An isomorphism between a fragment of sequent calculus and an extension of natural deduction

José Espírito Santo

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

Abstract. Variants of Herbelin's λ-calculus, here collectively named Herbelin calculi, have proved useful both in foundational studies and as internal languages for the efficient representation of λ-terms.
An obvious requirement of both these two kinds of applications is a clear understanding of the relationship between cut-elimination in Herbelin calculi and normalisation in the λ-calculus. However, this understanding is not complete so far. Our previous work showed that λ is isomorphic to a Herbelin calculus, here named $\lambda\mathcal{P}$, only admitting cuts that are both left- and right-permuted. In this paper we consider a generalisation $\lambda\mathcal{P}h$ admitting any kind of right-permuted cut.
We show that there is a natural deduction system $\lambda\mathcal{N}h$ which conservatively extends λ and is isomorphic to $\lambda\mathcal{P}h$. The idea is to build in the natural deduction system a distinction between applicative term and application, together with a distinction between head and tail application. This is suggested by examining how natural deduction proofs are mapped to sequent calculus derivations according to a translation due to Prawitz.
In addition to β, $\lambda\mathcal{N}h$ includes a reduction rule that mirrors left permutation of cuts, but without performing any append of lists/spines.

1 Introduction

Herbelin introduced in [9] a fragment of sequent calculus for intuitionistic implicational logic (here named the *canonical* fragment), together with a term calculus, named the $\overline{\lambda}$-calculus, as a way of extending the Curry-Howard isomorphism to sequent calculus. Since then, variants of $\overline{\lambda}$ (let us call them *Herbelin calculi*) have been mainly used as a tool for proof-theoretical studies [5, 12, 6].

At the same time, the potential of Herbelin calculi as internal languages for the λ-calculus was recognised. This is manifest in the design of the "spine calculus" [2]. The potential lies in the fact that the representation of applicative terms brings the head to the surface [4]. This is useful in normalisation or unification procedures [2].

A clear understanding of the relationship between cut-elimination in Herbelin calculi and normalisation in the λ-calculus is obviously a basic requirement for both foundational studies and applications to the implementation of λ-calculi. Yet, such understanding is not complete so far. We are aware of only two examples of such kind of study [2, 6].

Our starting point is [6], where a very particular Herbelin calculus (here named $\lambda\mathcal{P}$) was proved isomorphic to λ. In $\lambda\mathcal{P}$ only cuts that are both left- and right-permuted are allowed. In this paper we define an extension of $\lambda\mathcal{P}$, named $\lambda\mathcal{P}h$, in which any right-permuted cut is admitted. Then we show that there is a natural deduction system, named $\lambda\mathcal{N}h$, which conservatively extends λ and is isomorphic to $\lambda\mathcal{P}h$.

The idea for $\lambda\mathcal{N}h$ is obtained by examining a mapping of natural deduction proofs to sequent calculus derivations due to Prawitz [11]. Through this mapping, eliminations correspond to two kinds of inferences: either a particular kind of left inference, or a cut. The idea is then to build in the natural deduction side this distinction. The result is a calculus with a distinction between *applicative term* and application, together with a distinction between two kinds of application: *head* and *tail* applications.

The isomorphism $\lambda\mathcal{P}h \cong \lambda\mathcal{N}h$ gives insight into the mismatch that exists, even for implication, between sequent calculus and natural deduction - when natural deduction is defined as a system isomorphic to λ.

The paper is organised as follows. First, we study $\lambda\mathcal{P}h$. Then we recall Prawitz's mapping. Next we consider $\lambda\mathcal{N}h$ and the isomorphism $\lambda\mathcal{N}h \cong \lambda\mathcal{P}h$. Finally, we compare our result with the results in [2].

Remark: Because of space limitations, no proofs are given here. They may be found in [7].

Notations and terminology

λ-calculi: α-equivalent terms are seen as equal. Barendregt's variable convention [1] applies to all λ-calculi in this paper. A *value* is a variable or a λ-abstraction. A value application is an application MN where M is a value.

Types: Types are ranged over by A, B, C, D. We just treat intuitionistic implicational logic. Implication is written $A \supset B$.

Contexts: A *context* is a *consistent* set of *declarations* $x : A$. By consistent we mean that if $x : A$ and $x : B$ are in a context, then $A = B$. Contexts are ranged over by Γ. We write $x \in \Gamma$ meaning $x : A \in \Gamma$ for some A. $\Gamma, x : A$ denotes the *consistent union* $\Gamma \cup \{x : A\}$, which means that, if x is already declared in Γ, then it is declared with type A.

Relationship between calculi: We will find several times the following situation. (1) The terms of a calculus λ_1 are also terms of another calculus λ_2. (2) If $t \to u$ in λ_1 then $t \to^+ u$ in λ_2. (3) There is a mapping $p : \lambda_2 \to \lambda_1$ such that (i) $pt = t$, for all t in λ_1 and (ii) $t \to u$ in λ_2 implies $pt \to^* pu$ in λ_1. Such mapping will be called a *projection*. Then, we say that λ_2 is a *conservative extension* of λ_1, because it holds that

$$t \to^* u \text{ in } \lambda_1 \text{ iff } t \to^* u \text{ in } \lambda_2, \text{ for all } t, u \text{ in } \lambda_1.$$

"Only if" follows from (2). As to "if", suppose $t \to^* u$ in λ_2, with t, u in λ_1. Then, by (3-ii), $pt \to^* pu$ in λ_1. But $pt = t$ and $pu = u$, by (3-i).

2 Cut-elimination in the canonical fragment

2.1 The $\lambda\mathcal{P}h$-calculus

The canonical fragment of sequent calculus was rediscovered several times in the 1990's [9, 10, 5, 3]. A derivation is *canonical* if every left inference

$$\frac{\Gamma \vdash A \quad \Gamma, \mathbf{B} \vdash C}{\Gamma, A \supset B \vdash C} \tag{1}$$

occurring in it is canonical, which, in turn, means that the active formula in the right premiss (the bold B in (1)) is main and linear. The exact meaning of "main" and "linear" depends on how the syntax is set up (particularly the structural rules). We immediately move to an example, precisely the calculus of cut-elimination $\lambda\mathcal{P}h$ that we introduce in this paper (see Table 1). For the moment we are interested in the typing rules, given in Table 2.

Table 1. The $\lambda\mathcal{P}h$-calculus

$$\begin{array}{rl} (Terms) & u, v, t ::= x \mid \lambda x.t \mid t(u \cdot l) \\ (Lists) & l, l' ::= [] \mid t :: l \end{array}$$

$$\begin{array}{lrcl} (\beta 1) & (\lambda x.t)(u \cdot []) & \to & subst(u, x, t) \\ (\beta 2) & (\lambda x.t)(u \cdot (v :: l)) & \to & subst(u, x, t)(v \cdot l) \\ (h) & (t(u \cdot l))(u' \cdot l') & \to & t(u \cdot append(l, u' :: l')) \end{array}$$

where

$$\begin{array}{rcl} subst(v, x, x) & = & v \\ subst(v, x, y) & = & y \,, y \neq x \\ subst(v, x, \lambda y.t) & = & \lambda y.subst(v, x, t) \\ subst(v, x, t(u \cdot l)) & = & subst(v, x, t)(subst(v, x, u) \cdot subst(v, x, l)) \end{array}$$

$$\begin{array}{rcl} subst(v, x, u :: l) & = & subst(v, x, u) :: subst(v, x, l) \\ subst(v, x, []) & = & [] \end{array}$$

$$\begin{array}{rcl} append(t :: l, l') & = & t :: append(l, l') \\ append([], l') & = & l' \end{array}$$

Table 2. Typing rules for $\lambda\mathcal{P}h$

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

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

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

We follow the tradition of [9] by using sequents with a *stoup* for enforcing canonical derivations. The stoup is a distinguished position in the LHS of sequents containing at most one formula. If a formula is in the stoup, it is guaranteed to be main and linear. This is easily confirmed in our particular system by inspection of the rules in Table 2. The only inference rules that produce sequents with non-empty stoups are Ax and Lft. On the one hand, the stoup-formulas they produce are main. On the other hand, observe that the only weakened formulas of the system are those of Γ in rules Var and Ax (thus the left A introduced by Ax is not weakened), whereas contraction happens implicitly by identification of Γ's in rules $HeadCut$ and Lft (thus no contraction occurs when $A\supset B$ is introduced by Lft). Therefore, every Lft inference is canonical.

Whether the stoup is empty or not determines two kinds of sequents and the corresponding two kinds of expressions (terms or lists, respectively) that annotate them - see Table 1 for the grammars. Terms may be variables, λ-abstractions or head-cuts $t(u\cdot l)$. When t in $t(u\cdot l)$ is a variable, the head-cut represents another form of left inference, the admissible

$$\frac{\Gamma, x:A\supset B;-\vdash u:A \quad \Gamma, x:A\supset B;B\vdash l:C}{\Gamma, x:A\supset B;-\vdash x(u\cdot l):C}\,Left$$

which is simply an abbreviation of the following instance of $HeadCut$:

$$\frac{\dfrac{}{\Gamma, x:A\supset B;-\vdash x:A\supset B}\,Var \qquad \Gamma, x:A\supset B;-\vdash u:A \quad \Gamma, x:A\supset B;B\vdash l:C}{\Gamma, x:A\supset B;-\vdash x(u\cdot l):C}$$

The difference between $Left$ and Lft is that the $A\supset B$ introduced by the former is not necessarily linear.

The typing derivation of a head-cut has a rigid structure. We consider the particular case $x(u_1\cdot l)$. Then, there are $k\geq 1, u_2,...,u_k$ such that $l=[u_2,...,u_k]$ (if $k=1$ this is $[]$) and the typing derivation looks like

$$
\cfrac{\begin{array}{c}\vdots\\ \Gamma;-\vdash u_1 : A_1\end{array} \qquad \cfrac{\begin{array}{c}\vdots\\ \Gamma;-\vdash u_2 : A_2\end{array} \qquad \begin{array}{c}\cfrac{\begin{array}{c}\vdots\\ \Gamma;-\vdash u_k : A_k\end{array} \qquad \cfrac{}{\Gamma;\mathbf{B}\vdash [] : B}\,Ax}{\Gamma;\mathbf{A_k}\supset\mathbf{B}\vdash [u_k] : B}\,Lft\\ \vdots\\ \Gamma;\mathbf{A_3}\supset ...\supset\mathbf{A_k}\supset\mathbf{B}\vdash [u_3,...,u_k] : B\end{array}}{\Gamma;\mathbf{A_2}\supset ...\supset\mathbf{A_k}\supset\mathbf{B}\vdash [u_2,...,u_k] : B}\,Lft}{\Gamma',x:\mathbf{A_1}\supset\mathbf{A_2}\supset ...\supset\mathbf{A_k}\supset\mathbf{B}\vdash x(u_1\cdot[u_2,...,u_k]) : B}\,Left \tag{2}
$$

where $\Gamma = \Gamma', x : A_1 \supset A_2 \supset ... \supset A_k \supset B$. We call the sequence of bold formulas the *principal path* of this derivation.

2.2 Comparison with Herbelin's system

Herbelin's original $\overline{\lambda}$-calculus differs from $\lambda\mathcal{P}h$ at some points. In the former we find, instead of Ax, a *dereliction* rule

$$
\frac{\Gamma, x : A; A \vdash l : B}{\Gamma, x : A; - \vdash xl : B} \tag{3}
$$

that explicitly takes a formula out of the stoup position; and Herbelin's head-cut has the form

$$
\frac{\Gamma; - \vdash t : A \quad \Gamma; A \vdash l : B}{\Gamma; - \vdash tl : B} \tag{4}
$$

which really looks like a cut. The syntax of terms in $\overline{\lambda}$ is

$$
t ::= xl \,|\, \lambda x.t \,|\, tl \ , \tag{5}
$$

and the syntax of lists is unchanged. Then, our x and $t(u \cdot l)$ are derived as Herbelin's $x[]$ and $t(u :: l)$. From this we see in what sense our head-cut is a cut - it may be seen as an abbreviation for

$$
\cfrac{\Gamma; - \vdash t : A \supset B \qquad \cfrac{\Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\,Lft}{\Gamma; - \vdash t(u :: l) : C}\,(4) \tag{6}
$$

Hence, our head-cuts are *right-permuted* cuts, because, in accordance with (6), their right cut-formulas are main and linear, and, for this reason, they cannot be permuted to the right any further. In Herbelin's system, the head-cut $t[]$ is right-permutable, as the right cut-formulas is introduced by an axiom:

$$t[] \to t \ . \tag{7}$$

Moreover, in Herbelin's system there is a distinction between $(x[])(u :: l)$, a head-cut whose left subderivation is an axiom, and the left inference $x(u :: l)$ that results from eliminating such trivial cut:

$$(x[])(u :: l) \to x(u :: l) \ . \tag{8}$$

This distinction does not exist in our system.

Herbelin's syntax has this advantage: cut=redex. However, there is a price to pay. First, from the point of view of getting a manageable and smooth extension of the λ-calculus, it is simply annoying the need for reduction steps like (7) and (8), as well as not having variables (only derelictions $x[]$). Second, experience has shown that reduction steps like (7) and (8) complicate the meta-theory of these calculi. An example is the study of the spine calculus [2], a variant of both $\overline{\lambda}$ and $\lambda\mathcal{P}h$ to which we will return below.

As to $\lambda\mathcal{P}h$, it is true that cuts $x(u \cdot l)$ are not redexes, but we get a smooth extension of λ. Terms in $\lambda\mathcal{P}h$ are either variables, λ-abstractions or a kind of generalised applications. Indeed, we may think of $t(u \cdot l)$ as t applied to u, with l providing further arguments. Usual application is recovered as $t(u \cdot [])$ - see mapping $\mathcal{G}$ below. Moreover, as Curry-Howard counterparts to the canonical fragment, the only difference between $\overline{\lambda}$ and $\lambda\mathcal{P}h$ is that in the latter some trivial cut-elimination steps become implicit.

2.3 Cut-elimination

We now consider cut-elimination in the canonical fragment. Herbelin's original system [9] was equipped with a stepwise cut-elimination procedure. The goal was to define a procedure simultaneously strongly normalising and complete at least w.r.t. β-normality. However, from the point of view of sequent calculus, this procedure was not very systematic, as, for instance, a cut was not allowed to permute upwards past another cut. More systematic is the proposal in [3]. If we consider, in the terminology of *op. cit.*, every formula to be t-coloured, then the canonical fragment is closed for (the intuitionistic part of) the *t-protocol*. The idea of this procedure is, on the one hand, to give priority to right permutation, when a cut is both right and left permutable - in this sense, we call the t-protocol a *right*-protocol; on the other hand, if we decide to permute a cut to the right (resp. left), then we must perform in a single go, by calling a meta-operator, the *complete* right (resp. left) permutation of that cut. In [6], Herbelin's syntax was equipped with a procedure in the style of the t-protocol.

The cut-elimination procedure represented by the reduction rules of $\lambda\mathcal{P}h$ is a right protocol for eliminating right-permuted cuts. A head-cut $t(u \cdot l)$ is left-permutable iff t is neither a variable nor λ-abstraction. In this case, t is of the form $t'(u' \cdot l')$ and the complete left permutation of $t(u \cdot l)$ is performed in a single go by reduction rule h: $(t'(u' \cdot l'))(u \cdot l)$ reduces to $t'(u' \cdot append(l', u :: l))$. On the other hand, if t is x, we have seen that the head-cut represents a left inference and, hence, is not a redex. Finally, if t is $\lambda x.t'$, then the head-cut is also left-permuted. In this case, both cut-formulas are main and linear and the key-step of cut-elimination applies.

Now, in Herbelin's original system, the key-step is written

$$(\lambda x.t)(u :: l) \rightarrow (t\{x := u\})l \ , \tag{9}$$

where $t\{x := u\}$ represents a *mid-cut*

$$\frac{\Gamma; - \vdash u : A \quad \Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash t\{x := u\} : B} \ .$$

A mid-cut is a cut whose right cut-formula is not in the stoup (as opposed to head-cuts (4)). It is also a right-permutable cut. In $\lambda\mathcal{P}h$ there is no explicit constructor for right-permutable cuts. What we have is the admissible rule

$$\frac{\Gamma; - \vdash u : A \quad \Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash subst(u, x, t) : B} \ .$$

The role of operator *subst* is to perform the complete right permutation of right-permutable cuts. This explains the use of *subst* in reduction rules $\beta 1$ and $\beta 2$. The operator replaces the right-permutable cut generated by the key-step of cut-elimination. Therefore, rules βi aggregate both the key-step and what [3] calls the first structural step (which, in the case of t-coloured formulas, means the complete right permutation). For this reason cut-elimination in $\lambda\mathcal{P}h$ is even more implicit than in the t-protocol.

Finally, the reason for separating two rules β is that, since we do not have Herbelin's head-cut tl (with l allowed to be $[]$), we cannot write in $\lambda\mathcal{P}h$ the key-step as

$$(\lambda x.t)(u :: l) \rightarrow subst(u, x, t)l \ .$$

We have to be sure that there is at least another argument in l for generating a new head-cut, as we do in rule $\beta 2$.

2.4 Relating $\lambda\mathcal{P}h$ and λ

We now sketch the relationship between λ and $\lambda\mathcal{P}h$. There is an injection $\mathcal{G}$ from λ to $\lambda\mathcal{P}h$

$$\begin{aligned} \mathcal{G}x &= x \\ \mathcal{G}(\lambda x.M) &= \lambda x.\mathcal{G}M \\ \mathcal{G}(MN) &= \mathcal{G}M(\mathcal{G}N \cdot []) \ . \end{aligned}$$

This is the traditional translation, going back to [8], that maps elimination inferences to combinations of left inferences and cuts.

Conversely, there is the following projection from $\lambda\mathcal{P}h$ to λ:

$$\begin{array}{rl} \mathcal{Q}x = x & \qquad \mathcal{Q}'(M_1, M_2, []) = M_1 M_2 \\ \mathcal{Q}(\lambda x.t) = \lambda x.\mathcal{Q}t & \qquad \mathcal{Q}'(M_1, M_2, u :: l) = \mathcal{Q}'(M_1 M_2, \mathcal{Q}u, l) \\ \mathcal{Q}(t(u \cdot l)) = \mathcal{Q}'(\mathcal{Q}t, \mathcal{Q}u, l) & \end{array}$$

Use is made of the auxiliar $\mathcal{Q}' : \lambda Terms \times \lambda Terms \times Lists \to \lambda Terms$. If $\mathcal{Q}(t) = M$ and $\mathcal{Q}(u_i) = N_i$, then the idea of $\mathcal{Q}$ is to map $t(u_1 \cdot [u_2, ..., u_k])$ to $MN_1N_2...M_k$.

This mapping sends βi reduction steps to β reduction steps, and collapses h reduction steps.

Proposition 1. *$\lambda\mathcal{P}h$ is a conservative extension of λ.*

3 Prawitz's embedding

We now consider an embedding of natural deduction into sequent calculus due to Prawitz [11]. Contrary to the traditional embedding $\mathcal{G}$, Prawitz's mapping $\mathcal{P}$ sends normal proofs to cut-free derivations, by taking advantage of the structure of normal proofs, uncovered in [11]. It is known that the range of $\mathcal{P}$ is within the canonical fragment [13]. Let us recall why.

$\mathcal{P}$ maps variables and λ-abstractions identically. Now, given an application, one firstly has to unfold it thus:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \mathbf{A_1} \supset ... \supset \mathbf{A_k} \supset \mathbf{B}}\,Var \qquad \begin{array}{c}\vdots\\ \Gamma \vdash N_1 : A_1\end{array}}{\Gamma \vdash xN_1 : \mathbf{A_2} \supset ... \supset \mathbf{A_k} \supset \mathbf{B}}\,Elim \atop {\vdots \atop \Gamma \vdash xN_1...N_{k-1} : \mathbf{A_k} \supset \mathbf{B}} \qquad \begin{array}{c}\vdots\\ \Gamma \vdash N_k : A_k\end{array}}{\Gamma', x : A_1 \supset ... \supset A_k \supset B \vdash xN_1...N_k : \mathbf{B}}\,Elim \tag{10}$$

where $\Gamma = \Gamma', x : A_1 \supset ... \supset A_k \supset B$ and $k \geq 1$ (if $k = 1$, $A_2 \supset ... \supset A_k \supset B$ is just B). The displayed sequence of bold formulas is called the *main branch* [11] of the proof. We now extract from this proof two smaller proofs. The first is just a proof of $\Gamma \vdash N_1 : A_1$ and the second is a proof of

$$\Gamma, z_1 : A_2 \supset ... \supset A_k \supset B \vdash z_1N_2...N_k : B \ ,$$

where z_1 is fresh.

Now apply $\mathcal{P}$ to these two smaller proofs and get two cut-free derivations of sequents $\Gamma \vdash \mathcal{P}(N_1) : A_1$ and $\Gamma, z_1 : A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(z_1N_2...N_k) : B$. Finally, conclude with an application of the left rule:

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash \mathcal{P}(N_1) : A_1\end{array} \qquad \begin{array}{c}\vdots\\ \Gamma, z_1 : \mathbf{A_2} \supset ... \supset \mathbf{A_k} \supset \mathbf{B} \vdash \mathcal{P}(z_1 N_2 ... N_k) : B\end{array}}{\Gamma', x : \mathbf{A_1} \supset \mathbf{A_2} \supset ... \supset \mathbf{A_k} \supset \mathbf{B} \vdash \mathcal{P}(x N_1 ... N_k) : B}\, Left$$

Now, this left inference is canonical. Consider its right active formula (the bold one). It is main, by definition of $\mathcal{P}$, and linear, because z_1 is fresh. Hence, by unfolding $\mathcal{P}(z_1 N_2 ... N_k)$, one sees that the right subderivation above this left inference consists of a stack of left inferences introducing (from top to bottom) the successive linear formulas $B, A_k \supset B, ..., A_2 \supset ... \supset A_k \supset B$. In a syntax like that of $\lambda\mathcal{P}h$, these linear formulas would be in the stoup, instead of being annotated with fresh z's.

Prawitz's embedding may then be defined as the following mapping from λ to $\lambda\mathcal{P}h$.

$$\mathcal{P}(x) = x \tag{11}$$

$$\mathcal{P}(\lambda x.M) = \lambda x.\mathcal{P}(M) \tag{12}$$

$$\mathcal{P}(x N_1 N_2 ... N_k) = x(\mathcal{P}(N_1) \cdot [\mathcal{P}(N_2), ..., \mathcal{P}(N_k)]) \tag{13}$$

A slight generalisation to non-normal λ-terms is available:

$$\mathcal{P}((\lambda x.M) N_1 N_2 ... N_k) = (\lambda x.\mathcal{P}(M))(\mathcal{P}(N_1) \cdot [\mathcal{P}(N_2), ..., \mathcal{P}(N_k)]) \ . \tag{14}$$

$\mathcal{P}$ transforms proof (10) into derivation (2), if we let $\mathcal{P}(N_i) = u_i$. There is a precise correspondence between formulas in the main branch and in the principal path, and between elimination inferences and left inferences (in the sense of both $Left$ or Lft inferences). However, as it were, $\mathcal{P}$ turns the main branch upside down. This is so because the topmost elimination corresponds to the bottom-most left inference; and the elimination just below the former corresponds to the left inference just above the latter, and so on. This phenomenon is the logical counterpart to the inversion of $(...((xN_1)N_2)...N_k)$ as $x(u_1 \cdot (u_2 :: ...(u_k :: [])...))$ operated by $\mathcal{P}$.

4 An extension of the $\boldsymbol{\lambda}$-calculus

4.1 The $\boldsymbol{\lambda\mathcal{N}h}$-calculus

We now compare proof (10) with derivation (2) in order to show that the difference between $\lambda\mathcal{P}h$ and λ is greater than the question of how application is bracketed. There are three observations to make:

1. In (2) two kinds of left inferences, corresponding to distinct constructors in $\lambda\mathcal{P}h$, correspond in (10) to the same kind of elimination inference.

2. Recall that the $Left$ inference at the bottom of (2) is a particular case $x(u_1 \cdot l)$ of head-cut. How about counterparts in λ to general head-cuts $t(u_1 \cdot l)$? The only extension of main branches available is $(\lambda x.M)N_1...N_k$ - a branch topped with a head-redex instead of a head-variable. However, in $t(u_1 \cdot l)$, t is not restricted to variables or λ-abstractions. Actually, t and u_1 form a kind of leftmost application, whereas in λ one only recognises the leftmost end of an applicative term when a value is found.
3. A head-cut cannot be identified with its leftmost application. On the other hand, in λ, $xN_1...N_k$ cannot be distinguished from its right-most application.

From these observations we define $\lambda\mathcal{N}h$, an extension of λ based on two ideas: (i) the inclusion of a constructor for *applicative terms*, distinguished from application. (ii) the separation of two kinds of applications, *head* and *tail* applications, matching the separation between bottom-most inferences of principal paths and Lft inferences, respectively. Head application provides a notion of leftmost application in an applicative term.

The definition of $\lambda\mathcal{N}h$ is given in Tables 3 and 4. There are two classes of expressions: terms and applications, the latter ranged over by A. An applicative term is of the form $app(A)$. Head and tail applications are constructors MN and AN, respectively, both typed with an elimination rule.

Table 3. The $\lambda\mathcal{N}h$-calculus

$$
\begin{array}{rrcl}
(Terms) & M, N & ::= & x \mid \lambda x.M \mid app(A) \\
(Apps) & A & ::= & MN \mid AN
\end{array}
$$

$$
\begin{array}{lrcl}
(\beta 1) & app((\lambda x.M)N) & \to & M[N/x] \\
(\beta 2) & ((\lambda x.M)N)N' & \to & M[N/x]N' \\
(h) & app(A)N & \to & AN
\end{array}
$$

where

$$
\begin{array}{rclrcl}
x[N/x] & = & N & (M_1M_2)[N/x] & = & M_1[N/x]M_2[N/x] \\
y[N/x] & = & y,\ y \neq x & (AM)[N/x] & = & A[N/x]M[N/x] \\
(\lambda y.M)[N/x] & = & \lambda y.M[N/x] & & & \\
(app(A))[N/x] & = & app(A[N/x]) & & &
\end{array}
$$

Defining β is an interesting exercise because the obvious $(\lambda x.M)N \to M[N/x]$ fails. The redex is an application whereas the *contractum* is a term. Rule $\beta 1$ alone cannot reduce $app(((\lambda x.M)N)N')$. Hence, the extra rule $\beta 2$ is needed and we

Table 4. Typing rules for $\lambda\mathcal{N}h$

$$Var\ \frac{}{\Gamma, x : B \vdash x : B} \qquad Intro\ \frac{\Gamma, x : B \vdash M : C}{\Gamma \vdash \lambda x.M : B \supset C} x \notin \Gamma$$

$$App\ \frac{\Gamma \vdash A : B}{\Gamma \vdash app(A) : B}$$

$$HdElim\ \frac{\Gamma \vdash M : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash MN : C}$$

$$TailElim\ \frac{\Gamma \vdash A : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash AN : C}$$

find again a separation of β into two rules. $\beta 1$ is a relation on $Terms$ and $\beta 2$ is a relation on $Apps$.

A h-redex is a head application $A' = M'N'_1$ that is not a value application, hence M' is some applicative term $app(MN_1...N_k)$. With a h-step, the head application $A = MN_1$ at the bottom of M' becomes the head application of the applicative term where A' lives:

$$app(\underline{app(MN_1...N_k)N'_1}...N'_m) \to app(\underline{MN_1}...N_k N'_1...N'_k) \tag{15}$$

We underlined the head applications for highlighting the simplification operated by h. Curiously, this happens in a single step, without any kind of append.

4.2 Relating $\lambda\mathcal{N}h$ and λ

There is an injection ι between λ and $\lambda\mathcal{N}h$ that simply says that every application in λ is head:

$$\begin{aligned} \iota x &= x \\ \iota(\lambda x.M) &= \lambda x.\iota M \\ \iota(MN) &= app(\iota(M)\iota(N)) \end{aligned}$$

Conversely, there is a projection $|\_|$ from $\lambda\mathcal{N}h$ to λ that forgets both the boundaries of applivative terms and whether an application is head or tail:

$$\begin{aligned} |x| &= x & |MN| &= |M||N| \\ |\lambda x.M| &= \lambda x.|M| & |AN| &= |A||N| \\ |app(A)| &= |A| \end{aligned}$$

This projection maps each βi reduction step to a β reduction step, and collapses h reduction steps.

Proposition 2. *$\lambda\mathcal{N}h$ is a conservative extension of λ.*

4.3 Mappings Ψ and Θ

We now codify in a mapping $\Psi : \lambda\mathcal{N}h \rightarrow \lambda\mathcal{P}h$ (see Table 5) the intended correspondences between the eliminations in a main branch and the inferences in a principal path. Since the distinction between head and tail eliminations is built in $\lambda\mathcal{N}h$, the mapping is rather direct. Equation $\Psi(app(A)) = \Psi'(A, [])$ starts the translation of an applicative term. As long as we find tail eliminations, we generate Lft inferences ($\Psi'(AN, l) = \Psi'(A, \Psi N :: l)$). Once we find the head elimination, the bottom-most inference of a principal path (that is, an instance of head-cut) is returned ($\Psi'(MN, l) = \Psi M(\Psi N \cdot l)$). The final effect is that, if $\Psi(M) = t$ and $\Psi(N_i) = u_i$, the applicative term $app(MN_1N_2...N_k)$ is mapped to the head-cut $t(u_1 \cdot [u_2, ..., u_k])$. This is done with the help of an auxiliar $\Psi' : Apps \times Lists \rightarrow Terms$, where $Terms$ is in $\lambda\mathcal{P}h$.

Table 5. From $\lambda\mathcal{N}h$ to $\lambda\mathcal{P}h$

$$\begin{array}{rcl} \Psi(x) &=& x \\ \Psi(\lambda x.M) &=& \lambda x.\Psi M \\ \Psi(app(A)) &=& \Psi'(A, []) \end{array} \qquad \begin{array}{rcl} \Psi'(MN, l) &=& \Psi M(\Psi N \cdot l) \\ \Psi'(AN, l) &=& \Psi'(A, \Psi N :: l) \end{array}$$

Conversely, there is an inverse to Ψ, named Θ and defined in Table 6. Use is made of an auxiliar $\Theta' : Apps \times Lists \rightarrow Terms$, where $Terms$ is in $\lambda\mathcal{N}h$. Again, if $\Theta(t) = M$ and $\Theta(u_i) = N_i$, the idea is simply to map a head-cut $t(u_1 \cdot [u_2, ..., u_k])$ to the applicative term $app(MN_1N_2...N_k)$.

Table 6. From $\lambda\mathcal{P}h$ to $\lambda\mathcal{N}h$.

$$\begin{array}{rcl} \Theta(x) &=& x \\ \Theta(\lambda x.t) &=& \lambda x.\Theta t \\ \Theta(t(u \cdot l)) &=& \Theta'(\Theta t \Theta u, l) \end{array} \qquad \begin{array}{rcl} \Theta'(A, []) &=& app(A) \\ \Theta'(A, u :: l) &=& \Theta'(A\Theta u, l) \end{array}$$

Mappings Ψ and Θ are mutually inverse. Moreover, they map a reduction step in one calculus to a reduction step of the same kind in the other calculus.

Theorem 1 (Isomorphism). *Let* $R \in \{\beta 1, \beta 2, h\}$.

1. $M \rightarrow_R M'$ *in* $\lambda\mathcal{N}h$ *iff* $\Psi M \rightarrow_R \Psi M'$ *in* $\lambda\mathcal{P}h$.
2. $t \rightarrow_R t'$ *in* $\lambda\mathcal{P}h$ *iff* $\Theta t \rightarrow_R \Theta t'$ *in* $\lambda\mathcal{N}h$.

5 Related results

5.1 A lifting

One isomorphism of the kind of Theorem 1 is given in [6] and holds between λ and a calculus named λ_H in *op. cit.* Here we give a slight different (but equivalent) presentation of λ_H that will be named $\lambda\mathcal{P}$. $\lambda\mathcal{P}$ may be defined as the appropriate restriction of $\lambda\mathcal{P}h$ when one only allows head-cuts $t(u \cdot l)$ with t a value, that is, cuts which are both right and left permuted. Terms are then restricted to

$$t, u ::= \ x \mid \lambda x.t \mid x(u \cdot l) \mid (\lambda x.t)(u \cdot l) \ . \tag{16}$$

Instead of full $t(u \cdot l)$, one has the admissible

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

where $insert(u, l, t)$ is defined by

$$\begin{aligned} insert(u, l, x) &= x(u \cdot l) \\ insert(u, l, \lambda x.t) &= (\lambda x.t)(u \cdot l) \\ insert(u, l, x(u' \cdot l')) &= x(u' \cdot append(l', u :: l)) \\ insert(u, l, (\lambda x.t)(u' \cdot l')) &= (\lambda x.t)(u' \cdot append(l', u :: l)) \ . \end{aligned} \tag{17}$$

The idea of the last two equations is to perform implicit h steps. Of course, h reduction does not exist in $\lambda\mathcal{P}$. Moreover, uses of head-cuts in the other reduction rules of $\lambda\mathcal{P}h$ have to be replaced by calls to *insert*. Hence, in addition to $\beta 1$, $\lambda\mathcal{P}$ has the following version of $\beta 2$

$$(\lambda x.t)(u \cdot (v :: l)) \rightarrow insert(v, l, subst(u, x, t)) \ , \tag{18}$$

with *subst* adapted appropriately, as well.

Equations (11) to (14) suggest a match between λ-terms and terms in (16) and indeed $\mathcal{P}$ is an isomorphism between λ and $\lambda\mathcal{P}$. This is proved in [7] following the route of [6]. First, λ is presented in the style of $\lambda\mathcal{N}h$, with a separate class of applications as follows:

$$A ::= \ xN \mid (\lambda x.M)N \mid AN \ .$$

That is, we require M to be a value in every head application MN. We name $\lambda\mathcal{N}$ this subsystem of $\lambda\mathcal{N}h$. Then, appropriate restrictions of Ψ and Θ are defined between $\lambda\mathcal{P}$ and $\lambda\mathcal{N}$, constituting a pair of mutually inverse isomorphisms of normalisation procedures.

We do not give more details about $\lambda\mathcal{N}$ here. However, it is clear now that the isomorphism we introduce in this paper is a "lifting" of the isomorphism proved in [6] - see Fig.1. For the sake of symmetry, Fig.1 mentions an arrow $\mathcal{N} : \lambda \to \lambda\mathcal{N}$. There is one natural arrow of this kind satisfying $\mathcal{N} = \Theta \circ \mathcal{P}$. Details in [7].

Fig. 1. A lifting

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[rr, shift left, leftarrow, "\Psi"] \arrow[rr, shift right, "\Theta"'] & & \lambda\mathcal{N}h \\
& \lambda \arrow[ul, "\mathcal{G}"'] \arrow[ur, "\iota"] \arrow[dl, "\mathcal{P}"] \arrow[dr, "\mathcal{N}"'] & \\
\lambda\mathcal{P} \arrow[rr, shift left, leftarrow, "\Psi"] \arrow[rr, shift right, "\Theta"'] & & \lambda\mathcal{N}
\end{tikzcd}$$

5.2 The spine calculus

We now consider the intuitionistic part of the linear spine calculus of [2], denoted here $\lambda Spine$. The terms of $\lambda Spine$ are those of Herbelin - recall grammar (5): derelictions xl, λ-abstractions $\lambda x.t$ and head-cuts tl. Lists (=spines) as usual. The reduction rules are

$$(\lambda x.t)(u :: l) \to subst(u, x, t)l \tag{19}$$

$$(xl)[] \to xl \tag{20}$$

$$(tl)[] \to tl \tag{21}$$

(20) and (21) are called *nil*-reductions and garbage collect empty lists.

In [2], $\lambda Spine$ is compared with λ and an "isomorphism" is established for well-typed, η-long, nil-normal terms. Typing rules for $\lambda Spine$ are rules (3) and (4) of section 2.2 for dereliction and head-cut, and rules $Right$, Ax and Lft of $\lambda\mathcal{P}h$. However, there is the proviso that in every sequent $\Gamma; B \vdash l : A$, type A must be atomic. Hence, every dereliction xl and every head-cut tl (which correspond to applicative terms) must have atomic type. This entails that, if $(xl)l'$ or $(tl)l'$ are well-typed, then $l' = []$. Hence, up to nil-reduction, the only well-typed head-cuts are of the form $(\lambda x.t)l$ - actually $(\lambda x.t)(u :: l')$, as $(\lambda x.t)[]$ is not well-typed. Thus, basically, we are back to (16), the terms of $\lambda\mathcal{P}$!

Reduction rules of $\lambda Spine$ and $\lambda\mathcal{P}$ look very different. However, in the η-long setting, one does not need to *insert* as in (18). In $\lambda Spine$, take for instance $(\lambda x.t)(u_1 :: u_2 :: [])$. Term t cannot have atomic type, hence it is some λ-abstraction. After one (19)-step we get some $(\lambda y.t')(u_2 :: [])$. It is useless to insert $u_2, []$ in $\lambda y.t'$, because, according to (17), the same head-cut would be returned. Now this t' must have atomic type. Suppose $t' = y[]$. Then, the next (19)-step generates a nil-redex $(u_2[])[]$ (actually two, because u_2 has atomic type). Hence, nil-normal terms are not closed for (19) and the nuisance of nil-reductions is unavoidable. Redex $u_2[]$ was generated because we substituted in a dereliction $y[]$ instead of a variable y. The outer redex $(u_2[])[]$ was created because rule (19) failed to recognise the empty list in $(\lambda y.t')(u_2 :: [])$. In $\lambda\mathcal{P}$, because of the separation of β into $\beta 1$ and $\beta 2$, one does not need to garbage collect such a list.

Acknowledgements: We have used Paul Taylor's macros for typesetting diagrams and proof trees. The author is supported by Fundação para a Ciência e Tecnologia, Portugal.

References

1. H.P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.
2. I. Cervesato and F. Pfenning. A linear spine calculus. Technical Report CMU-CS-97-125, Department of Computer Science, Carnegie Mellon University, 1997.
3. V. Danos, J-B. Joinet, and H. Schellinx. A new deconstructive logic: linear logic. *The Journal of Symbolic Logic*, 62(2):755–807, 1997.
4. R. Dyckhoff and L. Pinto. Cut-elimination and a permutation-free sequent calculus for intuitionistic logic. *Studia Logica*, 60:107–118, 1998.
5. R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212:141–155, 1999.
6. J. Espírito Santo. Revisiting the correspondence between cut-elimination and normalisation. In *Proceedings of ICALP'2000*, volume 1853 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
7. J. Espírito Santo. *Conservative extensions of the λ-calculus for the computational interpretation of sequent calculus*. PhD thesis, University of Edinburgh, 2002. Available at http://www.lfcs.informatics.ed.ac.uk/reports/.
8. G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The collected papers of Gerhard Gentzen*. North Holland, 1969.
9. H. Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.
10. G. Mints. Normal forms for sequent derivations. In P. Odifreddi, editor, *Kreiseliana*, pages 469–492. A. K. Peters, Wellesley, Massachusetts, 1996.
11. D. Prawitz. *Natural Deduction. A Proof-Theoretical Study*. Almquist and Wiksell, Stockholm, 1965.
12. H. Schwichtenberg. Termination of permutative conversions in intuitionistic gentzen calculi. *Theoretical Computer Science*, 212, 1999.

13. A. Troelstra and H. Schwitchtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.