 [6] 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):

$$\begin{aligned} X^* &= X \\ (A \supset B)^* &= \neg\overline{B} \supset \neg\overline{A} \end{aligned}$$

We thus obtain

$$\begin{aligned} \overline{X} &= \neg\neg X \\ \overline{A \supset B} &= \neg\neg(\neg\overline{B} \supset \neg\overline{A}) \end{aligned}$$

The 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 even for weak simulation to hold. See Subsection 4.4 for a discussion of this issue.

The translation of all syntactic elements T will be presented in Plotkin's [32] colon notation $(T : K)$ for some term K taken from simply-typed λ-calculus. A term t of $\lambda\mathbf{J^{m}}^{se}$ will then be translated into the simply-typed λ-term

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

Figure 10: CPS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$

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

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

Figure 11: Admissible typing rules for CPS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$

$$
\frac{\Gamma \vdash t : A \quad \overline{\Gamma}, \Gamma' \vdash K : \neg A^*}{\overline{\Gamma}, \Gamma' \vdash (t : K) : \bot}
\qquad
\frac{\Gamma \xrightarrow{c} A \quad \overline{\Gamma}, \Gamma' \vdash K : \neg A^*}{\overline{\Gamma}, \Gamma' \vdash (c : K) : \bot}
$$

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

with a "fresh" variable k (one that is not free in t). The definition of $(T : K)$ in Figure 10 uses the definition of $\overline{t}$ as an abbreviation (the variables m, w are supposed to be "fresh", in the obvious sense). The translation admits the typing rules of Figure 11.[6] Only the first premise in these three rules refers to $\lambda\mathbf{J}^{\mathbf{m}se}$, 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:

$$
\Gamma \vdash_{\lambda\mathbf{J}^{\mathbf{m}se}} t : A \Longrightarrow \overline{\Gamma} \vdash_\lambda \overline{t} : \overline{A}
$$

This CPS translation is also sound for reduction, in the sense that each reduction step in $\lambda\mathbf{J}^{\mathbf{m}se}$ 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 $\lambda\mathbf{J}^{\mathbf{m}se}$.

Proposition 4.1. *If $t \to u$ in $\lambda\mathbf{J}^{\mathbf{m}se}$, then $\overline{t} \to^*_\beta \overline{u}$ 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: it identifies ϵ and π-steps, sends one μ-step into zero or more β-steps in λ-calculus and sends one β or σ-step into one or more β-steps in λ-calculus. Some comments on lemmata used in this proof can be found in the next section. $\qquad\square$

---

[6]Regrettably, the contexts Γ' observed in these rules, as well as those observable below in the rules of Fig. 13, were missing in [12].

4.2. **CGPS translation for $\lambda \mathbf{J}^{\mathbf{m}se}$.** 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 [20]. 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 [29], we ensure strict 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 [6] (there in order to treat disjunction and not for sequent calculi as we do).

Our CGPS translation passes "garbage", in addition to continuations. We mean by "garbage" λ-terms, denoted G, that are carried around for their operational properties, not for denotational purposes. They inhabit a type $\top$, of which we only require that there is a term $\mathsf{s} : \top \supset \top$ such that $\mathsf{s}\,G \to_\beta^+ G$. This can of course be realized by any type, with $\mathsf{s} := \lambda x.x$, but it is useful, in view of a comparison with [20], to have in mind another realization, namely $\top := \bot \supset \bot$ and $\mathsf{s} := \lambda x.[x; \lambda z.z]$. Here we are using the abbreviation $[t; u] := (\lambda x.t)u$ for some $x \notin t$. Then, $[t; u] \to_\beta t$, and $\Gamma \vdash t : A$ and $\Gamma \vdash u : B$ together imply $\Gamma \vdash [t; u] : A$ (as a derived typing rule of simply-typed λ-calculus). This is a form of "deliberate garbage" that is used in [20]. Instead of $\mathsf{s}\,G$, we will write $\mathsf{s}(G)$. We will also speak about "units of garbage". This is so because, in our translation, garbage will always have one of the forms g (a variable), $\mathsf{s}(g)$, $\mathsf{s}(\mathsf{s}(g))$, etc. We say that $\mathsf{s}(G)$ has one more "unit of garbage" than G, or that, in $\mathsf{s}(G)$, G is "incremented". In the particular realization $\top := \bot \supset \bot$, $\mathsf{s}(G) =_\beta [G; \lambda z.z]$; we may regard $\lambda z.z$ (which lives in $\top$) as the "unit" that is added to G. In [20], garbage is built by "adding" a continuation K to G, as in $[G; K]$.

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 $\lambda\mathbf{J}^{\mathbf{m}se}$ and simply-typed λ-terms G and K. Then, the translation of term t is defined to be

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

with "new" variables g, k, that is again used as an abbreviation inside the recursive definition of $(T : G, K)$ in Figure 12 (the variables m, w are again "fresh").[7]

If one removes the garbage argument, one precisely obtains the CPS translation. The translation admits the typing rules of Figure 13.

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 $\overline{\Gamma} \vdash \overline{t} : \overline{A}$.

Lemma 4.2. *In $\lambda\mathbf{J}^{\mathbf{m}se}$ the following holds:*

(1) $[\overline{t}/x](T : G, K) \to_\beta^* ([t/x]T : [\overline{t}/x]G, [\overline{t}/x]K)$ *for T any u, l or c, and, in particular,*
    $[\overline{t}/x]\overline{u} \to_\beta^* \overline{[t/x]u}$.

---

[7]There is a slight, but important, difference between the *definition* of the CGPS translation presented here and that presented in [12]. In [12], several clauses in the definition of $(l : G, K)$ or $(c : G, K)$ contained garbage increment $\mathsf{s}(G)$, whereas in the present definition those increments are, so to speak, concentrated in the clause for $(\{c\} : G, K)$. The importance of this re-definition is that it makes the purpose of those increments more perspicuous - see the discussion around the simulation theorem below. For the sake of a precise connection between the two definitions, let us write the translation of [12] as $[T : G, K]$ and $\overline{\overline{t}}$. Then, by an easy induction, one obtains $(t : G, K) = [t : G, K]$, $(l : \mathsf{s}(G), K) = [l : G, K]$, and $(c : \mathsf{s}(G), K) = [c : G, K]$. Hence $\overline{t} = \overline{\overline{t}}$.

Figure 12: CGPS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$

$$
\begin{array}{rcl}
(x : G, K) & = & x\,\mathsf{s}(G)K \\
(\lambda x.t : G, K) & = & [K(\lambda wx.w\overline{t}); G] \\
(\{c\} : G, K) & = & (c : \mathsf{s}(G), K) \\[4pt]
([\,] : G, K) & = & \lambda w.w\,GK \\
(u :: l : G, K) & = & \lambda w.w\,G(\lambda m.m\,(l : G, K)\,\overline{u}) \\
((x)c : G, K) & = & \lambda x.(c : G, K) \\[4pt]
(t[\,] : G, K) & = & (t : G, K) \\
(t(u :: l) : G, K) & = & (t : G, \lambda m.m\,(l : G, K)\,\overline{u}) \\
(t(x)c : G, K) & = & ((x)c : G, K)\overline{t}
\end{array}
$$

Figure 13: Admissible typing rules for CGPS translation of $\lambda\mathbf{J}^{\mathbf{m}se}$

$$
\frac{\Gamma \vdash t : A \quad \overline{\Gamma}, \Gamma' \vdash G : \top \quad \overline{\Gamma}, \Gamma' \vdash K : \neg A^*}{\overline{\Gamma}, \Gamma' \vdash (t : G, K) : \bot}
\qquad
\frac{\Gamma | l : A \vdash B \quad \overline{\Gamma}, \Gamma' \vdash G : \top \quad \overline{\Gamma}, \Gamma' \vdash K : \neg B^*}{\overline{\Gamma}, \Gamma' \vdash (l : G, K) : \neg\overline{A}}
$$

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

(2) $[t/x](T : G, K) = (T : [t/x]G, [t/x]K)$ for T any u, l or c such that $x \notin T$.

(3) G and K are subterms of $(T : G, K)$ for T any u, l or c.

(4) $(t : \mathsf{s}(G), K) \to_\beta^+ (t : G, K)$.

(5) $(l : G, K)\overline{t} \to_\beta^* (tl : G, K)$

(6) $\lambda x.(xl : G, K) \to_\beta^+ (l : G, K)$ if $x \notin l, G, K$.

(7) (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. Notice that the second statement has to be proven simultaneously, but that it follows immediately from the particular case $T = u$ of the first statement.

4.

$$
\begin{array}{rcll}
(t : \mathsf{s}(G), K) & = & [\mathsf{s}(G)/g](t : g, K) & \text{(by 2.)} \\
& \to_\beta^+ & [G/g](t : g, K) & (*) \\
& = & (t : G, K) & \text{(by 2.)}
\end{array}
$$

where $(*)$ is justified by the fact that g occurs in $(t : g, K)$, as guaranteed by 3.

5. Straightforward case analysis on l.

6. Case analysis on l.

Case $l = [\,]$.

$$\begin{aligned}
\lambda x.(x[] : G, K) &= \lambda x.(x : G, K)\\
&= \lambda x.x\,\mathsf{s}(G)K\\
&\to_\beta^+ \lambda x.xGK\\
&= ([] : G, K) \qquad (\text{as } x \notin G, K)
\end{aligned}$$

Case $l = u :: l'$.

$$\begin{aligned}
\lambda x.(x(u :: l') : G, K) &= \lambda x.(x : G, \lambda m.m(l' : G, K)\overline{u})\\
&= \lambda x.x\,\mathsf{s}(G)(\lambda m.m(l' : G, K)\overline{u})\\
&\to_\beta^+ \lambda x.xG(\lambda m.m(l' : G, K)\overline{u})\\
&= (u :: l' : G, K) \qquad\qquad (\text{as } x \notin u, l', G, K)
\end{aligned}$$

Case $l = (y)c$.

$$\begin{aligned}
\lambda x.(x(y)c : G, K) &= \lambda x.(\lambda y.(c : G, K))\overline{x}\\
&\to_\beta \lambda x.[\overline{x}/y](c : G, K)\\
&= \lambda y.[y/x][\overline{x}/y](c : G, K)\\
&= \lambda y.[[y/x]\overline{x}/y](c : G, K) \quad (\text{as } x \notin c, G, K, \text{ and by 2.})\\
&= \lambda y.[\overline{y}/y](c : G, K)\\
&\to_\beta^* \lambda y.([y/y]c : G, K) \qquad (\text{by 1.})\\
&= ((y)c : G, K)
\end{aligned}$$

7. (a) and (b) are proved simultaneously by induction on l.
Case $l = []$.

$$\begin{aligned}
(t[] : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u}) &= (t : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u})\\
&\to_\beta^+ (t : G, \lambda m.m(l' : G, K)\overline{u}) \qquad (\text{by 4.})\\
&= (t\,([]@(u :: l')) : G, K)
\end{aligned}$$

$$\begin{aligned}
([] : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u}) &= \lambda w.w\,\mathsf{s}(G)(\lambda m.m(l' : G, K)\overline{u})\\
&\to_\beta^+ \lambda w.wG(\lambda m.m(l' : G, K)\overline{u})\\
&= ([]@(u :: l') : G, K)
\end{aligned}$$

Case $l = u_0 :: l_0$.

$$\begin{aligned}
&(t(u_0 :: l_0) : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u})\\
=\ &(t : \mathsf{s}(G), \lambda n.n(l_0 : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u})\overline{u_0})\\
\to_\beta^+\ &(t : \mathsf{s}(G), \lambda n.n(l_0@(u :: l') : G, K)\overline{u_0}) \qquad (\text{by IH (b)})\\
\to_\beta^+\ &(t : G, \lambda n.n(l_0@(u :: l') : G, K)\overline{u_0}) \qquad (\text{by 4.})\\
=\ &(t\,((u_0 :: l_0)@(u :: l')) : G, K)
\end{aligned}$$

$$\begin{aligned}
&(u_0 :: l_0 : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u})\\
=\ &\lambda w.w\,\mathsf{s}(G)(\lambda n.n(l_0 : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u})\overline{u_0})\\
\to_\beta^+\ &\lambda w.w\,\mathsf{s}(G)(\lambda n.n(l_0@(u :: l') : G, K)\overline{u_0}) \qquad (\text{by IH (b)})\\
\to_\beta^+\ &\lambda w.wG(\lambda n.n(l_0@(u :: l') : G, K)\overline{u_0})\\
=\ &((u_0 :: l_0)@(u :: l') : G, K)
\end{aligned}$$

Case $l = (x)v_0l_0$. Part (b) follows from the induction hypothesis (a) for l_0, and part (a) is an immediate consequence of (b). $\qquad\square$

Theorem 4.3 (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: $T \to T' \implies (T : G, K) \to_\beta^+ (T' : G, K)$ for T, T' terms, co-terms or commands. We illustrate the cases of the base rules.

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

$$
\begin{aligned}
((\lambda x.t)(u :: l) : G, K) &= (\lambda x.t : G, \lambda m.m(l : G, K)\overline{u}) \\
&= [(\lambda m.m(l : G, K)\overline{u})(\lambda wx.w\overline{t}); G] \\
&\to_\beta^3 (\lambda x.(l : G, K)\overline{t})\overline{u} \\
&\to_\beta^* (\lambda x.(tl : G, K))\overline{u} \qquad \text{(Lemma 4.2.5)} \\
&= (u(x)tl : G, K)
\end{aligned}
$$

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

$$
\begin{aligned}
(\{tl\}[] : G, K) &= (\{tl\} : G, K) \\
&= (tl : \mathsf{s}(G), K) \\
&\to_\beta^+ (tl : G, K) \qquad \text{(Lemma 4.2.4)} \\
&= (t(l@[]) : G, K).
\end{aligned}
$$

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

$$
\begin{aligned}
(\{tl\}(u :: l') : G, K) &= (\{tl\} : G, \lambda m.m(l' : G, K)\overline{u}) \\
&= (tl : \mathsf{s}(G), \lambda m.m(l' : G, K)\overline{u}) \\
&\to_\beta^+ (t(l@(u :: l')) : G, K) \qquad \text{(Lemma 4.2.7)}
\end{aligned}
$$

Case σ: $t(x)c \to [t/x]c$.

$$
\begin{aligned}
(t(x)c : G, K) &= (\lambda x.(c : G, K))\overline{t} \\
&\to_\beta [\overline{t}/x](c : G, K) \\
&\to_\beta^* ([t/x]c : G, K) \qquad \text{(Lemma 4.2.1)}
\end{aligned}
$$

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

$$
((x)xl : G, K) = \lambda x.(xl : G, K) \to_\beta^+ (l : G, K) \qquad \text{(Lemma 4.2.6)}
$$

Case ϵ: $\{t[]\} \to t$.

$$
\begin{aligned}
(\{t[]\} : G, K) &= (t[] : \mathsf{s}(G), K) \\
&= (t : \mathsf{s}(G), K) \\
&\to_\beta^+ (t : G, K) \qquad \text{(Lemma 4.2.4)}
\end{aligned}
$$

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

Remark 4.4. Unlike the failed strict simulation by CPS reported in [29] 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 4.2.3 is responsible for propagation of strict simulation. The structure of our garbage – essentially just "units of garbage" – can thus be easier than in the CGPS in [20] 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 [20] 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. However, since we need garbage

for some base cases, we also had to make sure that reductions in garbage arguments are not lost during propagation through the closure rules.

Let us compare the CPS and CGPS translations in order to understand how garbage-passing ensures strict simulation. The analogue to Lemma 4.2 for the CPS translation is obtained by erasing garbage throughout, and replacing $\to_\beta^+$ by equality in items 4 and 7, and by $\to_\beta^*$ in item 6. So, the properties of the CGPS translation are as "good" as those of the CPS translation, and at least a weak simulation could be expected.

An inspection of the proofs of Lemma 4.2 and Theorem 4.3 shows that the CGPS translation generates reduction sequences which, so to speak, differ from those generated by CPS translation by the insertion of sequences of the form $\mathsf{s}(G) \to_\beta^+ G$. The point is that the CGPS translation does such insertion at all points where the CPS does an undesired identification (although it also does at other points where such insertion is unnecessary).

The ultimate cause for the existence of such dynamic *garbage decrement steps* is the static garbage increment contained in the clauses defining $(x : G, K)$ and $(\{c\} : G, K)$. Moreover, it can be argued that the clause for $(x : G, K)$ is responsible for strict simulation of μ-steps, whereas the clause for $(\{c\} : G, K)$ is the cause for strict simulation of π- or ϵ-steps.

The key for strict simulation of μ-steps is Lemma 4.2.6. An inspection of the proof shows that garbage plays no role in the case $l = (y)c$ (which already generated reduction steps through the CPS translation), and that, had $(x : G, K)$ been defined as xGK, the same identifications obtained before with the CPS translation would have arisen again in the cases $l = []$ and $l = u :: l'$. The definition of $(x : G, K)$ causes many garbage decrement steps, which are useless most of the time (typically adding to the administrative steps, already generated in the case of the CPS translations, that mediate between $\overline{[t/x]}\overline{u}$ and $\overline{[t/x]u}$), but not so in the particular situations described in the cases $l = []$ and $l = u :: l'$ of Lemma 4.2.6.

The role of clause $(\{c\} : G, K)$ is plain for ϵ and the case $E = []$ of π. As to the case $E = u :: l'$, it suffices to observe that $(tl : G, \lambda m.m(l' : G, K)\overline{u}) = (t\,(l@(u :: l')) : G, K)$ (as an inspection of the proof of Lemma 4.2.7 easily shows). So, again, had $(\{c\} : G, K)$ been defined as $(c : G, K)$, the same identifications of π- or ϵ-steps obtained before with the CPS translation would have arisen. The definition of $(\{c\} : G, K)$ means that garbage-passing does, among other things, some form of counting braces. The braces decrement observed in π- or ϵ-steps in the source is reflected by garbage decrement steps in the target.

Corollary 4.5. *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 [33] using reducibility candidates and before by Lengrand's [23] 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.

Since each of m, s and e preserves typability and strictly simulates reduction (Proposition 3.1), it follows from Corollary 4.5 that:

Corollary 4.6. *The typable terms of* $\lambda\mathbf{J}^{\mathbf{m}s}$, $\lambda\mathbf{J}^{\mathbf{m}}$ *and* $\lambda\mathbf{J}$ *are strongly normalising.*

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

(1) For $\lambda\mathbf{J^{ms}}$:

$$
\begin{aligned}
(tl : G, K) &= (tl; \mathsf{s}(G), K) \\
((x)V : G, K) &= \lambda x.(V : G, K) \ (V \text{ a value}) \\
((x)tl : G, K) &= \lambda x.(tl; G, K) \\
(t(x)v; G, K) &= ((x)v : G, K)\overline{t} \\
(t(u :: l); G, K) &= (t : G, \lambda m.m\,(l : G, K)\,\overline{u})
\end{aligned}
$$

(2) For $\lambda\mathbf{J^m}$: there is no auxiliary operator $(tl; G, K)$.

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

(3) Finally, for $\lambda\mathbf{J}$:

$$
\begin{aligned}
(t(u, x.V) : G, K) &= (t : \mathsf{s}(G), \lambda m.m\,(\lambda x.(V : \mathsf{s}(G), K))\,\overline{u}) \quad (V \text{ a value}) \\
(t(u, x.v) : G, K) &= (t : \mathsf{s}(G), \lambda m.m\,(\lambda x.(v : G, K))\,\overline{u}) \qquad (v \text{ an application})
\end{aligned}
$$

In the case of $\lambda\mathbf{J^{ms}}$, the distinction between $(tl : G, K)$ and $(tl; G, K)$ is consistent with the distinction, in $\lambda\mathbf{J^{mse}}$, between $(\{c\} : G, K)$ and $(c : G, K)$.[8]

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

Proposition 4.7. *Let $\mathcal{L} \in \{\lambda\mathbf{J^{ms}}, \lambda\mathbf{J^m}, \lambda\mathbf{J}\}$. Let $f_{\mathcal{L}}$ be the embedding of $\mathcal{L}$ in the immediate extension of $\mathcal{L}$ in the spectrum of Fig. 3, and let $g_{\mathcal{L}}$ be the embedding of $\mathcal{L}$ in $\lambda\mathbf{J^{mse}}$. Then, for all $t \in \mathcal{L}$, $\overline{t} = \overline{f_{\mathcal{L}}(t)}$. Hence, for all $t \in \mathcal{L}$, $\overline{t} = \overline{g_{\mathcal{L}}(t)}$.*

Proof. For $\lambda\mathbf{J^{ms}}$, let $P(t) := \forall G, K((t : G, K) = (e(t) : G, K))$, for every $t \in \lambda\mathbf{J^{ms}}$. Then, one proves

(i) $P(t)$; and

(ii) $(l : G, K) = (e(l) : G, K)$ and $\forall t \in \lambda\mathbf{J^{ms}}(P(t) \Longrightarrow (e(t)e(l) : G, K) = (tl; G, K))$

by simultaneous induction on t and l.

For $\lambda\mathbf{J^m}$, on proves $(t : G, K) = (s(t) : G, K)$ and $(l : G, K) = (s(l) : G, K)$ by simultaneous induction on t and l.

For $\lambda\mathbf{J}$, one proves $(t : G, K) = (m(t) : G, K)$ by induction on t. $\qquad\square$

Therefore, since each of m, s and e, as well as the CGPS translation of $\lambda\mathbf{J^{mse}}$, preserves typability, so does each CGPS translation of the subsystems.

Theorem 4.8 (Simulation). *Let $\mathcal{L} \in \{\lambda\mathbf{J^{ms}}, \lambda\mathbf{J^m}, \lambda\mathbf{J}\}$. If $t \to u$ in $\mathcal{L}$, then $\overline{t} \to_{\beta}^{+} \overline{u}$ in the λ-calculus.*

Proof. By Propositions 3.1 and 4.7 and Theorem 4.3. $\qquad\square$

---

[8]We take the opportunity to correct a mistake in the CGPS translations for the subsystems of $\lambda\mathbf{J^{mse}}$ given in [12]. The mistake was that some clauses in the definition of those translations lacked a needed case analysis. We repair the mistake now, using in this footnote the notations $(T : G, K)$ and $\overline{t}$ with their meanings in [12]. For $\lambda\mathbf{J^{ms}}$: $((x)v : G, K) = \lambda x.(v : G', K)$, where G' is either $\mathsf{s}(G)$, if v is a value; or G, otherwise. For $\lambda\mathbf{J^m}$, it should be understood that the clause for $((x)v : G, K)$ just given is inherited. For $\lambda\mathbf{J}$: $(t(u, x.v) : G, K) = (t : \mathsf{s}(G), \lambda m.m\,(\lambda x.(v : G', K))\,\overline{u})$, where G' is $\mathsf{s}(G)$, if v is a value; or G, otherwise.

Since the various CGPS translations preserve typability, Corollary 4.6 follows also from the previous theorem (and strong normalisation of the simply-typed λ-calculus).

The CGPS translations defined above for the subsystems of $\lambda\mathbf{J^{m}}^{se}$, being consistent with the CGPS translation of $\lambda\mathbf{J^{m}}^{se}$, have the advantage of inheriting the simulation theorem, and the disadvantage of not being optimized for the particular system on which they are defined. In fact, such translations can be optimized by omitting garbage increments $\mathsf{s}(G)$ in one or more of their clauses. We give one example of this phenomenon.

Theorem 4.9 (Simulation). *Let $\overline{t}$ and $(t : G, K)$ be given for $t \in \lambda\mathbf{J}$ by:*

$$
\begin{array}{rcl}
\overline{t} & = & \lambda gk.(t : g, k) \\
(x : G, K) & = & xGK \\
(\lambda x.t : G, K) & = & [K(\lambda wx.w\overline{t}); G] \\
(t(u, x.v) : G, K) & = & (t : \mathsf{s}(G), \lambda m.m\,(\lambda x.(v : G, K))\,\overline{u})
\end{array}
$$

If $t \to u$ in $\lambda\mathbf{J}$, then $\overline{t} \to_\beta^+ \overline{u}$ in the λ-calculus.

Proof. Similar to, but simpler than that of Theorem 4.3. $\qquad\qquad\square$

4.4. **C(G)PS translations with less double negations.** Our definition of $(A \supset B)^*$ produces a type logically equivalent to $\overline{A} \supset \neg\neg\overline{B}$, which has an extra double negation of $\overline{B}$ when compared with traditional CPS's. One may wonder what happens if one sets $(A \supset B)^* = \overline{A} \supset \overline{B}$. There is no problem in *defining* a CPS translation based on that, but we would even lose weak simulation in the form of Proposition 4.1. Let us take the simplified type translation, whose new clauses are:

$$
\begin{array}{rcl}
(\lambda x.t : K) & = & K(\lambda x.\overline{t}) \\
(u :: l : K) & = & \lambda w.w(\lambda m.(l : K)(m\overline{u})) \\
(t(u :: l) : K) & = & (t : \lambda m.(l : K)(m\overline{u}))
\end{array}
$$

This translation obeys to the typing rules of Figure 11, but already β steps at the root do not obey to Proposition 4.1:

$$
\begin{array}{rcl}
((\lambda x.t)(u :: []) : K) & = & (\lambda m.(\lambda w.wK)(m\overline{u}))(\lambda x.\overline{t}) \\
& \to_\beta^2 & (\lambda x.\overline{t})\overline{u}K \\
& =_\beta & (\lambda x.\overline{t}K)\overline{u} \\
& \to_\beta & (\lambda x.(t : K))\overline{u} \\
& = & (u(x)(t[]) : K)
\end{array}
$$

The problem is that there is no reduction step from $(\lambda x.\overline{t})\overline{u}K$ to $(\lambda x.\overline{t}K)\overline{u}$ in λ-calculus. Similar remarks apply to the subsystem $\lambda\mathbf{J^{ms}}$.

The failed simulation just illustrated would have not occurred, had the β rule been defined with implicit substitution:

$$(\lambda x.\overline{t})\overline{u}K \to_\beta ([\overline{u}/x]\overline{t})K = [\overline{u}/x](\overline{t}K) \to_\beta [\overline{u}/x](t : K) \to_\beta^* ([u/x]t : K) = (\{[u/x]t\}[] : K)$$

This is consistent with another fact: weak simulation, through the simpler CPS, is recovered as soon as one moves down in the spectrum to $\lambda\mathbf{J^m}$ or $\lambda\mathbf{J}$ (systems where β-reduction employs implicit substitution). For these systems, the combination of garbage passing with

the simpler CPS delivers strict simulation. The theorem below exemplifies the situation with $\lambda\mathbf{J}$ (to be compared with Theorem 4.9).

Theorem 4.10 (Simulation). *For a type A, let $\overline{A} = \top \supset \neg\neg A^*$, $X^* = X$ and $(A \supset B)^* = \overline{A} \supset \overline{B}$ and for $t \in \lambda\mathbf{J}$, let $\overline{t} = \lambda gk.(t : g, k)$ and $(t : G, K)$ be defined as*

$$
\begin{aligned}
\overline{t} &= \lambda gk.(t : g, k) \\
(x : G, K) &= xGK \\
(\lambda x.t : G, K) &= [K(\lambda x.\overline{t}); G] \\
(t(u, x.v) : G, K) &= (t : \mathsf{s}(G), \lambda m.(\lambda x.(v : G, K))(m\overline{u}))
\end{aligned}
$$

(1) If $\Gamma \vdash_{\lambda\mathbf{J}} t : A$ then $\overline{\Gamma} \vdash_\lambda \overline{t} : \overline{A}$.
(2) If $t \to u$ in $\lambda\mathbf{J}$, then $\overline{t} \to_\beta^+ \overline{u}$ in the λ-calculus.

Proof. The proof of (1) is based on the fact that the first rule of Figure 13 is still admissible. Property (2) follows along the lines of theorems 4.9 and 4.3, requiring some properties analogous to those in Lemma 4.2. We illustrate below the base case for β (problematic for $\lambda\mathbf{J}^{mse}$ and $\lambda\mathbf{J}^{ms}$, as explained above):

$$
\begin{aligned}
((\lambda x.t)(u, y.v) : G, K) &= (\lambda x.t : \mathsf{s}(G), \lambda m.(\lambda y.(v : G, K))(m\overline{u})) \\
&= [(\lambda m.(\lambda y.(v : G, K))(m\overline{u}))(\lambda x.\overline{t}); \mathsf{s}(G)] \\
&\to_\beta^4 [[\overline{u}/x]\overline{t}]/y](v : G, K) \\
&\to_\beta^* [[\overline{u}/x]t]/y](v : G, K) \\
&\to_\beta^* ([[u/x]t]/y]v : G, K)
\end{aligned}
$$

$\square$

Note that this translation of types, variables and λ-abstractions coincides with that of [20]. Evidently, the case of generalized application is new since it was not considered there. Only here is the need for a garbage increment.

Finally, let us observe that the extra double negation in $(A \supset B)^*$ has to be integrated as $\neg\overline{B} \supset \neg\overline{A}$, and not as $\overline{A} \supset \neg\neg\overline{B}$. Had the latter alternative been adopted, and again, already for CPS, we would lose weak simulation. The CPS would then be defined by:

$$
\begin{aligned}
(\lambda x.t : K) &= K(\lambda xw.w\overline{t}) \\
(u :: l : K) &= \lambda w.w(\lambda m.m\overline{u}(l : K)) \\
(t(u :: l) : K) &= (t : \lambda m.m\overline{u}(l : K))
\end{aligned}
$$

With these definitions, one calculates:

$$
\begin{aligned}
((\lambda x.t)(u :: []) : K) &= (\lambda m.m\overline{u}(\lambda w.wK))(\lambda xw.w\overline{t}) \\
&\to_\beta (\lambda xw.w\overline{t})\overline{u}(\lambda w.wK) \\
&=_\beta (\lambda x.(\lambda w.w\overline{t})(\lambda w.wK))\overline{u} \\
&\to_\beta^3 (\lambda x.(t : K))\overline{u} \\
&= (u(x)(t[]) : K)
\end{aligned}
$$

Again, the undirected $=_\beta$-step cannot be dispensed with by reduction steps in λ-calculus.

5. Higher-Order Systems

In this section, we extend the CGPS translation to, and obtain strong normalisation for, the extensions of $\lambda\mathbf{J^{m}}^{se}$ described in the following table:

intuitionistic logic	sequent calculus	natural deduction system
propositional	$\lambda\mathbf{J^{m}}^{se}$	$\underline{\lambda}$
second-order propositional	$\lambda 2\mathbf{J^{m}}^{se}$	$\underline{\lambda}2$
higher-order propositional	$\lambda\omega\mathbf{J^{m}}^{se}$	$\underline{\lambda}\omega$
higher-order predicate	$\lambda H\mathbf{J^{m}}^{se}$	$\underline{\lambda}H$

Such extensions constitute several systems of intuitionistic logic formulated as sequent calculi, and have a corresponding natural deduction system. The latter are formulated as domain-free type theories [3], and all but one belong to the domain-free cube. The only exception is $\underline{\lambda}H$, which is a domain-free formulation of Geuvers' treatment of higher-order intuitionistic logic [14].

Each CGPS translation goes from a sequent calculus to the corresponding natural deduction system, where the latter is expected to satisfy strong normalisation. This is the case for the systems in the domain-free cube [3]. As to $\underline{\lambda}H$, it is well known that it is a pure type system [14] which, in addition, has a functional specification [3]. Now *op. cit.* shows that in such cases, strong normalisation of the domain-full system implies the same property for the domain-free one. Therefore, we infer from the strong normalisation of Geuvers' system that of $\underline{\lambda}H$.

The formulation of the systems of higher order (unlike those at second order) require the introduction of an upper level of domains of quantification and their inhabitants. In order to avoid that these technicalities blur the simplicity by which the properties of the CGPS extend beyond the (zero-th order) propositional case, we decided to develop first the second-order case with the simplest formulation, even at the price of a little amount of redundancy.

5.1. **Second Order.** All the results of the previous sections readily extend to the second order which is one of the important advantages of double-negation translations w. r. t. Gödel's negative translation (employed for first-order $\lambda\mu$-calculus by Parigot [31]). In order to give an idea of how to proceed, we will sketch how to equip $\lambda\mathbf{J^{m}}^{se}$ by a second-order universal quantifier (yielding system $\lambda 2\mathbf{J^{m}}^{se}$) and how to extend the CGPS translation of $\lambda\mathbf{J^{m}}^{se}$ into simply-typed λ-calculus to a translation of $\lambda 2\mathbf{J^{m}}^{se}$ into a "domain-free" version $\underline{\lambda}2$ [3] of second-order λ-calculus a. k. a. system F [15].

5.1.1. *System $\underline{\lambda}2$.* To recall, system F corresponds to second-order propositional logic and consequently also has the types of the form $\forall X.A$. Therefore, also on the type level, we need to allow silent renaming of bound variables. Just as it is done in [20], we stay with the Curry-style typing of our previous systems but nevertheless add $\Lambda X.t$ and tA to the term syntax for λ, for universal introduction and universal elimination, respectively. These two constructions normally belong to the typing discipline à la Church, but in addition to λ, they give (a variant of) system $\underline{\lambda}2$ of [3]. The new typing rules are:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \Lambda X.t : \forall X.A} \; RIntro2 \qquad \frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash tB : [B/X]A}$$

with $[B/X]A$ denoting type substitution, where $RIntro2$ is under the proviso that X is not free in any type in Γ. The new reduction rule is $\beta 2$:

$$(\Lambda X.t)B \to [B/X]t$$

with $[B/X]T$ type substitution in term t. It is shown in [3] that strong normalisation of typable terms is inherited from the same property for system F, that has been established by Tait's refinement [37] of Girard's weak normalisation proof [15].

5.1.2. *System $\lambda 2\mathbf{J}^{mse}$.* For $\lambda 2\mathbf{J}^{mse}$, we also extend the term syntax by $\Lambda X.t$ and extend the co-term syntax by $A :: l$ that count among the evaluation contexts. The cases $u :: l$ and $A :: l$ can be uniformly seen as $U :: l$, where U now stands for a term or type. Type substitution $[B/X]T$ for T a term/co-term/command can be defined in the obvious way. $\lambda 2\mathbf{J}^{mse}$ extends $\lambda \mathbf{J}^{mse}$ also by the rule $RIntro2$ above and by

$$\frac{\Gamma | l : [B/X]A \vdash C}{\Gamma | B :: l : \forall X.A \vdash C} \; LIntro2$$

The notion $l@l'$ is redefined with u replaced by U (and stays associative), and the admissible typing rules for substitution, weakening and @ carry over from $\lambda \mathbf{J}^{mse}$, as well as the obvious typing rules for type substitution. The only new reduction rule is

$$(\beta 2) \quad (\Lambda X.t)(B :: l) \to ([B/X]t)l$$

So, term substitution is dealt with in an explicit way in $\lambda 2\mathbf{J}^{mse}$, but type substitution is still left implicit. This gap would be annoying for dependently-typed systems, see [24] for a proposal that solves this problem.

The one-step reduction relation takes into account the new syntactic constructions, and subject reduction follows.

5.1.3. *CGPS translation.* The CGPS-translation of $\lambda \mathbf{J}^{mse}$ into $\underline{\lambda}$ is now extended to a CGPS of $\lambda 2\mathbf{J}^{mse}$ into $\underline{\lambda}2$. Unlike the case of implication, no further double negation w.r.t. [20] has to be added, since our sequent calculi do not provide an explicit type substitution; we set

$$(\forall X.A)^* = \forall X.\overline{A} \ .$$

Evidently, $([B/X]A)^* = [B^*/X]A^*$, hence (but to be proven simultaneously) $\overline{[B/X]A} = [B^*/X]\overline{A}$. We extend the definition of $(T : G, K)$ for $\lambda \mathbf{J}^{mse}$, with G, K terms of $\underline{\lambda}2$, by

$$
\begin{aligned}
(\Lambda X.t : G, K) &= [K(\Lambda X.\overline{t}); G] \\
(t(B :: l) : G, K) &= (t : G, \lambda m.(l : G, K)(mB^*)) \\
(B :: l : G, K) &= \lambda w.wG(\lambda m.(l : G, K)(mB^*))
\end{aligned}
$$

The clause for $\Lambda X.t$ is taken from [20]. This extended translation obeys to the same typing as for $\lambda \mathbf{J}^{mse}$ (now always w.r.t. $\underline{\lambda}2$), hence satisfies type soundness.

Lemma 5.1. *The CGPS translation of $\lambda 2\mathbf{J}^{mse}$ into $\underline{\lambda}2$ satisfies the following:*

(1) $-$ (7) *as in Lemma 4.2.*

(8) $[B^*/X](T : G, K) = ([B/X]T : [B^*/X]G, [B^*/X]K)$ *and* $[B^*/X]\overline{t} = \overline{[B/X]t}$.

(9) $[B/X](T : G, K) = (T : [B/X]G, [B/X]K)$ *for X not free in T.*

(10) (a) $(tl : \mathsf{s}(G), \lambda m.(l' : G, K)(mB^*)) \to_\beta^+ (t(l@(B :: l')) : G, K)$

   (b) $(l : \mathsf{s}(G), \lambda m.(l' : G, K)(mB^*)) \to_\beta^+ (l@(B :: l') : G, K)$

Now we prove the following theorem just as before Theorem 4.3.

Theorem 5.2 (Simulation). *If $t \to u$ in $\lambda 2\mathbf{J}^{\mathbf{m}se}$, then $\bar{t} \to_\beta^+ \bar{u}$ in $\underline{\lambda}2$.*

Proof. We show the new base cases.

Case $\beta 2$: $(\Lambda X.t)(B :: l) \to ([B/X]t)l$.

$$
\begin{aligned}
& ((\Lambda X.t)(B :: l) : G, K) \\
= \; & (\Lambda X.t : G, \lambda m.(l : G, K)(mB^*)) \\
= \; & [(\lambda m.(l : G, K)(mB^*))(\Lambda X.\bar{t}); G] \\
\to_\beta^3 \; & (l : G, K)[B^*/X]\bar{t} \\
= \; & (l : G, K)\overline{[B/X]t} \qquad\qquad \text{(Lemma 5.1.8.)} \\
\to_\beta^* \; & (([B/X]t)l : G, K) \qquad\quad\; \text{(Lemma 5.1.5.)}
\end{aligned}
$$

Case π: Sub-case $E = B :: l'$.

$$
\begin{aligned}
(\{tl\}(B :: l') : G, K) \quad = \quad & (tl : \mathsf{s}(G), \lambda m.(l' : G, K)(mB^*)) \\
\to_\beta^+ \quad & (t\,(l @ (B :: l')) : G, K) \qquad \text{(Lemma 5.1.10.)} \qquad \square
\end{aligned}
$$

Corollary 5.3. *The typable terms of $\lambda 2\mathbf{J}^{\mathbf{m}se}$ are strongly normalising.*

A technically more involved CGPS for the Church-style version of $\lambda 2\mathbf{J}^{\mathbf{m}se}$ into Church-style system F can be given along the lines of [26], where the colon translation has to be made relative to a context Γ.

5.2. F^ω and Higher-Order Logic.
In the second order systems, one assumes that X in the quantification $\forall X.A$ ranges over the domain $PROP$ of all propositions (or types). In this subsection we study systems allowing the formation of other *domains of quantification*, usually denoted $\mathcal{D}, \mathcal{E}$. Quantification now has the form $\forall X : \mathcal{D}.A$, but, at the proof-term level, abstraction $\Lambda X.t$ remains domain-free.

In the following we formulate intuitionistic higher-order predicate logic, both in the natural deduction format $\underline{\lambda}H$, and the sequent calculus format $\lambda H\mathbf{J}^{\mathbf{m}se}$. A minor restriction in each of these systems gives two formulations ($\underline{\lambda}\omega$ and $\lambda\omega\mathbf{J}^{\mathbf{m}se}$, respectively) of intuitionistic higher-order propositional logic, or system F^ω.

5.2.1. *Domains of quantification.* Domains (of quantification) are given by:

$$\mathcal{D}, \mathcal{E} \quad ::= \quad PROP \,|\, \mathsf{X} \,|\, \mathcal{D} \to \mathcal{D}$$

X ranges over a set of *domain variables*. These play the role of "sorts" in multi-sorted first-order logic. The set of domains is very much like Church's structure of simple types, except that, besides $PROP$ (the type of propositions), Church only admitted one other base type ι of *individuals*.

Next come the propositional, or type, or individual, function(al)s:

$$A, B, C \quad ::= \quad X \,|\, \lambda X.A \,|\, AB \,|\, A \supset B \,|\, \forall X : \mathcal{D}.A$$

These are the inhabitants of domains. X ranges over a set, whose elements may be seen as *type variables*, or *propositional variables*, or *individual variables*, etc. In the last case a meta-variable like x would be more expressive. Also, one may employ meta-variables φ and ψ instead of A, if one wants to emphasize that the inhabitant is a proposition, or t if one wants to emphasize that the inhabitant lives in some domain of individuals X. $\lambda X.A$

Figure 14: Domain assignment rules for higher-order logic

$$\frac{(X : \mathcal{D}) \in \Delta}{\Delta \vdash X : \mathcal{D}} \ Ax$$

$$\frac{\Delta, X : \mathcal{D} \vdash A : \mathcal{E}}{\Delta \vdash \lambda X.A : \mathcal{D} \to \mathcal{E}} \ I \to \qquad \frac{\Delta \vdash A : \mathcal{D} \to \mathcal{E} \quad \Delta \vdash B : \mathcal{D}}{\Delta \vdash AB : \mathcal{E}} \ E \to$$

$$\frac{\Delta \vdash A : PROP \quad \Delta \vdash B : PROP}{\Delta \vdash A \supset B : PROP} \ F \supset \qquad \frac{\Delta, X : \mathcal{D} \vdash A : PROP}{\Delta \vdash \forall X : \mathcal{D}.A : PROP} \ F\forall$$

and AB are the generic, and usual, mechanism for building inhabitants at all levels of the domain structure.[9]

The relationship between domains and their inhabitants is governed by *domain assignment rules*. Let Δ range over consistent sets of declarations $X : \mathcal{D}$. Such rules derive sequents of the form $\Delta \vdash A : \mathcal{D}$, as described in Figure 14. Besides the ordinary rules of the simply-typed λ-calculus, one has two *formation* rules. If $\Delta \vdash A : PROP$, then we say that A is a Δ-*proposition*, or just *proposition*. Alternative terminology is "formula" or "type".

Finally, the inhabitants of domains may reduce according to the following reduction rule:

$$(\beta_0) \quad (\lambda X.A)B \quad \to \quad [B/X]A \ .$$

The given definition of domains, their inhabitants, and the derivable sequents $\Delta \vdash A : \mathcal{D}$ remains fixed for the rest of this subsection (that is, in all the systems $\underline{\lambda}\omega$, $\underline{\lambda}H$, $\lambda\omega\mathbf{J}^{\mathbf{m}se}$, and $\lambda H\mathbf{J}^{\mathbf{m}se}$), except for one thing: in $\underline{\lambda}\omega$ and $\lambda\omega\mathbf{J}^{\mathbf{m}se}$, domain variables X are not allowed.

5.2.2. *Systems $\underline{\lambda}\omega$ and $\underline{\lambda}H$.* We now define the natural deduction system $\underline{\lambda}H$ and its minor variant $\underline{\lambda}\omega$. Specifically, we define proof expressions and their "typing" rules, that is, the rules governing what expressions inhabit what propositions. At this level, the systems $\underline{\lambda}\omega$ and $\underline{\lambda}H$ are indistinguishable; indeed, the single difference is the one already pointed out at the domains level.

In addition, at this level, also $\underline{\lambda}H$ and $\underline{\lambda}2$ would be indistinguishable, provided that (i) we had defined $\underline{\lambda}2$ with a trivial domain level $\mathcal{D} = PROP$, and with formal "domain assignment rules" generating the types/ propositions; (ii) we wrote $\forall X : \mathcal{D}.A$ and not just $\forall X.A$; (iii) sequents $\Gamma \vdash t : A$ carried an outer set Δ declaring necessary variables X with domain $PROP$. So, what follows may be used as a recapitulation of $\underline{\lambda}2$.

In $\underline{\lambda}H$ one has the following *proof terms*:

$$t, u, v \quad ::= \quad x \,|\, \lambda x.t \,|\, tu \,|\, \Lambda X.t \,|\, tA$$

Proof terms are assigned to propositions through *proposition assignment rules*, which generate sequents of the form $\Delta; \Gamma \vdash t : A$ according to the rules of Figure 15. Here Γ is a consistent set of declarations $x : A$; in addition we expect $\Delta \vdash \Gamma : PROP$ and

---

[9]Notation: different forms of abstraction are denoted by variants of the symbol λ, but application is always denoted by juxtaposition.

Figure 15: Proposition assignment rules of $\underline{\lambda}H$

$$\frac{\Delta \vdash \Gamma : PROP \quad (x : A) \in \Gamma}{\Delta; \Gamma \vdash x : A} \ Ax$$

$$\frac{\Delta; \Gamma \vdash t : A \supset B \quad \Delta; \Gamma \vdash u : A}{\Delta; \Gamma \vdash tu : B} \ E \supset \qquad \frac{\Delta; \Gamma, x : A \vdash t : B}{\Delta; \Gamma \vdash \lambda x.t : A \supset B} \ I \supset$$

$$\frac{\Delta; \Gamma \vdash t : \forall X : \mathcal{D}.A \quad \Delta \vdash B : \mathcal{D}}{\Delta; \Gamma \vdash tB : [B/X]A} \ E\forall \qquad \frac{\Delta, X : \mathcal{D}; \Gamma \vdash t : A}{\Delta; \Gamma \vdash \Lambda X.t : \forall X : \mathcal{D}.A} \ I\forall$$

$$\frac{\Delta; \Gamma \vdash t : A \quad \Delta \vdash B : PROP \quad A =_{\beta_0} B}{\Delta; \Gamma \vdash t : B} \ Conv.$$

$\Delta \vdash A : PROP$, whenever $\Delta; \Gamma \vdash t : A$ is generated. The notation $\Delta \vdash \Gamma : PROP$ means $(x : A) \in \Gamma \Rightarrow \Delta \vdash A : PROP$. Proof terms reduce according to these two reduction rules:

$$\begin{array}{rcl}
(\beta_1) & (\lambda x.t)u & \rightarrow & [u/x]t \\
(\beta_2) & (\Lambda X.t)B & \rightarrow & [B/X]t
\end{array}$$

Proof terms are capable of β_0-reduction, via the closure rule $B \rightarrow_{\beta_0} B' \Longrightarrow tB \rightarrow_{\beta_0} tB'$.

5.2.3. *Systems* $\lambda\omega\mathbf{J}^{\mathbf{m}se}$ *and* $\lambda H\mathbf{J}^{\mathbf{m}se}$. We now define the sequent calculi $\lambda H\mathbf{J}^{\mathbf{m}se}$ and its minor variant $\lambda\omega\mathbf{J}^{\mathbf{m}se}$. Again, at the level of proof expressions and their "typing" rules, the systems $\lambda\omega\mathbf{J}^{\mathbf{m}se}$ and $\lambda H\mathbf{J}^{\mathbf{m}se}$ are indistinguishable; indeed, the single difference is the one already pointed out at the domains level.

In addition, at this level, also $\lambda H\mathbf{J}^{\mathbf{m}se}$ and $\lambda 2\mathbf{J}^{\mathbf{m}se}$ would be indistinguishable, under the same provisos as before for the indistinguishability of $\underline{\lambda}H$ and $\underline{\lambda}2$. Hence, also the following definition is mostly a recapitulation of $\lambda 2\mathbf{J}^{\mathbf{m}se}$.

In $\lambda H\mathbf{J}^{\mathbf{m}se}$ one has the following *proof expressions*:

$$\begin{array}{llll}
\text{(Proof terms)} & t, u, v & ::= & x \,|\, \lambda x.t \,|\, \Lambda X.t \,|\, \{c\} \\
\text{(Proof co-terms)} & l & ::= & [\,] \,|\, u :: l \,|\, B :: l \,|\, (x)c \\
\text{(Proof commands)} & c & ::= & tl
\end{array}$$

Proposition assignment rules generate sequents of the forms $\Delta; \Gamma \vdash t : A$, and $\Delta; \Gamma | l : B \vdash A$, and $\Delta; \Gamma \xrightarrow{c} B$. In these sequents we expect $\Delta \vdash \Gamma : PROP$, $\Delta \vdash A : PROP$, and $\Delta \vdash B : PROP$. The rules are shown in Figure 16.

The rules for the reduction of proof expressions are:

$$\begin{array}{rcll}
(\beta_1) & (\lambda x.t)(u :: l) & \rightarrow & u((x)tl) \\
(\beta_2) & (\Lambda X.t)(B :: l) & \rightarrow & ([B/X]t)l \\
(\pi) & \{tl\}E & \rightarrow & t\,(l@E) \\
(\sigma) & t(x)c & \rightarrow & [t/x]c \\
(\mu) & (x)xl & \rightarrow & l, \text{ if } x \notin l \\
(\epsilon) & \{t[\,]\} & \rightarrow & t
\end{array}$$

Figure 16: Proposition assignment rules of $\lambda H\mathbf{J}^{\mathbf{m}se}$

$$\frac{\Delta \vdash \Gamma : PROP \quad \Delta \vdash A : PROP}{\Delta; \Gamma|[] : A \vdash A} \; LAx \qquad \frac{\Delta \vdash \Gamma : PROP \quad (x : A) \in \Gamma}{\Delta; \Gamma \vdash x : A} \; RAx$$

$$\frac{\Delta; \Gamma \vdash u : A \quad \Delta; \Gamma|l : B \vdash C}{\Delta; \Gamma|u :: l : A \supset B \vdash C} \; L\supset \qquad \frac{\Delta; \Gamma, x : A \vdash t : B}{\Delta; \Gamma \vdash \lambda x.t : A \supset B} \; R\supset$$

$$\frac{\Delta \vdash B : \mathcal{D} \quad \Delta; \Gamma|l : [B/X]A \vdash C}{\Delta; \Gamma|B :: l : \forall X : \mathcal{D}.A \vdash C} \; L\forall \qquad \frac{\Delta, X : \mathcal{D}; \Gamma \vdash t : A}{\Delta; \Gamma \vdash \Lambda X.t : \forall X : \mathcal{D}.A} \; R\forall(X \notin \Gamma)$$

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

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

$$\frac{\Delta; \Gamma \vdash t : A \quad \Delta \vdash B : PROP \quad A =_{\beta_0} B}{\Delta; \Gamma \vdash t : B} \; Conv.$$

Proof expressions are capable of β_0-reduction, via the closure rule $B \to_{\beta_0} B' \implies B :: l \to_{\beta_0} B' :: l$.

5.2.4. *CGPS translations.* We will see that, when the CGPS translation is extended to $\lambda\omega\mathbf{J}^{\mathbf{m}se}$ and $\lambda H\mathbf{J}^{\mathbf{m}se}$, its properties (type soundness and the simulation theorem) remain valid and are proved almost *verbatim* relative to the second-order case. Here is an explanation. The proofs of the properties of the CGPS translation have two components. The first component is a proof that the CGPS translation behaves well relative to domain inhabitants/assignment. This comprises (i) domain soundness (Lemma 5.4 below); (ii) commutation with substitution of type variables X; (iii) simulation of β_0 (Lemma 5.5). This component depends on the domain inhabitants/assignment and inhabitants reduction (β_0) of the system where the translation is defined. Very little variation exists between $\lambda 2\mathbf{J}^{\mathbf{m}se}$, $\lambda\omega\mathbf{J}^{\mathbf{m}se}$, and $\lambda H\mathbf{J}^{\mathbf{m}se}$ regarding these aspects, the only singularity being that there is no β_0 at second order. The second component is the proper proofs of type soundness and strict simulation, which are all the same for $\lambda 2\mathbf{J}^{\mathbf{m}se}$, $\lambda\omega\mathbf{J}^{\mathbf{m}se}$, and $\lambda H\mathbf{J}^{\mathbf{m}se}$, except for one inductive case of the strict simulation theorem, absent in $\lambda 2\mathbf{J}^{\mathbf{m}se}$, and relative to β_0 reduction at proof-expression level.

We define a CGPS translation from $\lambda H\mathbf{J}^{\mathbf{m}se}$ to $\underline{\lambda}H$. It can be seen as a CGPS translation from $\lambda\omega\mathbf{J}^{\mathbf{m}se}$ to $\underline{\lambda}\omega$ as well, and generalises only slightly the previous CGPS translation from $\lambda 2\mathbf{J}^{\mathbf{m}se}$ to $\underline{\lambda}2$, by providing translations for $\lambda X.A$ and AB.

Domains remain fixed, but their inhabitants are translated as in Figure 17. Recall that the relation of domain assignment of $\lambda H\mathbf{J}^{\mathbf{m}se}$ is the same as that of $\underline{\lambda}H$. Such relation is intended in the following result.

Figure 17: Translation of propositional/individual function(al)s

$$
\begin{aligned}
X^* &= X \\
(A \supset B)^* &= \neg \overline{B} \supset \neg \overline{A} \\
(\forall X : \mathcal{D}.A)^* &= \forall X : \mathcal{D}.\overline{A} \\
(\lambda\!\!\lambda X.A)^* &= \lambda\!\!\lambda X.A^* \\
(AB)^* &= A^* B^*
\end{aligned}
$$

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

Lemma 5.4 (Domain soundness). *The following holds:*

$$
\frac{\Delta \vdash A : \mathcal{D}}{\Delta \vdash A^* : \mathcal{D}} \qquad \frac{\Delta \vdash A : PROP}{\Delta \vdash \overline{A} : PROP}
$$

Proof. By simultaneous induction on A. $\qquad\square$

Recall also that the relation of domain assignment of $\lambda\omega \mathbf{J}^{\mathbf{m}se}$ is the same as that of $\underline{\lambda}\omega$. If this latter relation is intended, the previous result also holds, with the same proof. The previous lemma generalises the fact that, at second order, if A is a proposition (type), then so is A^* and $\overline{A}$.

The same grammar generates the sets of proof expressions of $\lambda H \mathbf{J}^{\mathbf{m}se}$, $\lambda\omega \mathbf{J}^{\mathbf{m}se}$, and $\lambda 2 \mathbf{J}^{\mathbf{m}se}$; another single grammar generates the sets of proof expressions of $\underline{\lambda}H$, $\underline{\lambda}\omega$, and $\underline{\lambda}2$. These two grammars are already known from the second-order systems, so the CGPS translation at the level of proof expressions is known and we do not repeat it.

The equations $([B/X]A)^* = [B^*/X]A^*$ and $\overline{[B/X]A} = [B^*/X]\overline{A}$ still hold, and are proved by the same simultaneous induction, supplemented with the straightforward new cases $\lambda\!\!\lambda X.A$ and AB.

Lemma 5.5. *If $A \to_{\beta_0} B$ in $\lambda H \mathbf{J}^{\mathbf{m}se}$ (resp. $\lambda\omega \mathbf{J}^{\mathbf{m}se}$), then $A^* \to_{\beta_0} B^*$ and $\overline{A} \to_{\beta_0} \overline{B}$ in $\underline{\lambda}H$ (resp. $\underline{\lambda}\omega$).*

Proof. Straightforward induction on $A \to_{\beta_0} B$. The base case follows from $([B/X]A)^* = [B^*/X]A^*$. The inductive cases are routine. $\qquad\square$

Then one obtains the admissible typing rules of Figure 18. This is the same typing obeyed by the CGPS translation from $\lambda 2 \mathbf{J}^{\mathbf{m}se}$ to $\underline{\lambda}2$, provided, as remarked before, $\lambda 2 \mathbf{J}^{\mathbf{m}se}$ and $\underline{\lambda}2$ are defined with a formal level of domains, etc.

Lemma 5.6. *The CGPS translations of $\lambda H \mathbf{J}^{\mathbf{m}se}$ into $\underline{\lambda}H$, and of $\lambda\omega \mathbf{J}^{\mathbf{m}se}$ into $\underline{\lambda}\omega$, satisfy the items (1) to (10) of Lemma 5.1.*

Theorem 5.7 (Simulation). *If $t \to u$ in $\lambda H \mathbf{J}^{\mathbf{m}se}$ (resp. $\lambda\omega \mathbf{J}^{\mathbf{m}se}$), then $\overline{t} \to_\beta^+ \overline{u}$ in $\underline{\lambda}H$ (resp. $\underline{\lambda}\omega$).*

Proof. The same proof as in the second-order case applies. There is only one new inductive case, to prove $(B :: l : G, K) \to_\beta^+ (B' :: l : G, K)$, when $B \to_{\beta_0} B'$, a case which is an immediate consequence of Lemma 5.5 and the definition of the CGPS translation. $\qquad\square$

Figure 18: Admissible proposition assignment rules for CGPS translation of $\lambda H \mathbf{J^{m}}^{se}$

$$\frac{\Delta; \Gamma \vdash t : A}{\Delta; \overline{\Gamma} \vdash \overline{t} : \overline{A}}$$

$$\frac{\Delta; \Gamma \vdash t : A \quad \Delta; \overline{\Gamma}, \Gamma' \vdash G : \top \quad \Delta; \overline{\Gamma}, \Gamma' \vdash K : \neg A^*}{\Delta; \overline{\Gamma}, \Gamma' \vdash (t : G, K) : \bot}$$

$$\frac{\Delta; \Gamma | l : A \vdash B \quad \Delta; \overline{\Gamma}, \Gamma' \vdash G : \top \quad \Delta; \overline{\Gamma}, \Gamma' \vdash K : \neg B^*}{\Delta; \overline{\Gamma}, \Gamma' \vdash (l : G, K) : \neg \overline{A}}$$

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

Corollary 5.8. *The typable terms of $\lambda H \mathbf{J^{m}}^{se}$ and $\lambda \omega \mathbf{J^{m}}^{se}$ are strongly normalising.*

6. Further remarks

Contributions. This article provides reduction-preserving CGPS translations of $\lambda \mathbf{J^{m}}^{se}$ 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 $\lambda \mathbf{J^{m}}$ and the intuitionistic fragment of $\overline{\lambda}\mu\tilde{\mu}$ are detailed, and confluence for them obtained. It is shown that all the results smoothly extend to systems with quantification over propositions and even functionals over propositions and (many-sorted) individuals. In all cases, the sequent-calculus format is embedded into the natural-deduction style.

C(G)PS and strong normalisation. In the literature one finds strong normalisation proofs for sequent calculi [7, 8, 23, 24, 33, 38], but not by means of CPS translations; or CPS translations for natural deduction systems [1, 2, 6, 17, 20, 30].

This article provides, in particular, a reduction-preserving CGPS translation for the lambda-calculus with generalised applications $\lambda \mathbf{J}$. [30] covers full propositional classical logic with general elimination rules and its intuitionistic implicational fragment corresponds to $\lambda \mathbf{J}$. However, [30] does not prove a strict simulation by CPS (permutative conversions are collapsed), so an auxiliary argument in the style of de Groote [6], involving a proof in isolation of SN for permutative conversions, is used.

In Curien and Herbelin's work [5, 18] 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 [19]; $\overline{(\_)}$ generalises Plotkin's call-by-name CPS translation [32]. (ii) $(\_)^n$ does not employ the colon operator; $\overline{(\_)}$ does employ (we suspect that doing administrative reductions at compile time is necessary to achieve strict 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 $\overline{(\_)}$ strictly simulates reduction and thus achieves a proof of strong normalisation.

Higher-order sequent calculi. Our formulations of system F^ω and higher-order logic in the sequent calculus format were helpful for showing the wide applicability of the CGPS technique. Nevertheless, they are another experience in the formulation of type theories as sequent calculi [24]. We adopted the guideline that only proof-expression could suffer a change in the proof-theoretical format, but other, more "uniform", possibilities exist, where also the domain assignment relation is changed to the sequent calculus format. An improvement, in view of proof-search, is to restrict the conversion rule of the typing system to an expansion rule [36]. Finally, in $\lambda H\mathbf{J}^{\mathbf{m}se}$, $\lambda\omega\mathbf{J}^{\mathbf{m}se}$, and $\lambda 2\mathbf{J}^{\mathbf{m}se}$ we re-encounter explicit substitutions in higher-order type theories [4, 28], but with a simpler treatment (no explicit execution) and in a simpler setting (no dependent types).

Future work. 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 $\overline{\lambda}\mu\tilde{\mu}$, 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 [20] (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.

References

[1] Gilles Barthe, John Hatcliff, and Morten Heine Sørensen. A notion of classical pure type system (preliminary version). In Stephen Brookes and Michael Mislove, editors, *Proceedings of the Thirteenth Conference on the Mathematical Foundations of Programming Semantics*, volume 6 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1997. 56 pp.

[2] Gilles Barthe, John Hatcliff, and Morten Heine Sørensen. CPS translations and applications: The cube and beyond. *Higher-Order and Symbolic Computation*, 12(2):125–170, 1999.

[3] Gilles Barthe and Morten Heine Sørensen. Domain-free pure type systems. *Journal of Functional Programming*, 10(5):417–452, September 2000.

[4] Roel Bloo. Pure type systems with explicit substitution. *Mathematical Structures in Computer Science*, 11(1):3–19, 2001.

[5] P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of the fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montréal*, pages 233–243. IEEE, 2000.

[6] Philippe de Groote. On the strong normalisation of intuitionistic natural deduction with permutation-conversions. *Information and Computation*, 178:441–464, 2002.

[7] A. Dragalin. *Mathematical Intuitionism.*, volume 67 of *Translations of Mathematical Monographs*. American Mathematical Society, 1988.

[8] 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.

[9] J. Espírito Santo. Completing Herbelin's programme. In Simona Ronchi della Rocca, editor, *Proceedings of TLCA '07*, volume 4583 of *Lecture Notes in Computer Science*, pages 118–132. Springer Verlag, 2007.

[10] J. Espírito Santo and Luís 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.

[11] J. Espírito Santo and Luís Pinto. Confluence and strong normalisation of the generalised multiary λ-calculus. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Revised selected papers from the International Workshop TYPES 2003*, volume 3085 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[12] José Espírito Santo, Ralph Matthes, and Luís Pinto. Continuation-passing style and strong normalisation for intuitionistic sequent calculi. In Simona Ronchi della Rocca, editor, *Proceedings of TLCA '07*, volume 4583 of *Lecture Notes in Computer Science*, pages 133–147. Springer Verlag, 2007.

[13] 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.

[14] J. Herman Geuvers. The calculus of constructions and higher order logic. In Ph. de Groote, editor, *The Curry-Howard isomorphism*, volume 8 of *Cahiers du Centre de logique (Université catholique de Louvain), Academia, Louvain-la-Neuve (Belgium)*, pages 139–191, 1995.

[15] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse de Doctorat d'État, Université de Paris VII, 1972.

[16] T. Griffin. A formulae-as-types notion of control. In *ACM Conf. Principles of Programming Languages*, pages 47–58. ACM Press, 1990.

[17] Robert Harper and Mark Lillibridge. Operational interpretations of an extension of F_ω with control operators. *J. Funct. Program.*, 6(3):393–417, 1996.

[18] H. Herbelin. C'est maintenant qu'on calcule, 2005. Habilitation Thesis, Paris XI.

[19] Martin Hofmann and Thomas Streicher. Completeness of computation models for λ_μ-calculus. *Information and Computation*, 179:332–355, 2002.

[20] Satoshi Ikeda and Koji Nakazawa. Strong normalization proofs by CPS-translations. *Information Processing Letters*, 99:163–170, 2006.

[21] F. Joachimski and R. Matthes. Standardization and confluence for a lambda-calculus with generalized applications. In Leo Bachmair, editor, *Proc. of Int. Conference on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, 2000.

[22] Felix Joachimski and Ralph 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.

[23] 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.

[24] Stéphane Lengrand, Roy Dyckhoff, and James McKinna. A sequent calculus for type theory. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 441–455. Springer Verlag, 2006.

[25] Silvia Likavec. *Types for object-oriented and functional programming languages*. PhD thesis, Università di Torino, Italy, ENS Lyon, France, February 2005.

[26] Ralph Matthes. Stabilization—an alternative to double-negation translation for classical natural deduction. In Viggo Stoltenberg-Hansen and Jouko Väänänen, editors, *Proceedings of the Logic Colloquium 2003*, volume 24 of *Lecture Notes in Logic*, pages 167–199. A K Peters, 2006.

[27] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

[28] César Muñoz. Dependent types and explicit substitutions: a meta-theoretical development. *Mathematical Structures in Computer Science*, 11(1):91–129, 2001.

[29] Koji Nakazawa and Makoto 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.

[30] Koji Nakazawa and Makoto Tatsuta. Strong normalization of classical natural deduction with disjunctions. *Annals of Pure and Applied Logic*, 153:21–37, 2008.

[31] Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997.

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

[33] E. Polonovski. Strong normalization of $\bar{\lambda}\mu\tilde{\mu}$-calculus with explicit substitutions. In Igor Walukiewicz, editor, *Proc. of 7th Int. Conference on Foundations of Software Sciences and Computation Structures (FoSSaCS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 423–437. Springer-Verlag, 2004.

[34] A. Sabry and P. Wadler. A reflection on call-by-value. In *Proc. of ACM SIGPLAN Int. Conference on Functional Programming ICFP 1996*, pages 13–24. ACM Press, 1996.

[35] H. Schwichtenberg. Termination of permutative conversions in intuitionistic Gentzen calculi. *Theoretical Computer Science*, 212(1–2):247–260, 1999.

[36] Jonathan P. Seldin. A Gentzen-style sequent calculus of constructions with expansion rules. *Theor. Comput. Sci.*, 243(1-2):199–215, 2000.

[37] William W. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium Boston 1971/72*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer Verlag, 1975.

[38] C. Urban and G. Bierman. Strong normalisation of cut-elimination in classical logic. In Jean-Yves Girard, editor, *Proceedings of TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*, pages 365–380. Springer-Verlag, 1999.