Unity in structural proof theory and structural extensions of the λ-calculus

José Espírito Santo

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

Abstract. We show, for intuitionistic implicational logic, that sequent calculus and natural deduction, if appropriately defined, are isomorphic in the following strong sense: the replacement, in derivations, of left-introductions by eliminations (or vice versa) determines a sound bijection between the sets of derivations that makes the cut-elimination relation and the normalisation relation isomorphic. Natural deduction is defined with an ordinary elimination rule and a primitive rule of substitution, plus an auxiliary sort of derivations to be used as the main premiss of eliminations or the "actual parameter" of substitutions. Normalisation forces auxiliary derivations to start with an assumption, and the "actual parameter" of each substitution to be an elimination. Von Plato's general elimination rule is recovered as a substitution plus an ordinary elimination. In addition, sequent calculus and natural deduction are presented as two extensions of the simply-typed λ-calculus, both elaborating the notion of multiple application, and both embodying a Curry-Howard kind of correspondence. This allows us to describe the isomorphism (and the difference) between the two kinds of systems as a mere inversion of the associativity of applicative terms.

1 Introduction

This paper is a contribution, in the context of intuitionistic implicational logic, to the solution of two closely related problems:

Problem I: Unity in structural proof theory. This is the problem of understanding the differences (and recognizing the similarities) between sequent calculus (a system based on left-introduction and cut) and natural deduction (a system based on elimination and substitution). The problem permanently attracted attention since the birth of structural proof theory [8] until the present day, and is about the following entanglement found in most comparisons: an elimination is seen as a combination of cut and left-introduction, and a left-introduction is seen as a combination of substitution and elimination.

Problem II: Structural extensions of the λ-calculus. This is the problem of extending the simply-typed λ-calculus so as to obtain a meaningful Curry-Howard correspondence with sequent calculus. The extension is structural because types are not touched, only the structure of terms is changed in order to achieve a correspondence with another formalism of structural proof theory. An explicit, early statement of this problem is found in [7].

In [9], Herbelin proved that there is a "λ-calculus structure" for a specific sequent calculus. Such "structure" suggests that the difference between sequent calculus and natural deduction can be reduced to the *associativity* of applicative terms, so that left-associativity (which brings the head of the applicative term to the surface) corresponds to sequent calculus. In the cut-free/normal case, this thesis holds; however, if cuts and non-normal deductions are allowed, the situation is not so clear. In [3, 4] the mapping that inverts the associativity of applicative terms is shown to establish an isomorphism between fragments of Herbelin's sequent calculus and certain *extensions* of natural deduction. "Isomorphism" means sound bijection between the sets of derivations that makes the cut-elimination relation and the normalisation relation isomorphic. However, in those fragments of sequent calculus considered, restrictions are imposed on left-introductions and cuts. In this paper we show that the same mapping is an isomorphism between full sequent calculus and an appropriate, even larger, extension of natural deduction.

Our system of natural deduction has primitive rules of *modus ponens* (= Gentzen's elimination rule) and substitution, plus an auxiliary sort of derivations to be used as the main premiss of eliminations or the "actual parameter" of substitutions. Normalisation forces auxiliary derivations to start with an assumption, and the "actual parameter" of each substitution to be an elimination. Our system extends and refines the system of von Plato [13] because the general elimination rule is decomposed into a substitution and an ordinary elimination.

In addition, both sequent calculus and our system of natural deduction are presented as extensions λ^{G} and λ_{N}, respectively, of the simply-typed λ-calculus. This is possible because there are two parallel, meaningful ways of extending the λ-calculus, both elaborating the notion of multiple application, so that a Curry-Howard kind of correspondence with our proof-systems obtains. Under this presentation, the isomorphism (and the difference) between the two kinds of systems can be described as a mere inversion of the associativity of applicative terms, as desired.

The isomorphism is presented first, in a preliminary version, between systems $\lambda^{\mathbf{gm}} \subset \lambda^{\mathsf{G}}$ and $\lambda_{\mathbf{gs}} \subset \lambda_{\mathsf{N}}$. On the one hand, by this means we hope to motivate better the full systems λ^{G} and λ_{N} and the final version of the isomorphism. On the other hand $\lambda^{\mathbf{gm}}$ and $\lambda_{\mathbf{gs}}$ are interesting on their own. The former is based on the idea of having a separate syntactic class of "applicative contexts", as in [9]. The latter also has a separate syntactic class of "functions", *i.e.* expressions that only occur in the function position of applications. There is an appealing duality here: an applicative context specifies an immediate and linear use (to the expression that goes to the "hole" of the context); a "function" requires such kind of use.

Structure of the paper: Section 2 presents $\lambda^{\mathbf{gm}}$. Section 3 presents $\lambda_{\mathbf{gs}}$. Section 4 (resp. section 5) presents (resp. analyzes) $\lambda^{\mathbf{gm}} \cong \lambda_{\mathbf{gs}}$. Section 6 is about $\lambda^{\mathsf{G}} \cong \lambda_{\mathsf{N}}$. Section 7 concludes. See Fig. 1 for a road-map.

Fig. 1. Sequent calculi (left) and natural deduction systems (right)

$$
\begin{tikzcd}
\lambda^{\mathsf{G}} \arrow[rrrr, shift left, "\Theta"] \arrow[d, dash] & & & & \lambda_{\mathsf{N}} \arrow[llll, shift left, "\Psi"] \arrow[d, dash] \\
\lambda^{\mathbf{gm}} \arrow[rrrr, shift left, "\Theta"] \arrow[dr, dash] & & & & \lambda_{\mathbf{gs}} \arrow[llll, shift left, "\Psi"] \arrow[dl, dash] \\
& \lambda^{\mathbf{g}} \arrow[rr, shift left, "\Theta"] & & \lambda_{\mathbf{g}} \arrow[ll, shift left, "\Psi"] & \\
& & \lambda\mathbf{g} \arrow[ul, "\mathcal{G}"] \arrow[ur, "\iota"'] & &
\end{tikzcd}
$$

Notations: Types (=formulas) are ranged over by A, B, C and generated from type variables using the "arrow type" (=implication), written $A \supset B$. Contexts Γ are consistent sets of declarations $x : A$. "Consistent" means that for each variable x there is at most one declaration in Γ. The notation $\Gamma, x : A$ always denotes a consistent union, that is, one that produces a consistent set.

Natural deduction with general elimination rules [**?**] may be presented as a type system for the λ-calculus with generalized application. The latter is the system ΛJ of [10], which we rename here as $\lambda\mathbf{g}$, for the sake of uniformity with the names of other calculi. Terms of $\lambda\mathbf{g}$ are given by

$$M, N, P \ ::= \ x \mid \lambda x.M \mid M(N, x.P)$$

The typing rule for generalized application is

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash M(N, x.P) : C} \ gElim$$

The λg-calculus has two reduction rules: first, a β-rule

$$(\lambda x.M)(N, y.P) \to [[N/x]M/y]P \ ,$$

where $[N/x]M$ denotes the substitution of N for x in M; second, a rule named π and corresponding to the permutative conversion allowed by general eliminations:

$$M(N, x.P)(N', y.P') \to M(N, x.P(N', y.P')) \ .$$

The usual λ-calculus embeds in $\lambda\mathbf{g}$ by setting $MN = M(N, x.x)$. Likewise, *modus ponens* (=Gentzen's elimination rule for implication) may be seen as the particular case of the *gElim* where $B = C$ and the rightmost premiss is omitted.

2 Sequent calculus

We present the system $\lambda^{\mathbf{gm}}$. It should be understood as an extension of the λ-calculus whose typing rules define a sequent calculus. As an extension of the λ-calculus, it adds to the application constructor the features of generality and multiarity (whence g and m), to be explained soon. As a sequent calculus, $\lambda^{\mathbf{gm}}$ contains, as primitive, cuts of a special form, namely those whose cut-formula in the right premiss is main in a left-introduction.

There are two sorts of expressions in $\lambda^{\mathbf{gm}}$:

$$\begin{array}{ll}\text{(Terms)} & t,u,v ::= x \,|\, \lambda x.t \,|\, tl \\ \text{(Applicative Contexts)} & l ::= u \cdot (x)v \,|\, u :: l\end{array}$$

A term of the form tl is called a *gm-application*. In general, l has the form $u_1 :: ... :: u_{n-1} :: u_n \cdot (x)v$, for some $n \geq 1$. We regard these expressions as *applicative contexts*, following for instance [1]. In tl, think of t as a function and of l as an expression that provides a non-empty list of arguments for t (this is the multiarity feature), plus some "continuation" $(x)v$, specifying what to do after the last argument is consumed (this is the generality feature). Define

$$[u] = u \cdot (x)x \ .$$

Expressions of the form $u_1 :: ... :: u_{n-1} :: [u_n]$ are regarded as lists and writen as $[u_1, ..., u_{n-1}, u_n]$. Accordingly, think of general applicative contexts as "pseudo-lists".

There are two sorts of sequents in $\lambda^{\mathbf{gm}}$, namely $\Gamma \vdash t : A$ and $\Gamma; A \vdash l : B$. The distinguished position in the antecedent of sequents of the latter kind is named the *stoup*. Typing rules are as follows:

$$\frac{}{\Gamma, x : A \vdash x : A}\ Axiom$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\ Right \qquad \frac{\Gamma \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\ lc - Left$$

$$\frac{\Gamma \vdash t : A \quad \Gamma; A \vdash l : B}{\Gamma \vdash tl : B}\ h - Cut \qquad \frac{\Gamma \vdash u : A \quad \Gamma, x : B \vdash v : C}{\Gamma; A \supset B \vdash u \cdot (x)v : C}\ l - Left$$

These rules define a sequent calculus, with two sorts of primitive left-introduction rules and a primitive rule of cut. The system is such that, in any derivation, the stoup always contains a formula that is *main and linear*, that is, main in a left-introduction rule, and introduced without contraction. Given that the cut rule and the left-introduction rules require some of the active formulas and/or the main formula to be in the stoup, these inference rules are of a particular kind. Following Herbelin [9], the cut rule may be called a *head-cut*, as it requires the right cut-formula to be in the stoup. This implies that the cut-formula is always an implication, and that the cut is, in an obvious sense, right-permuted. As to left-introduction rules, they are both linear, in the sense that they both introduce without contraction. In addition, rule $lc - Left$, by requiring its right active formula to be main and linear, is said to be *canonical*, because it is of the restricted form identified in $[9, 2, 11]$.

Let $x : A$ be in Γ. If we cut $\Gamma \vdash x : A$ with $\Gamma; A \vdash l : B$, then we get $\Gamma \vdash xl : B$, that is, the formula A is taken out of the stoup and "given the name x". Using this trick, we get the non-linear versions of the two primitive left-introduction rules: $x(u \cdot (y)v)$ and $x(u :: l)$. This also means that cuts of the form xl are not to be eliminated, as they are the only representation, in the system, of non-linear left-introductions. As to general cuts (*i.e.* cut whose right cut-formula is not in the stoup), the following rule is admissible

$$\frac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash s(u, x, t) : B}$$

Here $s(u, x, t)$ denotes the substitution of u for x in t and is defined by

$$\begin{array}{rlrl} s(u,x,x) &= u & s(u,x,t\cdot(y)v) &= s(u,x,t)\cdot(y)s(u,x,v) \\ s(u,x,y) &= y\,, y \neq x & s(u,x,v :: l) &= s(u,x,v) :: s(u,x,l) \\ s(u,x,\lambda y.t) &= \lambda y.s(u,x,t) & & \\ s(u,x,tl) &= s(u,x,t)s(u,x,l) & & \end{array}$$

There are three reduction rules, aiming at eliminating all head-cuts tl such that t is not a variable:

$$\begin{array}{lc} (\beta 1) & (\lambda x.t)(u \cdot (y)v) \to s(s(u,x,t),y,v) \\ (\beta 2) & (\lambda x.t)(u :: l) \to s(u,x,t)l \\ (\pi) & t(u_1 :: ... :: u_{m-1} :: u_m \cdot (x)v)l \to t(u_1 :: ... :: u_{m-1} :: u_m \cdot (x)vl) \end{array} \ ,$$

where $m \geq 1$. Rule π is global, as the list of the inner cut has to be fully inspected. This rule may be defined as $(tl)l' \to t\,append(l, l')$, with the help of an operator *append* for "appending" pseudo-lists l and l'. Rules $\beta 1$ and $\beta 2$ correspond to the elimination of key cuts, that is, cuts with both cut-formulas main in the premisses. In defining the transformation, we employ substitution at those places where a general cut is required. Rule π corresponds to the left permutation of the outer cut past the inner cut. There is a further reduction rule of a different nature:

$$(\mu) \qquad u \cdot (x)xl \to u :: l, \text{ if } x \notin l$$

If $x \notin l$, $u \cdot (x)xl$ represents a canonical left introduction.

The $\lambda^{\mathbf{gm}}$-calculus is a new presentation of the system $\lambda\mathbf{J^m}$ of [5]. A gm-application $t(u_1 :: ... :: u_{n-1} :: u_n \cdot (x)v)$ is written there as $t(u_1, [u_2, ..., u_n], (x)v)$. This representation brings to the surface of the application the first argument u_1 and the "continuation" $(x)v$. This has advantages, for instance, if one wants to define rules, like the permutative conversions of [5], that inspect the $(x)v$ part. Also rule π becomes local, in contrast with the π rule of $\lambda^{\mathbf{gm}}$. The price to pay for these advantages is that the presentation in *op. cit.* has a hybrid proof-theoretical character. The immediate availability of the first argument implies that the typing rule of the application constructor in *op. cit.* looks like an elimination rule; and we will see that the immediate availability of "continuations" is a distinctive feature of natural deduction. Nevertheless, $\lambda^{\mathbf{gm}}$ inherits the properties of $\lambda\mathbf{J}^{\mathbf{m}}$:

Theorem 1 ([6]). *In $\lambda^{\mathbf{gm}}$, the reduction relation generated on typable terms by $\beta1 \cup \beta2 \cup \pi \cup \mu$ is SN. Moreover, the reduction relation generated on λ^{gm}-terms by any combination of the reduction rules of $\lambda^{\mathbf{gm}}$ is confluent.*

Let t be a λ^{gm}-term. t is a λ^{g}*-term* if it contains no occurrence of construction $u :: l$, and is a $\lambda^{\Box}$*-term* if, in addition, every occurrence of construction $u \cdot (x)v$ in t is of the form $u \cdot (x)x$. There is a bijection $\mathcal{G}$ between λ-terms and $\lambda^{\Box}$-terms, that translates between two ways of writing application: MN and $t[u]$. Proof-theoretically, $\mathcal{G}$ is the traditional mapping of natural deduction to sequent calculus, that interprets eliminations as combinations of a cut and a left-introduction, and whose idea goes back to Gentzen [8]. $\mathcal{G}$ is readily extended to a bijection between λg-terms and λ^{g}-terms, that translates between $M(N, x.P)$ and $t(u \cdot (x)v)$. This is the interpretation of general eliminations as combinations of a cut and a left-introduction found in [**?**]. Now, the set of λ^{g}-terms is closed for substitution and rules $\beta1$ and π. Such set, equipped with such rules, defines a fragment $\lambda^{\mathbf{g}}$ of $\lambda^{\mathbf{gm}}$ that is a mere notational variant of $\lambda\mathbf{g}$.

We may also define in $\lambda^{\mathbf{gm}}$ the following *permutation* rules (where $m \geq 1$):

$$\begin{array}{ll} (p) & t(u_1 :: ... :: u_{m-1} :: u_m \cdot (x)v) \to s(t(u_1 :: ... :: u_{m-1} :: [u_m]), x, v), \;\; v \neq x \\ (q) & t(u_1 :: ... :: u_{m-1} :: u_m :: l) \to t(u_1 :: ... :: u_{m-1} :: [u_m])l \end{array}$$

Then, a λ^{gm}-term t is a $\lambda^{\Box}$-term (morally a λ-term) iff t is a pq-normal form.

3 Natural deduction

We present the system $\lambda_{\mathbf{gs}}$. It should be understood as an extension of the λg-calculus and of natural deduction with general elimination rules. Indeed, there is a subsystem $\lambda_{\mathbf{g}}$ of $\lambda_{\mathbf{gs}}$ which is a copy of $\lambda\mathbf{g}$. The syntax of $\lambda_{\mathbf{gs}}$ has a new feature, named sortedness (whence s). Sortedness means two things: (i) a two-sorted syntax, with a distinguished class of derivations to be used as major premiss of elimination rules; and (ii) the co-existence of two sorts of elimination rules: a general elimination rule and *modus ponens*.

Expressions in $\lambda_{\mathbf{gs}}$ are given by:

$$\begin{array}{lr} \text{(Terms)} & M, N, P ::= x \,|\, \lambda x.M \,|\, app(F, N, (x)P) \\ \text{(Functions)} & F ::= hd(M) \,|\, FN \end{array}$$

This is a syntax with two syntactic classes: terms and *functions*. A term of the form $app(F, N, (x)P)$ may be called either *gs-application* or *outer application*.

Think of this construction as an extension of the generalized application of $\lambda\mathbf{g}$. Indeed, generalized application is recovered as $app(hd(M), N, (x)P)$, because any term M can be coerced to a function $hd(M)$. There is a second kind of application construction, FN, named *inner application*.

The elements of the second syntactic class are named functions because, in expressions, they only occur in the function position of applications. Application FN is inner because, being a function, occurs in the function position of another application. The general form of a function is $hd(M)N_1...N_{m-1}$, for some $m \geq 1$. A function of the form $hd(M)$ is called a *head*. An intuition about functions is that they are expressions which require an immediate and linear use. On the other hand, in $app(F, N, (x)P)$, the use of the application of F to N, specified by the "continuation" $(x)P$ is required neither to be immediate nor linear.

There are two sorts of sequents in $\lambda_{\mathbf{gs}}$, namely $\Gamma \vdash M : A$ and $\Gamma \rhd F : A$. Typing rules are as follows:

$$\frac{}{\Gamma, x : A \vdash x : A}\ Assumption$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ Intro \qquad \frac{\Gamma \rhd F : A \supset B \quad \Gamma \vdash N : A}{\Gamma \rhd FN : B}\ inner-Elim$$

$$\frac{\Gamma \rhd F : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash app(F, N, (x)P) : C}\ outer-Elim$$

$$\frac{\Gamma \vdash M : A}{\Gamma \rhd hd(M) : A}\ Coercion$$

In accordance with what was observed before, this is a natural deduction system extending that of von Plato's. The system contains two primitive elimination rules, *outer elimination* and *inner elimination*, the latter being a form of *modus ponens*. There is a further rule, whose instances are called *coercions*, with *coercion formula* A. The general elimination rule is recovered as a combination of outer elimination and coercion. In addition, a sequent of the form $\Gamma \rhd F : A$ occurs in a derivation of the system iff it occurs as the major premiss of an elimination and A is an implication. The $Coercion$ rule, then, means that any sequent of the first kind can serve as major premiss of an elimination.

There are three reduction rules, aiming at reducing every head to the form $hd(x)$:

$$\begin{array}{ll} (\beta 1) & app(hd(\lambda x.M), N, (y)P) \rightarrow [[N/x]M/y]P \\ (\beta 2) & \qquad\qquad hd(\lambda x.M)N \rightarrow hd([N/x]M) \end{array}$$

and rule (π)

$$\begin{array}{l} app(hd(app(F, N, (x)P))N_1...N_{m-1}, N_m, (y)P') \rightarrow \\ \qquad\qquad \rightarrow app(F, N, (x)app(hd(P)N_1...N_{m-1}, N_m, (y)P')) \ , \end{array}$$

where $m \geq 1$. Notice that rule $\beta 2$ is a relation on functions. As to rule π, if $F = hd(M)$ and $m = 1$, we recognize the π rule of $\lambda\mathbf{g}$. In the general case, rule π is global, because, if we let the redex be $app(F', N_m, (y)P)$, it requires the full inspection of F' until the head emerges. $[N/x]M$ denotes the substitution of N for x in M, and is defined by:

$$\begin{aligned} [N/x]x &= N \\ [N/x]y &= y\,, y \neq x \\ [N/x](\lambda y.M) &= \lambda y.[N/x]M \\ [N/x]app(F, N', (y)P) &= app([N/x]F, [N/x]N', (y)[N/x]P) \\ \\ [N/x]hd(M) &= hd([N/x]M) \\ [N/x](FN') &= ([N/x]F)([N/x]N') \end{aligned}$$

There is a fourth reduction rule, of different nature, named (μ):

$$app(F, N, (x)app(hd(x)N_1 ... N_{m-1}, N_m, (y)P)) \to app(FNN_1 ... N_{m-1}, N_m, (y)P) \ ,$$

where $m \geq 1$. The proviso is $x \notin N_1, ..., N_m, P$. Again, this is a global rule. Let the redex be $app(F, N, (x)P')$. The particular form of P' means that the application of F to N may be considered inner.

We define

$$app(F, N) = app(F, N, (x)x) \ .$$

Let M be a λ_{gs}-term. M is a λ_g*-term* if it contains no occurrence of construction FN, and is a $\lambda_\Box$*-term* if, in addition, every outer-application occurring in t is of the form $app(F, N)$. There is a bijection ι between λ-terms and $\lambda_\Box$-terms, that translates between two ways of writing application: MN and $app(hd(M), N)$. Mapping ι is readily extended to a bijection between λg-terms and λ_g-terms, that translates between $M(N, x.P)$ and $app(hd(M), N, (x)P)$. Now, the set of λ_g-terms is closed for substitution and rules $\beta 1$ and π. Such set, equipped with such rules, defines a fragment $\lambda_\mathbf{g}$ of $\lambda_\mathbf{gs}$ that is a mere notational variant of $\lambda\mathbf{g}$.

M is in $\beta\pi$-nf iff every coercion occurring in M is of the form $hd(x)$. A derivation $\mathcal{D}$ in $\lambda_\mathbf{gs}$ is $\beta\pi$-normal iff every coercion formula occurring in $\mathcal{D}$ is an assumption. In $\lambda_\mathbf{g}$ this gives von Plato's criterion of normality, because in $\lambda_\mathbf{g}$ coercion formula = main premiss of elimination.

We may also define in $\lambda_\mathbf{gs}$ the following *permutation* rules:

$$\begin{array}{ll} (p) & app(F, N, (x)P) \to [app(F, N)/x]P, \ P \neq x \\ (q) & FN \to hd(app(F, N)) \end{array}$$

Then, a λ_{gs}-term t is a $\lambda_\Box$-term (morally a λ-term) iff t is a *pq*-normal form.

4 Isomorphism (first version)

We now define mappings between the set of λ^{gm}-terms and the set of λ_{gs}-terms. Once and for all, variables and λ-abstractions are mapped identically. The question will always be how to map cuts, left introductions and eliminations.

We start with a mapping $\Psi : \lambda_{gs} - Terms \longrightarrow \lambda^{gm} - Terms$. Let $\Psi(M) = t$, $\Psi(N_i) = u_i$ and $\Psi(P) = v$. The idea is to map, say, $app(hd(M)N_1N_2, N_3, (x)P)$ to $t(u_1 :: u_2 :: u_3 \cdot (x)v)$. This is achieved with the help of an auxiliary function $\Psi' : \lambda_{gs} - Functions \times \lambda^{gm} - Lists \longrightarrow \lambda^{gm} - Terms$ as follows:

$$\begin{array}{rlrl} \Psi(x) &= x & \Psi'(hd(M), l) &= (\Psi M)l \\ \Psi(\lambda x.M) &= \lambda x.\Psi M & \Psi'(FN, l) &= \Psi'(F, \Psi N :: l) \\ \Psi(app(F, N, (x)P)) &= \Psi'(F, \Psi N \cdot (x)\Psi P) & & \end{array}$$

Next we consider a mapping $\Theta : \lambda^{gm} - Terms \longrightarrow \lambda_{gs} - Terms$. Let $\Theta(t) = M$, $\Theta(u_i) = N_i$ and $\Theta(v) = P$. The idea is to map, say, $t(u_1 :: u_2 :: u_3 \cdot (x)v)$ to $app(hd(M)N_1N_2, N_3, (x)P)$. This is achieved with the help of an auxiliary function $\Theta' : \lambda_{gs} - Functions \times \lambda^{gm} - Lists \longrightarrow \lambda_{gs} - Terms$ as follows:

$$\begin{array}{rlrl} \Theta(x) &= x & \Theta'(F, u \cdot (x)v) &= app(F, \Theta u, (x)\Theta v) \\ \Theta(\lambda x.t) &= \lambda x.\Theta t & \Theta'(F, u :: l) &= \Theta'(F\Theta u, l) \\ \Theta(tl) &= \Theta'(hd(\Theta t), l) & & \end{array}$$

Theorem 2 (Isomorphism). *Mappings Ψ and Θ are sound, mutually inverse bijections between the set of λ^{gm}-terms and the set of λ_{gs}-terms. Moreover, for each $R \in \{\beta 1, \beta 2, \pi, \mu, p, q\}$:*

1. *$M \rightarrow_R M'$ in $\lambda_{\mathbf{gs}}$ iff $\Psi M \rightarrow_R \Psi M'$ in $\lambda^{\mathbf{gm}}$.*
2. *$t \rightarrow_R t'$ in $\lambda^{\mathbf{gm}}$ iff $\Theta t \rightarrow_R \Theta t'$ in $\lambda_{\mathbf{gs}}$.*

The proof follows the pattern of the proofs of similar results in [3].

When Θ and Ψ are restricted to λ^{g}-terms and λ_{g}-terms, respectively, they translate between two notational variants of generalized application, because $\Theta(t(u \cdot (x)v)) = app(hd(\Theta t), \Theta u, (x)\Theta v)$ and $\Psi(app(hd(M), N, (x)P)) = (\Psi M)(\Psi N \cdot (x)\Psi P)$. In this case, the isomorphism reduces to the expected isomorphism between $\lambda^{\mathbf{g}}$ and $\lambda_{\mathbf{g}}$.

As a corollary to the above theorem, $\lambda_{\mathbf{gs}}$ inherits the properties of $\lambda^{\mathbf{gm}}$.

Corollary 1. *In $\lambda_{\mathbf{gs}}$, the reduction relation generated on typable terms by $\beta 1 \cup \beta 2 \cup \pi \cup \mu$ is SN. Moreover, the reduction relation generated on λ_{gs}-terms by any combination of the reduction rules of $\lambda_{\mathbf{gs}}$ is confluent.*

5 Analyzing the isomorphism

5.1 Sequent calculus vs natural deduction

The presentation of sequent calculus and natural deduction as systems $\lambda^{\mathbf{gm}}$ and $\lambda_{\mathbf{gs}}$, respectively, reduces the difference between the two kinds of systems to the difference between two ways of organizing "applicative terms". By "applicative term" we mean the following data: a function[1] (or head), m arguments ($m \geq 1$) and a continuation (or tail). The notion of applicative term is intended as a common abstraction to the notions of head-cut in $\lambda^{\mathbf{gm}}$

$$t(u_1 :: ... :: u_{m-1} :: u_m \cdot (x)v) \ , \tag{1}$$

and outer application in $\lambda_{\mathbf{gs}}$

$$app(hd(M)N_1...N_{m-1}, N_m, (x)P) \ . \tag{2}$$

When (1) and (2) are regarded in the abstract way of just providing the data that constitutes an applicative term, the only difference that remains between the two expressions is that (1) associates to the right, so that the head t is at the surface and the continuation $(x)v$ is hidden at the bottom of the expression, whereas (2) associates to the left, so that the head $hd(M)$ is hidden at the bottom of the expression, and the continuation $(x)P$ is at the surface. Therefore, at a certain level of abstraction:

1 *The feature that tells sequent calculus from natural deduction is whether applicative terms are left-associative or right-associative.*

The isomorphism between $\lambda^{\mathbf{gm}}$ and $\lambda_{\mathbf{gs}}$ may, then, be described as a mere inversion of the *associativity* of applicative terms.

However, the inversion of the associativity of applicative terms is only observable if $m > 1$, that is, if multiple arguments are present. In that case, there are $m - 1$ occurrences of construction $u :: l$ in (1) and the same number of occurrences of construction FN in (2), and, rougly speaking:

2 *It is the replacement of each occurrence of lc-left-introduction by an occurrence of inner elimination (or vice versa) that causes the inversion of the associativity.*

Indeed, in the case $m = 1$, the translation between (1) and (2) is a translation between two notational variants $t(u \cdot (x)v)$ and $app(hd(M), N, (x)P)$ of a generalized application $M(N, x.P)$. In a generalized application, both head and continuation are at the surface. Actually, as applicative terms, the two notational variants are the same object, a term which has both a sequent calculus organization and a natural deduction organization. The identification of $t(u \cdot (x)v)$ and $app(hd(M), N, (x)P)$ yields the identification of $\lambda^{\mathbf{g}}$ and $\lambda_{\mathbf{g}}$, the two copies of $\lambda\mathbf{g}$. Therefore, at a certain level of abstraction:

3 *Natural deduction with general elimination rules (*i.e. *system $\lambda\mathbf{g}$) is in the intersection between sequent calculus $\lambda^{\mathbf{gm}}$ and the natural deduction system $\lambda_{\mathbf{gs}}$.*

[1] Here "function" is meant in an informal sense, not in the specific sense the word has in the context of system $\lambda_{\mathbf{gs}}$.

5.2 The telescopic effect

Let t_0 (resp. M_0) be the head-cut (1) (resp. outer application (2)) of the last subsection, and let $\Theta t = M$, $\Theta u_i = N_i$ and $\Theta v = P$. There are m choices of F, l such that $M_0 = \Theta'(F, l)$, ranging from the choice $F = hd(M)N_1...N_{m-1}$ and $l = u_m \cdot (x)v$ to the choice $F = hd(M)$ and $l = u_1 :: ...u_{m-1} :: u_m \cdot (x)v$. This last case is particularly important, because the representation of application M_0 as $\Theta'(hd(M), l)$, for such l, brings to the surface the head $hd(M)$. In general, we will use pattern matching of outer application with $\Theta'(hd(M), l)$ to obtain the effect of extracting the head of the application, an effect we call the *telescopic* effect.

Conversely, there are m choices of F, l such that $t_0 = \Psi'(F, l)$. The important one is $F = hd(M)N_1...N_{m-1}$ and $l = u_m \cdot (x)v$, because the representation of the head-cut t_0 as $\Psi'(F, u_m \cdot (x)v)$, for such F, brings to the surface the least accessible information in the cut.

The telescopic effect is useful in making global rules look local. Reduction rules π in $\lambda^{\mathbf{gm}}$ and $\lambda_{\mathbf{gs}}$, and μ in $\lambda_{\mathbf{gs}}$ may be defined as follows (for elegance, we omit the primes):

$$\begin{array}{lc}
(\pi) & \Psi(F, u \cdot (x)v)l \to \Psi(F, u \cdot (x)vl) \\
(\pi) & \Theta(hd(app(F, N, (x)P)), l) \to app(F, N, (x)\Theta(hd(P), l)) \\
(\mu) & app(F, N, (x)\Theta(hd(x), l)) \to \Theta(FN, l), \text{ if } x \notin l
\end{array}$$

Also permutation rules in $\lambda^{\mathbf{gm}}$ receive a neat presentation:

$$\begin{array}{lc}
(p) & \Psi(F, u \cdot (x)v) \to s(\Psi(F, [u]), x, v), \text{ if } v \neq x \\
(q) & \Psi(F, u :: l) \to \Psi(F, [u])l
\end{array}$$

6 Isomorphism (full version)

Cuts in $\lambda^{\mathbf{gm}}$ are such that the right cut-formula is main in a left-introduction. Accordingly, a term tl is necessarily a form of application. In this section we start by defining an extension of $\lambda^{\mathbf{gm}}$, named λ^{G}, where no constraint is imposed on cuts. For this reason, we regard this system as a sort of maximal sequent calculus. Next, we introduce an extension of system $\lambda_{\mathbf{gs}}$, named λ_{N}, so that the isomorphism $\lambda^{\mathbf{gm}} \cong \lambda_{\mathbf{gs}}$ lifts to $\lambda^{\mathsf{G}} \cong \lambda_{\mathsf{N}}$.

The extension of $\lambda^{\mathbf{gm}}$ starts with the following decomposition of linear left-introductions

$$\dfrac{\Gamma \vdash u : A \qquad \dfrac{\Gamma, x : B \vdash v : C}{\Gamma; B \vdash (x)v}\,Selection}{\Gamma; A \supset B \vdash u :: (x)v : C}\,Left$$

We add a new rule *Selection*, whose purpose is to select a formula in the antecedent of sequents and put it in the stoup; and we also add the corresponding construction $(x)v$, similar to the construction $\tilde{\mu}x.v$ of [1]. This has a cascade of consequences: (i) we only need a primitive left rule; (ii) the formula in the stoup is no longer necessarily main and linear, it is just the *selected* formula for being active in left-introduction or cut; (iii) the restrictions that the stoup imposed on cuts and left-introductions in system $\lambda^{\mathbf{gm}}$ fall in the new system; (iv) the second syntactic class of expressions contains *contexts*, which are not necessarily applicative; (v) there is no need for two β-rules, but a new rule is necessary for eliminating cuts of the form $t((x)v)$. The system thus obtained is named λ^{G} and presented in Table 1. Rules π, p and q are presented with telescopic effect, a presentation that requires the definition of the expressions of system λ_{N}, as well as mapping Ψ below.

An extension of $\lambda_{\mathbf{gs}}$ is obtained by decomposing gs-applications as follows :

$$\dfrac{\dfrac{\Gamma \rhd E : A \supset B \quad \Gamma \vdash N : A}{\Gamma \rhd EN : B}\ Elim \qquad \Gamma, x : B \vdash P : C}{\Gamma \vdash \{EN/x\}P : C}\ Subst$$

We add a substitution rule *Subst* and the corresponding substitution construction $\{E/x\}P$. We refer to E, x and P as the *actual parameter*, the *formal parameter* and the *scope* of the substitution, respectively. This addition has a cascade of consequences: (i) we only need a primitive elimination rule; (ii) conclusions of elimination inferences may be the left premiss of substitutions; (iii) expressions of the second syntactic class (ranged over by E) are named *FAP-expressions*, because they may appear in the function position of applications, but also as actual parameters of substitutions (where an immediate and linear use is not guaranteed); (iv) there is no need for two β-rules, but a new rule is necessary for eliminating substitutions of the form $\{hd(M)/x\}P$, whose actual parameter is not an application. The system thus obtained is named λ_{N} and presented in Table 2. In λ_{N}, a derivation is $\beta\pi\sigma$-normal iff every coercion formula occurring in it is both an assumption and the main premiss of an elimination.

Mappings Θ and Ψ are extended in the following way:

$$\begin{aligned} \Theta(x) &= x & \Theta'(E,(x)v) &= \{E/x\}\Theta v \\ \Theta(\lambda x.t) &= \lambda x.\Theta t & \Theta'(E, u :: k) &= \Theta'(E\Theta u, k) \\ \Theta(tk) &= \Theta'(hd(\Theta t), k) & & \end{aligned}$$

$$\begin{aligned} \Psi(x) &= x & \Psi'(hd(M), k) &= (\Psi M)k \\ \Psi(\lambda x.M) &= \lambda x.\Psi M & \Psi'(EN, k) &= \Psi'(E, \Psi N :: k) \\ \Psi(\{E/x\}P) &= \Psi'(E, (x)\Psi P) & & \end{aligned}$$

Theorem 3 (Isomorphism). *Mappings Ψ and Θ are sound, mutually inverse bijections between the set of λ^{G}-terms and the set of λ_{N}-terms. Moreover, for each $R \in \{\beta, \sigma, \pi, \mu, p, q\}$:*

1. *$M \to_R M'$ in λ_{N} iff $\Psi M \to_R \Psi M'$ in λ^{G}.*
2. *$t \to_R t'$ in λ^{G} iff $\Theta t \to_R \Theta t'$ in λ_{N}.*

In λ^{G}, a term t is a pq-normal form iff every cut occurring in it is either of the form $t[u]$ or $t((x)v)$. In λ_{N}, a term M is a pq-normal form iff every substitution

Table 1. The λ^{G}-calculus

Expressions

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

(Contexts) $\quad k ::= (x)v \,|\, u :: k$

Abbreviations $\quad [u] = u :: (z)z$

Reduction rules

$$
\begin{array}{ll}
(\beta) & (\lambda x.t)(u :: k) \to u((x)t)k \\
(\sigma) & t((x)v) \to s(t, x, v) \\
(\pi) & \Psi(E, (x)v)(u :: k) \to \Psi(E, (x)v(u :: k)) \\
(\mu) & (x)xk \to k, \ \text{if } x \notin k
\end{array}
$$

Permutation rules

$$
\begin{array}{ll}
(p) & \Psi(EN, (x)v) \to s(\Psi(E, [\Psi N]), x, v), \ \text{if } v \neq x \\
(q) & \Psi(EN, u :: k) \to \Psi(E, [\Psi N])(u :: k)
\end{array}
$$

where

$$
\begin{array}{rclrcl}
s(u, x, x) &=& u & s(u, x, (y)v) &=& (y)s(u, x, v) \\
s(u, x, y) &=& y \,, y \neq x & s(u, x, v :: k) &=& s(u, x, v) :: s(u, x, k) \\
s(u, x, \lambda y.t) &=& \lambda y.s(u, x, t) & & & \\
s(u, x, tk) &=& s(u, x, t)s(u, x, k) & & &
\end{array}
$$

Sequents $\quad \Gamma \vdash t : A \quad$ and $\quad \Gamma; A \vdash k : B$

Typing / Inference rules

$$
\frac{}{\Gamma, x : A \vdash x : A} \; Axiom
$$

$$
\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B} \; Right
\qquad
\frac{\Gamma \vdash u : A \quad \Gamma; B \vdash k : C}{\Gamma; A \supset B \vdash u :: k : C} \; Left
$$

$$
\frac{\Gamma \vdash t : A \quad \Gamma; A \vdash k : B}{\Gamma \vdash tk : B} \; Cut
\qquad
\frac{\Gamma, x : A \vdash v : B}{\Gamma; A \vdash (x)v : B} \; Selection
$$

Provisos: $x \notin \Gamma$ in rules *Right* and *Select*

Table 2. The λ_{N}-calculus

Expressions

(Terms) $M, N, P ::= x \mid \lambda x.M \mid \{E/x\}P$
(FAP-Expressions) $E ::= hd(M) \mid EN$

Abbreviations $ap(E) = \{E/z\}z$

Reduction rules

$$
\begin{array}{ll}
(\beta) & hd(\lambda x.M)N \to hd(\{hd(N)/x\}M) \\
(\sigma) & \{hd(M)/x\}P \to [M/x]P \\
(\pi) & \Theta(hd(\{E/x\}P), u :: k) \to \{E/x\}\Theta(hd(P), u :: k) \\
(\mu) & \{E/x\}\Theta(hd(x), k) \to \Theta(E, k),\ \text{if } x \notin k
\end{array}
$$

Permutation rules

$$
\begin{array}{ll}
(p) & \{EN/x\}P \to [ap(EN)/x]P,\ \text{if } P \neq x \\
(q) & ENN' \to hd(ap(EN))N'
\end{array}
$$

where

$$
\begin{array}{rlrl}
[N/x]x &= N & [N/x]hd(M) &= hd([N/x]M) \\
[N/x]y &= y\,, y \neq x & [N/x](EN') &= ([N/x]E)([N/x]N') \\
[N/x](\lambda y.M) &= \lambda y.[N/x]M & & \\
[N/x]\{E/y\}P &= \{[N/x]E/y\}[N/x]P & &
\end{array}
$$

Sequents $\Gamma \vdash M : A$ and $\Gamma \rhd E : A$

Typing / Inference rules

$$\frac{}{\Gamma, x : A \vdash x : A}\ Assumption$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ Intro \qquad \frac{\Gamma \rhd E : A \supset B \quad \Gamma \vdash N : A}{\Gamma \rhd EN : B}\ Elim$$

$$\frac{\Gamma \rhd E : A \quad \Gamma, x : A \vdash P : B}{\Gamma \vdash \{E/x\}P : B}\ Subst \qquad \frac{\Gamma \vdash M : A}{\Gamma \rhd hd(M) : A}\ Coercion$$

Provisos: $x \notin \Gamma$ in rules *Intro* and *Subst* occurring in it is either of the form $ap(hd(M)N)$ or $\{hd(M)/x\}P$. Now $t[u]$ and $ap(hd(M)N)$ are two ways of writing ordinary application; and $t((x)v)$ and $\{hd(M)/x\}P$ are two ways of writing explicit substitution [12]. Hence, a term is a *pq*-normal form iff it is (morally) a $\lambda\mathbf{x}$-term.

The embeddings $\lambda\mathbf{g} = \lambda^{\mathbf{g}} \subset \lambda^{\mathbf{gm}} \subset \lambda^{\mathsf{G}}$ and $\lambda\mathbf{g} = \lambda_{\mathbf{g}} \subset \lambda_{\mathbf{gs}} \subset \lambda_{\mathsf{N}}$ give the following decompositions of general elimination:

$$\dfrac{\Gamma \vdash t : A \supset B \qquad \dfrac{\Gamma \vdash u : A \qquad \dfrac{\Gamma, x : B \vdash v : C}{\Gamma ; B \vdash (x)v}\,Selection}{\Gamma ; A \supset B \vdash u :: (x)v : C}\,Left}{\Gamma \vdash t(u :: (x)v) : C}\,Cut$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash M : A \supset B}{\Gamma \rhd hd(M) : A \supset B}\,Coercion \qquad \Gamma \vdash N : A}{\Gamma \rhd hd(M)N : B}\,Elim \qquad \Gamma, x : B \vdash P : C}{\Gamma \vdash \{hd(M)N/x\}P : C}\,Subst$$

In each case, if we disregard the axiom/assumption and (right) introduction rules, all of the inference rules are needed for the decomposition.

Finally, we could make an analysis, in the style of subsection 5.1, of the extended isomorphism introduced in this section, and conclude that it is the replacement of occurrences of *Left* by *Elim* that causes the change of a sequent calculus structure to a natural deduction structure, and vice versa.

7 Conclusions

We summarize our contribution to the solution of Problems I and II, in the context of intuitionistic implicational logic: sequent calculus, equipped with appropriate cut-elimination rules, is isomorphic, in a strong sense, to an appropriate system of natural deduction. The essence of the isomorphism is the replacement, in derivations, of left-introductions by ordinary eliminations (or vice versa). Both systems may be presented as extensions λ^{G} and λ_{N} of the simply typed λ-calculus and, in this case, the difference between the two systems reduces to the associativity of applicative terms.

In addition, we reach the paradoxal conclusion that von Plato's system and thus traditional natural deduction are in the intersection between sequent calculus and our system of natural deduction.

The systems λ^{G} and λ_{N} were introduced with the purpose of answering *fundamental* questions and may be regarded as the end of the road for a certain kind of inquiry. The question of whether they have any practical value, or even good properties, was not the matter here, but deserves to be further investigated.

References

1. P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of International Conference on Functional Programming 2000*. IEEE, 2000.
2. R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212:141–155, 1999.
3. 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/.
4. J. Espírito Santo. An isomorphism between a fragment of sequent calculus and an extension of natural deduction. In M. Baaz and A. Voronkov, editors, *Proceedings of LPAR'02*, volume 2514 of *Lecture Notes in Artificial Intelligence*, pages 354–366. Springer-Verlag, 2002.
5. J. Espírito Santo and Luís Pinto. Permutative conversions in intuitionistic multiary sequent calculus with cuts. In M. Hoffman, editor, *Proc. of TLCA'03*, volume 2701 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, 2003.
6. J. Espírito Santo and Luís Pinto. Confluence and strong normalisation of the generalised multiary λ-calculus. In Ferruccio Damiani Stefano Berardi, Mario Coppo, editor, *Revised selected papers from the International Workshop TYPES 2003*, volume 3085 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
7. J. Gallier. Constructive logics. Part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:248–339, 1993.
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. F. Joachimski and R. Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 42:59–87, 2003.
11. G. Mints. Normal forms for sequent derivations. In P. Odifreddi, editor, *Kreiseliana*, pages 469–492. A. K. Peters, Wellesley, Massachusetts, 1996.
12. K. Rose. Explicit substitutions: Tutorial & survey. Technical Report LS-96-3, BRICS, 1996.
13. J. von Plato. Natural deduction with general elimination rules. *Annals of Mathematical Logic*, 40(7):541–567, 2001.