Permutative Conversions in Intuitionistic Multiary Sequent Calculi with Cuts

José Espírito Santo and Luís Pinto*

Departamento de Matemática, Universidade do Minho
4710-057 Braga, Portugal
{jes,luis}@math.uminho.pt

Abstract. This work presents an extension with cuts of Schwichtenberg's multiary sequent calculus. We identify a set of permutative conversions on it, prove their termination and confluence and establish the permutability theorem. We present our sequent calculus as the typing system of the *generalised multiary λ-calculus* $\lambda\mathbf{J}^{\mathbf{m}}$, a new calculus introduced in this work. $\lambda\mathbf{J}^{\mathbf{m}}$ corresponds to an extension of λ-calculus with a notion of *generalised multiary application*, which may be seen as a function applied to a list of arguments and then explicitly substituted in another term. Proof-theoretically the corresponding typing rule encompasses, in a modular way, generalised eliminations of von Plato and Herbelin's head cuts.

1 Introduction

It is well-known that two intuitionistic sequent calculus derivations determine the same natural deduction proof when they are inter-permutable [9, 6]. In [1] this idea is made precise for cut-free sequent calculus by the identification of a basic set of permutations and the definition of a confluent and weakly normalising rewriting system whose normal forms are the normal natural deductions. Schwichtenberg proved in [7] that a variant of this rewriting system is strongly normalising.

In this work we obtain similar results for an extension with cuts of the system in [7]. Specifically: (i) we define the system $\lambda\mathbf{J}^{\mathbf{m}}$ and permutative conversions on it; (ii) we prove confluence and strong normalisation of the corresponding rewriting system; (iii) we introduce an interpretation ϕ of $\lambda\mathbf{J}^{\mathbf{m}}$ into a notational variant of the λ-calculus; (iv) we prove the permutability theorem establishing that two $\lambda\mathbf{J}^{\mathbf{m}}$-terms have the same interpretation iff they are inter-permutable.

Schwichtenberg introduces in [7] the idea of *multiary left rule*, a generalisation of the ordinary binary left rule for implication, by which in a single inference one may introduce $A_1 \supset \ldots \supset A_k \supset B$, for some $k \geq 1$. Schwichtenberg also introduces the notion *multiary sequent terms*, used in [7] as a convenient tool to represent derivations, in order to obtain termination results.

* Both authors are supported by FCT through the Centro de Matemática da Universidade do Minho, Braga, Portugal

Here we take the view that the notion of multiary sequent terms is of interest on its own, having in mind the computational interpretation of sequent calculus. Indeed, $\lambda\mathbf{J^m}$ may be seen as an extension of the λ-calculus where application is generalised in two directions: (i) "generality" in the sense of von Plato's generalised eliminations [8]; and (ii) "multiarity" here formalised in the style of Herbelin's $\overline{\lambda}$-calculus [4]. We call this new construction *generalised multiary application.* Three particular cases of this construction (*generalised application*, *multiary application* and *simple application*) determine subsystems of $\lambda\mathbf{J^m}$, which turn out to correspond exactly to three previously know calculi: (i) ΛJ [5] (denoted here as $\lambda\mathbf{J}$) —the type-theoretic counterpart to von Plato's natural deduction system with generalised eliminations; (ii) $\lambda\mathcal{P}\mathbf{h}$ [2, 3] —the multiary λ-calculus, essentially corresponding to the head-cuts subsystem of Herbelin's $\overline{\lambda}$; (iii) $\lambda\mathcal{G}$ —a sequent calculus isomorphic to natural deduction [2]. Interpretations of $\lambda\mathbf{J^m}$ onto $\lambda\mathbf{J}$ and $\lambda\mathcal{P}\mathbf{h}$, and also from these two systems onto $\lambda\mathcal{G}$, are defined and the commutative diagram in Figure 1 is obtained.

$$\begin{tikzcd}
\lambda\mathbf{J^m} \arrow[r, "\mathbf{q^m}"] \arrow[d, "\mathbf{p^m}"'] \arrow[dr, "\phi" description] & \lambda\mathbf{J} \arrow[d, "\mathbf{p}"] \\
\lambda\mathcal{P}\mathbf{h} \arrow[r, "\mathbf{q}"'] & \lambda\mathcal{G}
\end{tikzcd}$$

Fig. 1. $\lambda\mathbf{J^m}$ and its subsystems

We consider two kinds of permutative conversions: $\mathbf{p}$-permutations (inspired by [7]) and the $\mathbf{q}$-permutation (specific to this work). Permutations of the former (resp. the latter) kind deal with "generality" (resp. "multiarity") of application and, when combined together, they reduce generalised multiary applications to simple applications. Each of the mappings in Figure 1 produces the normal form w.r.t. the appropriate kind(s) of permutations: mappings $\mathbf{p}$ and $\mathbf{p^m}$ (resp. $\mathbf{q}$ and $\mathbf{q^m}$) produce $\mathbf{p}$- (resp. $\mathbf{q}$-) normal forms, whereas ϕ produces $\mathbf{p}, \mathbf{q}$-normal forms.

2 $\boldsymbol{\lambda}\mathbf{J^m}$: the generalised multiary $\boldsymbol{\lambda}$-calculus

2.1 Expressions

The generalised multiary λ-calculus $\lambda\mathbf{J^m}$ is a term calculus for intuitionistic implicational logic, corresponding to an extension with cuts of Schwichtenberg's multiary cut-free sequent calculus presented in [7]. In $\lambda\mathbf{J^m}$, *formulas* (=*types*) A, B, C, ... are built up from *propositional variables* p, q, ... using just $\supset$ (for implication) and *contexts* Γ are finite sets of *variable : formula* pairs, associating at most one formula to each variable. In the following, $\mathbf{V}$ denotes the set of variables and x, y, w, z range over $\mathbf{V}$.

Definition 1 *The terms of* $\lambda\mathbf{J^m}$ *are described in the following grammar:*

$$\begin{array}{ll} (\textit{terms of } \lambda\mathbf{J^m})\ t, u, v ::= x \mid \lambda x.t \mid t(u, l, (x)v) \\ (\textit{lists of } \lambda\mathbf{J^m}) \qquad\quad l ::= t :: l \mid [] \end{array}$$

The sets of $\lambda\mathbf{J^m}$*-terms and* $\lambda\mathbf{J^m}$*-lists are denoted by* $\mathbf{\Lambda J^m}$ *and* $\mathcal{L}\mathbf{J^m}$ *respectively. Term constructions of the form* $t(u, l, (x)v)$ *are called* generalised multiary applications*. The list* $[]$ *is called the* empty list *and lists of the form* $t :: l$ *are called* cons-lists*. In terms* $\lambda x.v$ *and* $t(u, l, (x)v)$*, occurrences of* x *in* v *are bound.*

Schwichtenberg's multiary sequent terms are obtained from $\mathbf{\Lambda J^m}$ by requiring t to be a variable in every $t(u, l, (x)v)$ and writing $y(u, l, (x)v)$ as $v_x(y, \boldsymbol{L})$, where $\boldsymbol{L}$ is the list u, l. Notice this $\boldsymbol{L}$ is non-empty as required in [7]. In Section 3 we explain in what sense $t(u, l, (x)v)$ is a generalised form of application.

2.2 Typing rules

Sequents of $\lambda\mathbf{J^m}$ are of one of the following two forms

$$\begin{array}{c} \Gamma; - \vdash t : A \\ \Gamma; B \vdash l : C, \end{array}$$

called *term sequents* and *list sequents* respectively. The distinguished position in the LHS of sequents is called the *stoup* and may either be empty (as in term sequents) or hold a formula (the case of list sequents). Read a list sequent $\Gamma; B \vdash l : C$ as "list l leads the formula B to its instance C in context Γ". C is an *instance* of B if B is of the form $B_1 \supset ... \supset B_k \supset C$, for some $k \geq 0$.

Definition 2 *The typing rules of* $\lambda\mathbf{J^m}$ *are as follows:*

$$\frac{}{x : A, \Gamma; - \vdash x : A}\ \mathit{Axiom} \qquad\qquad \frac{x : A, \Gamma; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B}\ \mathit{Right}$$

$$\frac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C \quad x : C, \Gamma; - \vdash v : D}{\Gamma; - \vdash t(u, l, (x)v) : D}\ gm - \mathit{Elim}$$

$$\frac{\Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\ \mathit{Lft} \qquad\qquad \frac{}{\Gamma; C \vdash [] : C}\ \mathit{Ax}$$

with the proviso that $x : A$ *does not belong to* Γ *in* Right *and the proviso that* $x : C$ *does not belong to* Γ *in* gm-Elim*.*

An instance of rule $gm - \mathit{Elim}$ is called a *generalised multiary elimination* (or gm-elimination, for short). We explain now why we regard these typing rules as defining a sequent calculus.

Observe that

$$\frac{y:A\supset B,\Gamma;-\vdash u:A \quad y:A\supset B,\Gamma;B\vdash l:C \quad x:C,y:A\supset B,\Gamma;-\vdash v:D}{y:A\supset B,\Gamma;-\vdash y(u,l,(x)v):D}\ m-Left \tag{1}$$

is a derived rule in $\lambda\mathbf{J^m}$, whose instances are called *multiary left* (or m-left, for short) inferences. Rule $m-Left$ may be seen as the particular case of $gm-Elim$ when the leftmost premiss is an instance of $Axiom$. Schwichtenberg's multiary sequent calculus is then obtained by requiring every gm-elimination to be a m-left inference.

In $\lambda\mathbf{J^m}$, "multiarity" is implemented with rules Lft and Ax, a device due to Herbelin [4]. Rule Lft is a special form of a sequent calculus left rule that requires both the active formula of the right premiss and the main formula to be in the stoup. This entails, in a derivation, that if the inference immediately above the right premiss of an instance of Lft is not an instance of Ax, then it is again an instance of Lft. Therefore, the rightmost branch of a derivation $\mathcal{D}$ ending with a list sequent is a sequence of k instances of Lft ($k\geq 0$) topped with an instance of Ax.

Going back to (1), one can now see that the formula $A\supset B$ introduced by $m-Left$ is actually of the form $A\supset B_1\supset ...\supset B_k\supset C$, for some $k\geq 0$. A particular case of this rule is (2), obtained when the second premiss of $m-Left$ is an instance of Ax and, hence, $k=0$, $l=[]$ and $B=C$.

$$\frac{y:A\supset B,\Gamma;-\vdash u:A \quad x:B,y:A\supset B,\Gamma;-\vdash v:D}{y:A\supset B,\Gamma;-\vdash y(u,[],(x)v):D}\ Left \tag{2}$$

This is the ordinary binary left rule for implication.

We now want to explain what kind of derivations become available when one goes beyond Schwitchenberg's derivations, *i.e.* when one no longer requires every gm-elimination to be a m-left inference. For that purpose, we interpret $\lambda\mathbf{J^m}$ in Herbelin's $\overline{\lambda}$-calculus [4].

2.3 Interpretation into $\overline{\lambda}$-calculus

In $\overline{\lambda}$ we find the same separation between terms and lists, but there are no variables and no gm-applications. Instead there are the term constructors xl (dereliction), tl (head-cut) and $v\{x:=t\}$ (mid-cut), with typing rules

$$\frac{x:A,\Gamma;A\vdash l:B}{x:A,\Gamma;-\vdash xl:B}\ Der$$

$$\frac{\Gamma;-\vdash t:A \quad \Gamma;A\vdash l:B}{\Gamma;-\vdash tl:B}\ h-Cut \qquad \frac{\Gamma;-\vdash t:A \quad x:A,\Gamma;-\vdash v:B}{\Gamma;-\vdash v\{x:=t\}:B}\ m-Cut$$

Now, on the one hand, a variable y is interpreted as the dereliction $y[]$ and, on the other hand, a gm-application $t(u,l,(x)v)$ is interpreted as the combination $v\{x:=t(u::l)\}$ of an head-cut and a mid-cut:

$$\dfrac{\Gamma; - \vdash t : A \supset B \quad \dfrac{\Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\ Lft}{\dfrac{\Gamma; - \vdash t(u :: l) : C \qquad\qquad x : C, \Gamma; - \vdash v : D}{\Gamma; - \vdash v\{x := t(u :: l)\} : D}\ m - Cut}\ h - Cut \tag{3}$$

Notice that the computational interpretation of a mid-cut is an explicit substitution [4, 2]. The same interpretation was given to $v_x(y, \boldsymbol{L})$ in [7].

When the rightmost premiss of a gm-elimination is an instance of *Axiom*, the gm-elimination is essentially an head-cut $t(u :: l)$, which corresponds to a right-permuted cut. Only head-cuts $t[]$ are missing in $\lambda\mathbf{J^m}$. However, $t[]$ and t are the same, up to a trivial right-permutation of cuts. The particular form $x(u :: l)$ of head-cuts (together with variables x) gives to $\lambda\mathbf{J^m}$ the full power of dereliction xl. In this sense, $\lambda\mathbf{J^m}$ includes the entire cut-free fragment of $\overline{\lambda}$.

According to (3), a gm-elimination is a particular form of mid-cut. In $\lambda\mathbf{J^m}$ we do not have general mid-cuts - and this is what is missing for having a direct simulation of LJ with cuts inside $\lambda\mathbf{J^m}$. Instead, there is the derived rule

$$\frac{\Gamma; - \vdash t : A \quad x : A, \Gamma; - \vdash v : B}{\Gamma; - \vdash \mathbf{s}(t, x, v) : B} \tag{4}$$

where $\mathbf{s}(t, x, v)$ is the *generalised multiary substitution* operation (gm-substitution, for short), to be introduced in Definition 3 below.

2.4 Reduction rules

In $\lambda\mathbf{J^m}$ we have reduction rules and permutative conversions. Permutative conversions aim to reduce gm-eliminations to a particular form that corresponds to the elimination rule of natural deduction. They are defined in Section 4 and constitute the central topic of our study. Reduction rules are introduced now.

Definition 3 *The* reduction rules *for* $\lambda\mathbf{J^m}$ *are as follows:*

$$\begin{array}{lrcl}
(\beta_1) & (\lambda x.t)(u, [], (y)v) & \rightarrow & \mathbf{s}(\mathbf{s}(u, x, t), y, v) \\
(\beta_2) & (\lambda x.t)(u, v :: l, (y)v') & \rightarrow & \mathbf{s}(u, x, t)(v, l, (y)v') \\
(\pi) & t(u, l, (x)v)(u', l', (y)v') & \rightarrow & t(u, l, (x)v(u', l', (y)v')) \\
(\mu) & t(u, l, (x)x(u', l', (y)v)) & \rightarrow & t(u, \mathbf{append}(l, u', l'), (y)v), \quad x \notin u', l', v
\end{array}$$

where

$$\begin{array}{rcl}
\mathbf{s}(t, x, x) & = & t \\
\mathbf{s}(t, x, y) & = & y, \; y \neq x \\
\mathbf{s}(t, x, \lambda y.u) & = & \lambda y.\mathbf{s}(t, x, u) \\
\mathbf{s}(t, x, u(v, l, (y)v')) & = & \mathbf{s}(t, x, u)(\mathbf{s}(t, x, v), \mathbf{s}'(t, x, l), (y)\mathbf{s}(t, x, v')) \\
\\
\mathbf{s}'(t, x, []) & = & [] \\
\mathbf{s}'(t, x, v :: l) & = & \mathbf{s}(t, x, v) :: \mathbf{s}'(t, x, l) \\
\\
\mathbf{append}([], u, l) & = & u :: l \\
\mathbf{append}(u' :: l', u, l) & = & u :: \mathbf{append}(l', u, l)
\end{array}$$

The notations $\rightarrow_{\beta,\pi,\mu}$ and $\rightarrow^*_{\beta,\pi,\mu}$ stand for the one-step and the zero or more steps reduction relations, respectively, induced by the reduction rules (β_1), (β_2), (π) and (μ), that is $\rightarrow_{\beta,\pi,\mu}$ is the compatible closure of these rules and $\rightarrow^*_{\beta,\pi,\mu}$ is the reflexive and transitive closure of $\rightarrow_{\beta,\pi,\mu}$.

The set of normal forms w.r.t. $\rightarrow_{\beta,\pi,\mu}$, which we write as $\mathbf{nf}(\lambda\mathbf{J^m})$, is given by the following grammar:

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

where in the last production for terms, if v is of the form $y(u', l', (y')v')$, y must occur either in u', l' or v'. The normal forms w.r.t. $\rightarrow_{\beta,\pi}$ are as above omitting this last proviso. The latter and the former sets of normal forms correspond to Schwichtenberg's "multiary sequent terms" and their "multiary normal forms", respectively.

Reduction rule (μ) is structural and was already introduced in [7]. Consider the μ-redex in Definition 3 and a typing derivation $\mathcal{D}$ of this redex. When $x \notin u', l', v'$, the name x is in some sense redundant. Its type, of the form $A \supset B$, instead of being introduced in $\mathcal{D}$ by the m-left inference $x(u', l', (y)v')$, could have been introduced in a more "multiary" fashion by a Lft-inference $u' :: l'$. This transformation of $\mathcal{D}$ is the ultimate effect of (μ). Notice that there are in $\mathcal{D}$ subderivations of $\Gamma; D \vdash l : A \supset B$, $\Gamma; - \vdash u' : A$ and $\Gamma; B \vdash l' : C$, for some Γ, C, D, and that

$$\frac{\Gamma; D \vdash l : A \supset B \quad \Gamma; - \vdash u' : A \quad \Gamma; B \vdash l' : C}{\Gamma; D \vdash \mathbf{append}(l, u', l') : C} \tag{5}$$

is a derived rule of $\lambda\mathbf{J^m}$.

Reduction rules (β_1), (β_2) and (π) perform cut-elimination. Consider a gm-elimination $t(u, l, (y)v)$ and let us focus on the head-cut $t(u :: l)$ that exists inside it, in the sense of (3). Such an head-cut is right-permuted, because its right cut-formula $A \supset B$, being in the stoup, is main and linear. Now, if t is a gm-elimination, the head-cut is left-permutable. Rule π permutes the lower gm-elimination, where $t(u :: l)$ lives, over the upper gm-elimination t. If t is a λ-abstraction $\lambda x.t_0$, the head-cut is both left- and right-permuted, and the key-step of cut-elimination applies. A first cut with cut-formula A is generated and immediately eliminated by performing $\mathbf{s}(u, x, t_0)$. Now, if l is empty, we do not have arguments for forming a new gm-elimination, and a another substitution is called - this is rule (β_1). If $l = v_0 :: l_0$, a second cut with cut-formula B is generated, namely the head-cut $(\mathbf{s}(u, x, t_0))(v_0 :: l_0)$ that lives in the new gm-elimination $\mathbf{s}(u, x, t_0)(v_0, l_0, (y)v)$ - this is rule (β_2).

With the help of the fact that typing rules (4) and (5) are derivable, it is routine to prove subject reduction for $\rightarrow_{\beta,\pi,\mu}$.

We end this section by stating two properties of gm-substitution, that follow by routine induction.

Lemma 1 *For all $t, u \in \mathbf{\Lambda J^m}$, $\mathbf{s}(t, x, u) = u$, if $x \notin u$.*

Lemma 2 (Substitution Lemma) *For all* $t, u, v \in \mathbf{\Lambda J^m}$ *such that* $y \notin t$*, and* $x \neq y$*,* $\mathbf{s}(t, x, \mathbf{s}(u, y, v)) = \mathbf{s}(\mathbf{s}(t, x, u), y, \mathbf{s}(t, x, v))$*.*

3 Various subsystems of $\boldsymbol{\lambda}\mathbf{J^m}$

In this section we define several subsystems of $\lambda\mathbf{J^m}$ by constraining the construction $t(u, l, (x)v)$, as illustrated in the following diagram:

$$
\begin{tikzcd}
\begin{array}{|c|}\hline \lambda\mathbf{J^m} \\ \hline \text{gm-application} \\ \hline t(u,l,(x)v) \\ \hline \\ \hline \end{array} \arrow[r] \arrow[d] & \begin{array}{|c|}\hline \lambda\mathbf{J} \\ \hline \text{g-application} \\ \hline t(u,[],(x)v) \\ \hline \text{Abbreviation: } t(u{\cdot}(x)v) \\ \hline \end{array} \arrow[d] \\
\begin{array}{|c|}\hline \lambda\mathcal{P}\mathbf{h} \\ \hline \text{m-application} \\ \hline t(u,l,(x)x) \\ \hline \text{Abbreviation: } t(u{\cdot}l) \\ \hline \end{array} \arrow[r] & \begin{array}{|c|}\hline \lambda\mathcal{G} \\ \hline \text{application} \\ \hline t(u,[],(x)x) \\ \hline \text{Abbreviation: } t[u] \\ \hline \end{array}
\end{tikzcd}
$$

It turns out that, by doing so, we capture a number of previously known systems as subsystems of $\lambda\mathbf{J^m}$, w.r.t. expressions, typing rules and reduction rules allowed.

3.1 $\boldsymbol{\lambda}$J: the generalised $\boldsymbol{\lambda}$-calculus

Definition 4 *The terms of* $\lambda\mathbf{J}$ *are described in the following grammar:*

$$
\begin{array}{rl}
(\lambda\mathbf{J} - \mathit{terms})\ t, u, v & ::= x \mid \lambda x.t \mid t(u, l, (x)v) \\
(\lambda\mathbf{J} - \mathit{lists}) \qquad l & ::= []
\end{array}
$$

where x *ranges over the set* $\mathbf{V}$ *of* variables*.* $\mathbf{\Lambda J}$ *and* $\mathcal{L}\mathbf{J}$ *are used to denote the sets of* $\lambda\mathbf{J}$*-terms and* $\lambda\mathbf{J}$*-lists respectively.*

Hence, the sets $\mathbf{\Lambda J}$ and $\mathcal{L}\mathbf{J}$ are obtained from $\mathbf{\Lambda J^m}$ and $\mathcal{L}\mathbf{J^m}$, respectively, by omitting cons-lists. Since there is only one form of lists in $\lambda\mathbf{J}$, every gm-application in $\lambda\mathbf{J}$ is of the form $t(u, [], (x)v)$, which we call a *generalised application* (or g-application, for short). $\lambda\mathbf{J}$-terms can simply be described as:

$$(\lambda\mathbf{J} - \mathit{terms})\ t, u, v ::= x \mid \lambda x.t \mid t(u{\cdot}(x)v) \quad ,$$

where $t(u{\cdot}(x)v)$ is used as an abbreviation to $t(u, [], (x)v)$. This expression can be typed by the derived rule

$$\frac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A \quad x : B, \Gamma; - \vdash v : C}{\Gamma; - \vdash t(u{\cdot}(x)v) : C}\ g - Elim \quad , \tag{6}$$

with proviso $x : B$ does not belong to Γ, which corresponds to an instance of the rule $gm - Elim$ where the penultimate premiss is an instance of Ax.

Definition 5 *The* reduction rules *for* $\lambda\mathbf{J}$ *are as follows:*

$$\begin{array}{rl} (\beta_1) & (\lambda x.t)(u\cdot(y)v) \to \mathbf{s}(\mathbf{s}(u,x,t),y,v) \\ (\pi) & t(u\cdot(x)v)(u'\cdot(y)v') \to t(u\cdot(x)v(u'\cdot(y)v')) \end{array}$$

$$\begin{array}{lrcl} \textit{where} & \mathbf{s}(t,x,x) & = & x \\ & \mathbf{s}(t,x,y) & = & y,\ y \neq x \\ & \mathbf{s}(t,x,\lambda y.u) & = & \lambda y.\mathbf{s}(t,x,u) \\ & \mathbf{s}(t,x,u(v\cdot(y)v')) & = & \mathbf{s}(t,x,u)(\mathbf{s}(t,x,v)\cdot(y)\mathbf{s}(t,x,v')) \end{array}$$

Comparatively to $\lambda\mathbf{J^m}$, $\lambda\mathbf{J}$ drops all rules and clauses involving *cons*. Since β_2-redexes and μ-*contracta* fall outside $\mathbf{\Lambda J}$ (notice that $\mathbf{append}([], u', l')$ is a cons-list), the rules (β_2) and (μ) are omitted.

The set $\mathbf{nf}(\lambda\mathbf{J})$ of $\lambda\mathbf{J}$ normal forms is given by the following grammar:

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

Because of the omission of the (μ)-rule, there are $\lambda\mathbf{J}$ normal forms which are not $\lambda\mathbf{J^m}$ normal forms.

The system thus obtained is the so-called ΛJ-calculus of Joachimski and Matthes [5], which in turn is the Curry-Howard counterpart to a system of natural deduction due to von Plato [8], where the idea of generalised elimination rules originated. These rules allow arbitrary consequences, as in the usual natural deduction rule for disjunction elimination, and the rule for implication (with term annotations) is

$$\frac{t:A \supset B \quad u:A \quad \begin{array}{c}[x:B]\\ \vdots \\ v:C\end{array}}{t(u\cdot(x)v):C} \supset E \ .$$

The $g - Elim$ rule in (6) is then the "sequent style" formulation of this natural deduction rule.

3.2 $\lambda\mathcal{P}\mathbf{h}$: the multiary λ-calculus

Definition 6 *The terms of* $\lambda\mathcal{P}\mathbf{h}$ *are described in the following grammar:*

$$\begin{array}{rl} (\lambda\mathcal{P}\mathbf{h} - terms) & t, u ::= x \mid \lambda x.t \mid t(u, l, (x)x) \\ (\lambda\mathcal{P}\mathbf{h} - lists) & l ::= t :: l \mid [] \end{array}$$

where x *ranges over the set* $\mathbf{V}$ *of* variables. $\mathbf{\Lambda}\mathcal{P}\mathbf{h}$ *and* $\mathcal{L}\mathcal{P}\mathbf{h}$ *are used to denote the sets of* $\lambda\mathcal{P}\mathbf{h}$*-terms and* $\lambda\mathcal{P}\mathbf{h}$*-lists respectively.*

Hence, the sets $\mathbf{\Lambda}\mathcal{P}\mathbf{h}$ and $\mathcal{L}\mathcal{P}\mathbf{h}$ are obtained from $\mathbf{\Lambda J^m}$ and $\mathcal{L}\mathbf{J^m}$, respectively, by constraining v in $t(u, l, (x)v)$ to be x. A gm-application of the form $t(u, l, (x)x)$

is called a *multiary application* (or m-application, for short) and is written as $t(u{\cdot}l)$. It may be typed with the derived rule

$$\frac{\Gamma;-\vdash t{:}A\supset B \quad \Gamma;-\vdash u{:}A \quad \Gamma;B\vdash l{:}C}{\Gamma;-\vdash t(u{\cdot}l){:}C}\ m-Elim$$

which corresponds to an instance of the rule $gm - Elim$ where the rightmost premiss is an instance of $Axiom$.

Definition 7 *The* reduction rules *for* $\lambda\mathcal{P}\mathbf{h}$ *are as follows:*

$$\begin{array}{rrl}
(\beta_1) & (\lambda x.t)(u{\cdot}[]) & \to \mathbf{s}(u,x,t) \\
(\beta_2) & (\lambda x.t)(u{\cdot}v::l) & \to \mathbf{s}(u,x,t)(v{\cdot}l) \\
(h) & t(u{\cdot}l)(u'{\cdot}l') & \to t(u{\cdot}\mathbf{append}(l,u',l'))
\end{array}$$

$$\begin{array}{rrl}
\textit{where} & \mathbf{s}(t,x,x) & = x \\
 & \mathbf{s}(t,x,y) & = y,\ y\neq x \\
 & \mathbf{s}(t,x,\lambda y.u) & = \lambda y.\mathbf{s}(t,x,u) \\
 & \mathbf{s}(t,x,u(v{\cdot}l)) & = \mathbf{s}(t,x,u)(\mathbf{s}(t,x,v){\cdot}\mathbf{s}'(t,x,l)) \\
 & & \\
 & \mathbf{s}'(t,x,[]) & = [] \\
 & \mathbf{s}'(t,x,v::l) & = \mathbf{s}(t,x,v)::\mathbf{s}'(t,x,l) \\
 & & \\
 & \mathbf{append}([],u,l) & = u::l \\
 & \mathbf{append}(t::l,u',l') & = t::\mathbf{append}(l,u',l')
\end{array}$$

Observe that π-*contracta* and μ-redexes fall outside $\mathbf{\Lambda}\mathcal{P}\mathbf{h}$. The (π) and (μ) rules have been combined into the new rule (h):

$$\begin{array}{rl}
t(u{\cdot}l)(u'{\cdot}l') & = t(u,l,(x)x)(u',l',(y)y) \\
 & \to_\pi t(u,l,(x)x(u',l',(y)y)) \\
 & \to_\mu t(u,\mathbf{append}(l,u',l'),(y)y) \\
 & = t(u{\cdot}\mathbf{append}(l,u',l'))\ .
\end{array}$$

The set of $\lambda\mathcal{P}\mathbf{h}$ normal forms is the restriction to $\lambda\mathcal{P}\mathbf{h}$ of the set of $\lambda\mathbf{J^m}$ normal forms and can thus be described as follows:

$$\begin{array}{rl}
t,u ::= & x \mid \lambda x.t \mid x(u{\cdot}l) \\
l ::= & u::l \mid []
\end{array}$$

The system thus obtained is identical to the $\lambda\mathcal{P}\mathbf{h}$-calculus defined in [2, 3]. Originally this system was seen as a calculus of right-permuted cuts, but it can also be seen as the natural extension of the λ-calculus where functions may be applied to lists of arguments.

3.3 $\lambda\mathcal{G}$: Gentzen-style λ-calculus

The calculus $\lambda\mathcal{G}$ is the subsystem obtained from $\lambda\mathbf{J^m}$ by combining the constraints that define $\lambda\mathbf{J}$ and $\lambda\mathcal{P}\mathbf{h}$, that is, by constraining gm-applications to be of the form $t(u, [], (x)x)$, which we call a *(simple) application*. Therefore the set of $\lambda\mathcal{G}$-terms, written as $\mathbf{\Lambda}\mathcal{G}$, can be characterised as follows:

$$\begin{array}{rl} (\lambda\mathcal{G} - terms) & t, u ::= x \mid \lambda x.t \mid t(u, l, (x)x) \\ (\lambda\mathcal{G} - lists) & \quad l ::= [] \end{array}$$

As before an application $t(u, [], (x)x)$ can be abbreviated to $t(u \cdot [])$. Further, we even write this last expression as $t[u]$. The typing rule is the following derived rule:

$$\frac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A}{\Gamma; - \vdash t[u] : B} \; Elim$$

which one can recognize as the ordinary natural deduction elimination rule. This rule is an instance of the rule $gm - Elim$ where the last two premisses are instances of Ax and $Axiom$ respectively. Using the abbreviations above, and since there is only one form of lists in $\lambda\mathcal{G}$, $\lambda\mathcal{G}$-terms can simply be described as follows:

$$(\lambda\mathcal{G} - terms) \; t, u ::= x \mid \lambda x.t \mid t[u]$$

Definition 8 *The unique* reduction rule *for $\lambda\mathcal{G}$ is as follows:*

$$(\beta_1) \qquad (\lambda x.t)[u] \to \mathbf{s}(u, x, t)$$

$$\begin{array}{rrl} where & \mathbf{s}(t, x, x) & = x \\ & \mathbf{s}(t, x, y) & = y, \; y \neq x \\ & \mathbf{s}(t, x, \lambda y.u) & = \lambda y.\mathbf{s}(t, x, u) \\ & \mathbf{s}(t, x, u[v]) & = \mathbf{s}(t, x, u)[\mathbf{s}(t, x, v)] \end{array}$$

The system $\lambda\mathcal{G}$ is no more than an isomorphic copy of the λ-calculus. In fact, $\lambda\mathcal{G}$ is the image of Gentzen's mapping $\mathcal{G}$ from natural deduction into sequent calculus [2].

Again, the normal forms of $\lambda\mathcal{G}$ are not necessarily normal forms of its extensions. For example, $t[u][u']$ is not a redex of $\lambda\mathcal{G}$, but it is a redex of its extensions.

4 Permutative conversions for $\boldsymbol{\lambda}\mathbf{J}^{\mathbf{m}}$

Terms of $\lambda\mathbf{J}^{\mathbf{m}}$ can be interpreted onto $\lambda\mathcal{G}$ via the mapping ϕ introduced below. In this section we identify a set of conversions on $\lambda\mathbf{J}^{\mathbf{m}}$-terms, corresponding to certain oriented permutations on derivations. From the viewpoint of λ-calculi, the goal of permutations is to reduce gm-applications to simple applications. Taking this set of conversions as defining a rewriting system, one obtains a strongly normalising and confluent rewriting system such that the normal form of a $\lambda\mathbf{J}^{\mathbf{m}}$-term is its ϕ-image. Moreover we get a permutability theorem: two $\lambda\mathbf{J}^{\mathbf{m}}$-terms have the same ϕ-image iff they are inter-permutable.

4.1 Mapping ϕ

Definition 9 *$\phi : \mathbf{\Lambda J^m} \longrightarrow \mathbf{\Lambda}\mathcal{G}$ and $\phi' : \mathbf{\Lambda}\mathcal{G} \times \mathbf{\Lambda}\mathcal{G} \times \mathcal{L}\mathbf{J^m} \times \mathbf{V} \times \mathbf{\Lambda}\mathcal{G} \longrightarrow \mathbf{\Lambda}\mathcal{G}$ are defined by simultaneous recursion as follows:*

$$\begin{aligned}
\phi(x) &= x \\
\phi(\lambda x.t) &= \lambda x.\phi(t) \\
\phi(t(u,l,(x)v)) &= \phi'(\phi(t),\phi(u),l,x,\phi(v)) \\
\\
\phi'(t,u,[],x,v) &= \mathbf{s}(t[u],x,v) \\
\phi'(t,u,v::l,x,v') &= \phi'(t[u],\phi(v),l,x,v')
\end{aligned}$$

The following two propositions establish that ϕ commutes with application and that $\lambda\mathcal{G}$-terms are invariant under ϕ, and thus ϕ is onto.

Proposition 1 *For all $t, u \in \mathbf{\Lambda J^m}$, $\phi(t[u]) = \phi(t)[\phi(u)]$.*

Proof: $\phi(t[u]) = \phi'(\phi(t),\phi(u),[],x,\phi(x)) = \mathbf{s}(\phi(t)[\phi(u)],x,x) = \phi(t)[\phi(u)]$. □

Proposition 2 *For all $t \in \mathbf{\Lambda}\mathcal{G}$, $\phi(t) = t$.*

Proof: Routine induction on t. The case of application uses Proposition 1. □

Two other properties relating the mappings ϕ and ϕ' with the notion of substitution are as follows.

Proposition 3 *For all $t, u, v \in \mathbf{\Lambda J^m}$ and for all $l \in \mathcal{L}\mathbf{J^m}$,*

$$\phi'(\phi(t),\phi(u),l,x,\phi(v)) = \mathbf{s}(\phi(t(u{\cdot}l)),x,\phi(v)) \ .$$

Proof: By induction on l. □

Proposition 4

(i) $\phi(\mathbf{s}(t,x,v)) = \mathbf{s}(\phi(t),x,\phi(v))$, *for all $t, v \in \mathbf{\Lambda J^m}$.*

(ii) $\phi'(\mathbf{s}(\phi(t),x,\phi(v_1)),\mathbf{s}(\phi(t),x,\phi(v_2)),\mathbf{s}'(t,x,l),y,\mathbf{s}(\phi(t),x,\phi(v)))$
$= \mathbf{s}(\phi(t),x,\phi'(\phi(v_1),\phi(v_2),l,y,\phi(v)))$, *for all $t, v, v_1, v_2 \in \mathbf{\Lambda J^m}$, $l \in \mathcal{L}\mathbf{J^m}$.*

Proof: By simultaneous induction on v and l. □

4.2 Permutative Conversions

Permutative conversions on $\lambda\mathbf{J^m}$-terms, that from here on we call simply permutations, are divided into **p**-*permutations* and **q**-*permutations*. **p**-permutations (resp. **q**-permutations) deal with "generality" (resp. multiarity) of gm-application. The **p**-permutations are:

(p_1) $t(u,l,(x)y) \to y, \quad x \neq y$

(p_2) $t(u,l,(x)\lambda y.v) \to \lambda y.t(u,l,(x)v)$

(p_3) $t_1(u_1,l_1,(x)t_2(u_2,l_2,(y)v)) \to$
$t_1(u_1,l_1,(x)t_2)(t_1(u_1,l_1,(x)u_2),\mathbf{p_3'}(t_1,u_1,l_1,x,l_2),(y)v)$ if $x \notin v$,

where

$$\begin{aligned}
&\mathbf{p_3'} : \mathbf{\Lambda J^m} \times \mathbf{\Lambda J^m} \times \mathcal{L}\mathbf{J^m} \times \mathbf{V} \times \mathcal{L}\mathbf{J^m} \longrightarrow \mathcal{L}\mathbf{J^m} \\
&\mathbf{p_3'}(t,u,l,x,[]) = [] \\
&\mathbf{p_3'}(t,u,l,x,u'::l') = t(u,l,(x)u')::\mathbf{p_3'}(t,u,l,x,l') \ .
\end{aligned}$$

Mapping $\mathbf{p_3'}$ is a way of generating a gm-application for each element of a list. $\mathbf{p}$-permutations are essentially as in Schwichtenberg's [7]. In particular, we follow the idea of requiring $x \notin v$ in (p_3), which is crucial in guaranteeing termination. In a sense, our permutations are simpler because we are not obliged to stay in the cut-free fragment and do not need permutations to preserve μ-normal form.

The unique $\mathbf{q}$-permutation is

$$(q)\ \ t(u, v :: l, (x)v') \to t[u](v, l, (x)v')\ .$$

This kind of permutation is first pointed out in [2] in the context of the system $\lambda\mathcal{P}\mathbf{h}$. As opposed to [7], a permutation to deal with multiarity is necessary because we are not confined to the cut-free fragment.

We use $\to_p$ and $\to_q$ to denote the one-step reduction relations (compatible closure) induced by $\mathbf{p}$-permutations and by the $\mathbf{q}$-permutation respectively. The notation $\to_{p,q}$ represents the union of $\to_p$ and $\to_q$. As usual, given a one-step reduction relation $\to_r$, $\to_r^*$ and $\leftrightarrow_r^*$ denote respectively its reflexive and transitive closure and its equivalence closure.

Proposition 5 establishes invariance of permutation reducibility under the mapping ϕ. In its proof we use the following lemma, proved by induction on l.

Lemma 3 *$\phi'(t, u, \mathbf{s}'(t_1(u_1{\cdot}l_1), x, l), y, v) = \phi'(t, u, \mathbf{p_3'}(t_1, u_1, l_1, x, l), y, v)$, for all $t, u, v, t_1, u_1 \in \mathbf{\Lambda J^m}$, $l, l_1 \in \mathcal{L}\mathbf{J^m}$.*

Proposition 5 *If $t \to_{p,q}^* u$ then $\phi(t) = \phi(u)$, for all $t, u \in \mathbf{\Lambda J^m}$.*

Proof: The proof follows by induction on the relation $\to_{p,q}^*$. □

Proposition 6 asserts that any $\lambda\mathbf{J^m}$-term can be reduced to its ϕ-image using $\mathbf{p}$ and $\mathbf{q}$-permutations. Its proof uses the fact that g-applications can be reduced to substitutions using solely $\mathbf{p}$-permutations, as in the following lemma proved by induction on v.

Lemma 4 *$t(u{\cdot}(x)v) \to_p^* \mathbf{s}(t[u], x, v)$, for all $t, u, v \in \mathbf{\Lambda}\mathcal{G}$.*

Proposition 6 *(i) $t \to_{p,q}^* \phi(t)$, for all $t \in \mathbf{\Lambda J^m}$.*
(ii) $\phi(t)(\phi(u), l, (x)\phi(v)) \to_{p,q}^* \mathbf{s}(\phi(t(u{\cdot}l)), x, \phi(v))$,
for all $l \in \mathcal{L}\mathbf{J^m}$ and for all $t, u, v \in \mathbf{\Lambda J^m}$.

Proof: By simultaneous induction on the structure of t and l. □

4.3 Main Results

Propositions 5 and 6, establishing respectively invariance of $\mathbf{p}$ and $\mathbf{q}$-permutations under ϕ and reducibility to the ϕ-image using such permutations, are now used in proving the theorems below.

Theorem 1 (characterization of $\mathbf{p},\mathbf{q}$-normal forms) *For all $t \in \mathbf{\Lambda J^m}$, t is $\mathbf{p},\mathbf{q}$-normal iff $t \in \mathbf{\Lambda}\mathcal{G}$.*

Proof: On the one hand, terms of $\lambda\mathcal{G}$ have neither $\mathbf{p}$ or $\mathbf{q}$-redexes and so they are $\mathbf{p},\mathbf{q}$-normal. Consider, on the other hand, that t is $\mathbf{p},\mathbf{q}$-normal. By Proposition 6, $t \rightarrow^*_{p,q} \phi(t)$ and thus the normality of t implies $t = \phi(t)$. The proof concludes observing that the co-domain of ϕ is $\mathbf{\Lambda}\mathcal{G}$. $\square$

Theorem 2 (confluence) *For all $t, t_1, t_2 \in \mathbf{\Lambda J^m}$, if $t \rightarrow^*_{p,q} t_1$ and $t \rightarrow^*_{p,q} t_2$, then $t_1 \rightarrow^*_{p,q} t_3$ and $t_2 \rightarrow^*_{p,q} t_3$, for some $t_3 \in \mathbf{\Lambda J^m}$.*

Proof: Assuming $t \rightarrow^*_{p,q} t_1$ and $t \rightarrow^*_{p,q} t_2$, by Proposition 6 follows that $t_1 \rightarrow^*_{p,q} \phi(t_1)$ and $t_2 \rightarrow^*_{p,q} \phi(t_2)$. Yet by Proposition 6 we have $t \rightarrow^*_{p,q} \phi(t)$ and we can now use Proposition 5 to conclude that $\phi(t_1) = \phi(t) = \phi(t_2)$. $\square$

Theorem 3 (permutability) *$\phi(t_1) = \phi(t_2)$ iff $t_1 \leftrightarrow^*_{p,q} t_2$, for all $t_1, t_2 \in \mathbf{\Lambda J^m}$.*

Proof: Proposition 5 guarantees $\phi(t_1) = \phi(t_2)$ whenever $t_1 \leftrightarrow^*_{p,q} t_2$. As to the only if part, we use Proposition 6, obtaining $t_1 \rightarrow^*_{p,q} \phi(t_1)$ and $t_2 \rightarrow^*_{p,q} \phi(t_2)$ and thus, as by hypothesis $\phi(t_1) = \phi(t_2)$, t_1 and t_2 are inter-permutable. $\square$

In order to ensure termination of the rewriting system induced by permutations $\mathbf{p}$ and $\mathbf{q}$, we introduce a notion of weight for terms and lists of $\lambda\mathbf{J^m}$.

Definition 10 *$\mathbf{w}(t)$ and $\mathbf{w}(l)$, the* weight *of an $\lambda\mathbf{J^m}$-term t and of an $\mathcal{L}\mathbf{J^m}$-list l respectively, are defined as follows:*

$$\begin{array}{rclrcl}
\mathbf{w}(x) &=& 1 & \mathbf{w}([]) &=& 0 \\
\mathbf{w}(\lambda x.t) &=& 1 + \mathbf{w}(t) & \mathbf{w}(u :: l) &=& 2 + \mathbf{w}(u) + \mathbf{w}(l) \\
\mathbf{w}(t(u, l, (x)v)) &=& \mathbf{w}(v)(\mathbf{w}(t) + \mathbf{w}(u) + \mathbf{w}(l) + 1) & & &
\end{array}$$

Theorem 4 (termination) *The rewriting system induced by permutations $\mathbf{p}$ and $\mathbf{q}$ is strongly normalisable.*

Proof: Each permutation can be shown to have a RHS of weight lower than its LHS and thus every sequence of permutations must be finite. For permutation (p_3), we use the inequality below, that can be proved by induction on l_2.

$$\mathbf{w}(\mathbf{p'_3}(t_1, u_1, l_1, x, l_2)) < (\mathbf{w}(l_2) + 1)(\mathbf{w}(t_1) + \mathbf{w}(u_1) + \mathbf{w}(l_1)) + \mathbf{w}(l_2) \quad \square$$

Since $\rightarrow_{p,q}$ is confluent and strongly normalising, each $\lambda\mathbf{J^m}$-term t has a unique normal form that we denote by $\downarrow_{p,q}(t)$.

Theorem 5 (representation of ϕ) *$\phi(t) = \downarrow_{p,q}(t)$, for all $t \in \mathbf{\Lambda J^m}$.*

Proof: By Proposition 6, $t \rightarrow^*_{p,q} \phi(t)$ and $\phi(t)$ is a normal form. $\square$

5 Decomposing the main results

This section illustrates two ways of decomposing the mapping ϕ using $\lambda\mathcal{P}\mathbf{h}$ and $\lambda\mathbf{J}$ as intermediate systems, according to Figure 1 in the introductory section. In the spirit of the study in the previous section for mapping ϕ, similar results may be established (i) for the mappings $\mathbf{p}$ and $\mathbf{p^m}$ relatively to $\mathbf{p}$-permutations and (ii) for the mappings $\mathbf{q}$ and $\mathbf{q^m}$ relatively to $\mathbf{q}$-permutations.

Definition 11 *The mappings* $\mathbf{p^m}$*,* $\mathbf{q^m}$*,* $\mathbf{p}$ *and* $\mathbf{q}$ *are as follows.*

$$
\begin{aligned}
\mathbf{p^m} &: \mathbf{\Lambda J^m} \longrightarrow \mathbf{\Lambda}\mathcal{P}\mathbf{h}\\
\mathbf{p^m}(x) &= x\\
\mathbf{p^m}(\lambda x.t) &= \lambda x.\mathbf{p^m}(t)\\
\mathbf{p^m}(t(u,l,(x)v)) &= \mathbf{s}(\mathbf{p^m}(t)(\mathbf{p^m}(u)\cdot\mathbf{p^m}'(l)),x,\mathbf{p^m}(v))\\
\mathbf{p^m}'([]) &= []\\
\mathbf{p^m}'(u::l) &= \mathbf{p^m}(u)::\mathbf{p^m}'(l)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{q^m} &: \mathbf{\Lambda J^m} \longrightarrow \mathbf{\Lambda J}\\
\mathbf{q^m}(x) &= x\\
\mathbf{q^m}(\lambda x.t) &= \lambda x.\mathbf{q^m}(t)\\
\mathbf{q^m}(t(u,l,(x)v)) &= \mathbf{q^m}'(\mathbf{q^m}(t),\mathbf{q^m}(u),l,x,\mathbf{q^m}(v))\\
\mathbf{q^m}'(t,u,v::l,x,v') &= \mathbf{q^m}'(t[u],\mathbf{q^m}(v),l,x,v')\\
\mathbf{q^m}'(t,u,[],x,v) &= t(u\cdot(x)v)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{p} &: \mathbf{\Lambda J} \longrightarrow \mathbf{\Lambda}\mathcal{G}\\
\mathbf{p}(x) &= x\\
\mathbf{p}(\lambda x.t) &= \lambda x.\mathbf{p}(t)\\
\mathbf{p}(t(u\cdot(x)v)) &= \mathbf{s}(\mathbf{p}(t)[\mathbf{p}(u)],x,\mathbf{p}(v))
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{q} &: \mathbf{\Lambda}\mathcal{P}\mathbf{h} \longrightarrow \mathbf{\Lambda}\mathcal{G}\\
\mathbf{q}(x) &= x\\
\mathbf{q}(\lambda x.t) &= \lambda x.\mathbf{q}(t)\\
\mathbf{q}(t(u\cdot l)) &= \mathbf{q}'(\mathbf{q}(t),\mathbf{q}(u),l))\\
\mathbf{q}'(t,u,v::l)) &= \mathbf{q}'(t[u],\mathbf{q}(v),l))\\
\mathbf{q}'(t,u,[])) &= t[u]
\end{aligned}
$$

A mapping corresponding to $\mathbf{p}$ from $\lambda\mathbf{J}$ into the λ-calculus is given in [5]. In [2] an interpretation $\mathcal{Q}$ of $\lambda\mathcal{P}\mathbf{h}$ into the λ-calculus corresponding to $\mathbf{q}$ is studied.

Let us now consider $\mathbf{p}$ and $\mathbf{q}$-permutations in the context of the subsystems $\lambda\mathcal{P}\mathbf{h}$ and $\lambda\mathbf{J}$. In $\lambda\mathcal{P}\mathbf{h}$ we have that no $\mathbf{p}$-permutation applies to its terms and that the $\mathbf{q}$-permutation can be simplified as follows:

$$(q)\ t(u\cdot v::l) \to t[u](v\cdot l)\ .$$

Similarly, $\lambda\mathbf{J}$ is irreducible for $\mathbf{q}$-permutations and $\mathbf{p}$-permutations assume the following special forms.

$$
\begin{array}{lrl}
(p_1) & t(u\cdot(x)y) &\to y,\quad x\neq y\\
(p_2) & t(u\cdot(x)\lambda y.v) &\to \lambda y.t(u\cdot(x)v)\\
(p_3) & t_1(u_1\cdot(x)t_2(u_2\cdot(y)v)) &\to t_1(u_1\cdot(x)t_2)(t_1(u_1\cdot(x)u_2)\cdot(y)v)\ \text{ if }\ x\notin v
\end{array}
$$

The sequence of theorems in the previous section, leading to the representation of mapping ϕ via $\mathbf{p},\mathbf{q}$-normal forms, can also be established by analogous arguments for mappings $\mathbf{p}$, $\mathbf{q}$, $\mathbf{p^m}$ and $\mathbf{q^m}$. In particular, we have permutability theorems for all these mappings and we have confluence and strong normalisation of the rewriting systems induced by $\to_q$ on $\mathbf{\Lambda}\mathcal{P}\mathbf{h}$, $\to_p$ on $\mathbf{\Lambda J}$, $\to_p$ on $\mathbf{\Lambda J^m}$

and $\to_q$ on $\mathbf{\Lambda J^m}$. The unique normal forms of a $\lambda\mathbf{J^m}$-term t w.r.t. each of these systems is written as $\downarrow_q^{\lambda\mathcal{P}\mathbf{h}}(t)$, $\downarrow_p^{\lambda\mathbf{J}}(t)$, $\downarrow_p(t)$ and $\downarrow_q(t)$, respectively.

Theorem 6 (representation of p, q, $\mathbf{p^m}$ and $\mathbf{q^m}$)

(i) $\mathbf{q}(t) = \downarrow_q^{\lambda\mathcal{P}\mathbf{h}}(t)$*, for all* $t \in \mathbf{\Lambda}\mathcal{P}\mathbf{h}$. *(ii)* $\mathbf{p}(t) = \downarrow_p^{\lambda\mathbf{J}}(t)$*, for all* $t \in \mathbf{\Lambda J}$.
(iii) $\mathbf{p^m}(t) = \downarrow_p(t)$*, for all* $t \in \mathbf{\Lambda J^m}$. *(iv)* $\mathbf{q^m}(t) = \downarrow_q(t)$*, for all* $t \in \mathbf{\Lambda J^m}$.

6 Conclusion

This paper is a direct successor of [1, 7] in revisiting with the help of new λ-calculi —specifically, variants of Herbelin's calculus— Zucker and Pottinger classical results [9, 6]. The novelty here is that, at the same time, we pay attention to how the λ-calculus we arrive at, the system $\lambda\mathbf{J^m}$, more than being a term-annotation or book-keeping device [6], gives computational interpretations to fragments of sequent calculus. We believe it does so because: on the one hand $\lambda\mathbf{J^m}$ may be seen as an extension of the λ-calculus with a notion of generalised multiary application; on the other hand it captures as subsystems, in a modular fashion, a system isomorphic to the λ-calculus, the system ΛJ of Joachimski and Matthes and the system $\lambda\mathcal{P}\mathbf{h}$ in [3]. So, we consider worthwhile to investigate more into its properties. In particular, we intend to study confluence and normalisation of the reduction relation $\to_{\beta,\pi}$, introduced in Section 2 to perform cut-elimination. Also of interest is the study of properties of the rewriting system obtained by adding permutations $\mathbf{p}$ and $\mathbf{q}$ on top of the reduction relation $\to_{\beta,\pi}$.

References

1. R. Dyckhoff and L. Pinto, *Permutability of inferences in intuitionistic sequent calculi*, *Theoretical Computer Science*, **212** (1999) 141–155.
2. J. Espírito Santo, *Conservative extensions of the λ-calculus for the computational interpretation of sequent calculus*, *PhD thesis*, University of Edinburgh, 2002.
3. J. Espírito Santo, *An isomorphism between a fragment of sequent calculus and an extension of natural deduction*, *in: M. Baaz and A. Voronkov (Eds.), Proc. of LPAR'02*, 2002, Springer-Verlag, Lecture Notes in Artificial Intelligence, vol. **2514**, 354–366.
4. H. Herbelin, *A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure*, *in: L. Pacholski and J. Tiuryn (Eds.), Proceedings of CSL'94*, 1995, Springer-Verlag, Lecture Notes in Computer Science, vol. **933**, 61–75.
5. F. Joachimski and R. Matthes, *Short proofs of normalisation for the simply typed λ-calculus, permutative conversions and Gödel's* **T**, (accepted for publication in *Archive for Mathematical Logic*).
6. G. Pottinger, *Normalization as a homomorphic image of cut-elimination*, *Annals of Mathematical Logic* **12** (1977) 323–357.
7. H. Schwichtenberg, *Termination of permutative conversions in intuitionistic Gentzen calculi*, *Theoretical Computer Science*, **212** (1999) 247–260.
8. J. von Plato, *Natural deduction with general elimination rules*, *Annals of Mathematical Logic*, **40** (2001) 541–567.
9. J. Zucker, *The correspondence between cut-elimination and normalization*, *Annals of Mathematical Logic* **7** (1974) 1–112.