A λ-calculus Structure Isomorphic to Gentzen-style Sequent Calculus Structure

Hugo Herbelin *

LITP, University Paris 7, 2 place Jussieu, 78252 Paris Cedex 05, France
INRIA-Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France
Hugo.Herbelin@inria.fr

Abstract. We consider a λ-calculus for which applicative terms have no longer the form $(...((u\ u_1)\ u_2)...\ u_n)$ but the form $(u\ [u_1;...;u_n])$, for which $[u_1;...;u_n]$ is a list of terms. While the structure of the usual λ-calculus is isomorphic to the structure of natural deduction, this new structure is isomorphic to the structure of Gentzen-style sequent calculus. To express the basis of the isomorphism, we consider intuitionistic logic with the implication as sole connective. However we do not consider Gentzen's calculus LJ, but a calculus LJT which leads to restrict the notion of cut-free proofs in LJ. We need also to explicitly consider, in a simply typed version of this λ-calculus, a substitution operator and a list concatenation operator. By this way, each elementary step of cut-elimination exactly matches with a β-reduction, a substitution propagation step or a concatenation computation step.
Though it is possible to extend the isomorphism to classical logic and to other connectives, we do not treat of it in this paper.

1 Introduction

By the Curry-Howard isomorphism between natural deduction and simply-typed λ-calculus, and using Prawitz's standard translation [11] of cut-free LJ into natural deduction, we get an assignment of LJ proofs by λ-terms.

Zucker [14] and Pottinger [10] have studied the relations between normalisation in natural deduction and cut-elimination in LJ. They were considering normalisation without paying special attention to the computational cost of the substitution of a proof in place of an hypothesis. But in sequent calculus, among the different uses of the cut rule, there is one which stands for an explicit operator of substitution and among the elementary rules for cut-elimination, there are rules to compute the propagation of substitution. Therefore, Zucker and Pottinger were led to consider proofs up to the equivalence generated by these substitution propagation computation rules.

Here, we consider a λ-calculus with an explicit operator of substitution and with appropriated substitution propagation rules. This allows to have a more

* This research was partly supported by ESPRIT Basic Research Action "Types for Proofs and Programs" and by Programme de Recherche Coordonnées "Mécanisation du raisonnement".

precise correspondence with the elementary rules for cut-elimination. However, there are two problems. The first one is that several cut-free proofs of LJ are associated to the same normal simply-typed λ-terms. An answer to this problem is to rather consider a restriction of LJ, called LJT, having the same structure and same strength as LJ but for which there is a one-to-one correspondence with normal simply-typed terms. The second problem is that Gentzen-style sequent calculus and λ-calculus (or natural deduction) have not the same structure. Consequently, the reduction rules in one and the other calculi do not match. An answer to this second problem is to consider an alternative syntax for λ-calculus of which, this time, the simply-typed fragment is isomorphic to LJT.

Note that a radically different approach of the computational content of Gentzen's sequent calculus appears in Breazu Tanen *et al* [1], Gallier [4] and Wadler [13]. Each of them interprets the left introduction rules of sequent calculus as pattern construction rules.

2 A Motivated Approach to LJT and $\overline{\lambda}$-calculus

2.1 The Sequent Calculus LJ

We consider a version of LJ with the implication as sole connective. The **formulas** are defined by the grammar

$$A ::= X \mid A \to A$$

where X ranges over $\mathcal{V}_F$, an infinite set of which the elements are called **propositional variable names**. In the sequel, we reserve the letters A, B, C, ... to denote formulas.

Sequents of LJ have the form $\Gamma \vdash A$. To avoid the need of a structural rule we define Γ as a set. To avoid confusion between multiple occurrences of the same formula, this set is a set of named formulas. We assume the existence of an infinite set of which the elements are called **names**. Then, a **named formula** is just the pair of a formula and a name. Usually, we do not mention the names of formulas (anyway, no ambiguity occurs in the sequents we consider here).

Under the condition that A, with its name, does not belong to Γ, the notation Γ, A stands for the set-theoretic union of Γ and $\{A\}$.

To avoid the need of a weakening rule, we admit irrelevant formulas in axioms.

The rules of LJ are:

$$\frac{}{\Gamma, A \vdash A}\ Ax \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ Cont$$

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}\ I_L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ I_R$$

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ Cut$$

2.2 The Usual Interpretation of LJ Cut-free Proofs by Normal λ-terms

There is a standard way to interpret cut-free proofs of LJ as λ-terms, see for instance Prawitz [11] or, for a more formal presentation, Zucker [14], Pottinger [10] or Mints [9]. To express the interpretation, it is cumbersome to choose the set of λ-variables names as set of names. We then mention explicitly the name x of a formula A under the form $x:A$. The interpretation is by induction on the proofs and we mention the associated λ-terms on the right of the symbol $\vdash$.

$$\frac{}{\Gamma, x:A \vdash x:A}\ Ax \qquad \frac{\Gamma, x:A, y:A \vdash u:C}{\Gamma, x:A \vdash u\{y:=x\}:C}\ Cont$$

$$\frac{\Gamma \vdash u:A \quad \Gamma, y:B \vdash v:C}{\Gamma, x:A \to B \vdash v\{y:=(x\ u)\}:C}\ I_L \qquad \frac{\Gamma, x:A \vdash u:B}{\Gamma \vdash \lambda x.u:A \to B}\ I_R$$

for which $v\{x:=u\}$ denotes the term v in which each occurrence of x has been replaced by u.

2.3 Towards the Calculus LJT

However different proofs may be associated to the same λ-term. For instance:

$$\frac{\dfrac{\dfrac{}{A, C \vdash A}\ Ax \quad \dfrac{}{A, C, B \vdash B}\ Ax}{A \to B, A, C \vdash B}\ I_L}{A \to B, A \vdash C \to B}\ I_R$$

and

$$\frac{\dfrac{}{A \vdash A}\ Ax \quad \dfrac{\dfrac{}{A, C, B \vdash B}\ Ax}{A, B \vdash C \to B}\ I_R}{A \to B, A \vdash C \to B}\ I_L$$

are both associated to the Church-like typed λ-term $\lambda z:C.(x\ y):C \to B$ for a context in which $x:A \to B$ and $y:A$.

We decide to restrict LJ in order to get a bijective correspondence between normal simply-typed λ-terms and cut-free proofs. For this purpose, we restrict the use of the I_L rule in order to forbid the second proof. The calculus we obtain has two kind of sequents. We call it LJT, since it appears as the intuitionistic fragment of a calculus called LKT and defined by Danos, Joinet, Schellinx [2].

A **sequent** of LJT has either the form $\Gamma; \vdash A$ or the form $\Gamma; A \vdash B$. In both cases, Γ is defined as a set of named formulas. The semi-colon delimits a place on its right. A uniform notation for sequents of LJT is the following one: $\Gamma; \Pi \vdash B$ where Π is a notation to say that the place on the right of the semi-colon may be either empty or filled with one (not named) formula. The idea of using these kinds of sequents comes from Girard [5] who called "stoup" the special place between the symbols ";" and "$\vdash$".

The rules of LJT are

$$\dfrac{}{\Gamma; A \vdash A}\,Ax \qquad \dfrac{\Gamma, A; A \vdash B}{\Gamma, A; \vdash B}\,Cont$$

$$\dfrac{\Gamma; \vdash A \quad \Gamma; B \vdash C}{\Gamma; A \to B \vdash C}\,I_L \quad \dfrac{\Gamma, A; \vdash B}{\Gamma; \vdash A \to B}\,I_R$$

Remarks: 1) With these rules, the first proof above is not directly a proof in the restriction: the axiom rule of LJ has to be encoded in the restriction by an axiom rule followed by a contraction rule.

2)This calculus appears also in Danos *et al* [2] with a slight difference in the treatment of structural rules. Like its classical version LKT, it has been considered by Danos *et al* for its good behaviour w.r.t. embedding into linear logic. The calculus LJT appears also as a fragment of ILU, the intuitionistic neutral fragment of unified logic described by Girard in [6]. The calculus ILU is itself a form of LJ constrained with a stoup, for which Girard pointed out that "the formula [in the stoup] (if there is one) is the analogue of the familiar *head-variable* for typed λ-calculi".

Recently, Mints defined in [9] a notion of normal form for cut-free proofs of LJ which also coincides with the notion of cut-freeness in LJT.

We have also to mention the definition of a cut-free sequent calculus similar to the cut-free LJT in the paper of Howard [12] on the interpretation of natural deduction as a λ-calculus. Howard mentions that the proofs of this cut-free calculus are in one-to-one correspondence with the normal simply-typed λ-terms.

The proofs of the cut-free LJT are effectively in one-to-one correspondence with the normal simply-typed λ-terms. For instance, a normal term of the form $\lambda x_1 \ldots \lambda x_n.(y\ u_1\ \ldots u_p)$ and of type $A_1 \to \ldots \to A_n \to B$, in which y is of type $D = C_1 \to \ldots \to C_p \to B$, is unambiguously associated to a proof of the form

$$\dfrac{\Gamma; \vdash u_p : C_p \qquad \dfrac{}{\Gamma; y : B \vdash y : B}\,Ax}{\Gamma; y_p : C_p \to B \vdash (y_p\ u_p) : B}\,I_L$$

$$\vdots$$

$$\dfrac{\dfrac{\dfrac{\Gamma; \vdash u_1 : C_1 \quad \Gamma; y_2 : C_2 \to \ldots \to C_p \to B \vdash (y_2\ u_2\ \ldots u_p) : B}{\Gamma\ ; y_1 : D \vdash (y_1\ u_1\ \ldots u_p) : B}\,I_L}{\Gamma\ ;\ \vdash (y\ u_1\ \ldots u_p) : B}\,Cont}{\Gamma \backslash \{x_n\};\ \vdash \lambda x_n.(y\ u_1\ \ldots u_p) : A_n \to B}\,I_R$$

$$\vdots$$

$$\dfrac{\Gamma \backslash \{x_2, \ldots, x_n\};\ \vdash \lambda x_2 \ldots \lambda x_n.(y\ u_1\ \ldots u_p) : A_2 \to \ldots \to A_n \to B}{\Gamma \backslash \{x_1, \ldots, x_n\};\ \vdash \lambda x_1 \ldots \lambda x_n.(y\ u_1\ \ldots u_p) : A_1 \to \ldots \to A_n \to B}\,I_R$$

where Γ contains $y : D, x_1 : A_1, \ldots, x_n : A_n$.

Remark: The construction of the applicative part of the term starts from u_p and ends with u_1 in contrast with the usual way of building a term $(u\ u_1\ ...\ u_p)$ in λ-calculus. This is why we have not an exact correspondence with the substitution operator when we consider the cut rule.

2.4 Cut and Reduction Rules: Towards the $\overline{\lambda}$-calculus

According to the place of the cut formula (in the stoup or not), there are two kinds of cut rules in LJT:

head-cut rule mid-cut rule

$$\frac{\Gamma; \Pi \vdash A \quad \Gamma; A \vdash B}{\Gamma; \Pi \vdash B}\ C_H \qquad \frac{\Gamma; \vdash A \quad \Gamma, A; \Pi \vdash B}{\Gamma; \Pi \vdash B}\ C_M$$

for which Π means one or zero formula in the stoup.

The mid-cut rule is naturally interpreted as an operator of explicit substitution:

$$\frac{\Gamma; \vdash v : A \quad \Gamma, x : A; \Pi \vdash u : B}{\Gamma; \Pi \vdash u[x := v] : B}\ C_M$$

A standard way to eliminate cuts is to apply rewriting rules to proofs in order to propagate the cuts towards smaller proofs. Here is an example of such a rewriting rule (we let $C = A_2 \to ... \to A_n \to B$):

$$\frac{\Gamma; \vdash v : A \qquad \dfrac{\Gamma, x : A; \vdash u_1 : A_1 \quad \Gamma, x : A; y : C \vdash (y\ u_2\ ...\ u_n) : B}{\Gamma, x : A; y : A_1 \to C \vdash (y\ u_1\ ...\ u_n) : B}\ I_L}{\Gamma; y : A_1 \to C \vdash (y\ u_1\ ...\ u_n)[x := v] : B}\ C_M$$

reduces to

$$\frac{\dfrac{\Gamma; \vdash v : A \quad \Gamma, x : A; \vdash u_1 : A_1}{\Gamma; \vdash u_1[x := v] : A_1}\ C_M \qquad \dfrac{\Gamma; \vdash v : A \quad \Gamma, x : A; y : C \vdash (y\ u_2\ ...\ u_n) : B}{\Gamma; y : C \vdash (y\ u_2\ ...\ u_n)[x := v] : B}\ C_M}{\Gamma; y : A_1 \to C \vdash (y\ u_1[x := v]\ ...\ u_n)[x := v] : B}\ I_L$$

It seems "natural" that such a rewriting rule is in correspondence with a rule of substitution propagation. But it is not the case. Indeed it corresponds to the reduction of $(y\ u_1\ ...\ u_n)[x := v]$ into $(y\ u_1[x := v]\ ...\ u_n)[x := v]$ while we would like to get $((y\ u_1\ ...u_{n-1})[x := v]\ u_n[x := v])$.

This is because the structure of a proof in sequent calculus is different from the structure of the associated λ-term and this suggests to consider an alternative formalism for the λ-calculus in which an applicative term is no longer of the form $((u\ u_1)...u_n)$, but of the form $(u\ [u_1; ...; u_n])$, i.e. considered as the application of a function to the list of its arguments. We call $\overline{\lambda}$-calculus this alternative formalism for λ-calculus.

2.5 Digression: How to Recover LJ ?

LJT is as strengthful as LJ since a proof of a sequent $\Gamma \vdash A$ in LJ can be compositionnally translated into a proof of $\Gamma; \vdash A$ in LJT. To express the translation, it is more convenient to consider a variant of LJ with the I_L rule and the *Cont* rules mixed (i.e. we assume that $A \to B$ is already in Γ for the second item). We note $\leadsto$ the translation.

$$\frac{}{\Gamma, A \vdash A}\,Ax \quad \leadsto \quad \dfrac{\dfrac{}{\Gamma, A; A \vdash A}\,Ax}{\Gamma, A; \vdash A}\,Cont$$

$$\dfrac{\begin{matrix}\vdots \\ \Gamma \vdash A\end{matrix} \quad \begin{matrix}\vdots \\ \Gamma, B \vdash C\end{matrix}}{\Gamma \vdash C}\,I_L \quad \leadsto \quad \dfrac{\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma; \vdash A\end{matrix} \quad \dfrac{}{\Gamma; B \vdash B}\,Ax}{\Gamma; A \to B \vdash B}\,I_L}{\Gamma; \vdash B}\,Cont \quad \begin{matrix}\vdots \\ \Gamma, B; \vdash C\end{matrix}}{\Gamma; \vdash C}\,C_M$$

$$\dfrac{\begin{matrix}\vdots \\ \Gamma, A \vdash B\end{matrix}}{\Gamma \vdash A \to B}\,I_R \quad \leadsto \quad \dfrac{\begin{matrix}\vdots \\ \Gamma, A; \vdash B\end{matrix}}{\Gamma; \vdash A \to B}\,I_R$$

$$\dfrac{\begin{matrix}\vdots \\ \Gamma \vdash A\end{matrix} \quad \begin{matrix}\vdots \\ \Gamma, A \vdash B\end{matrix}}{\Gamma \vdash B}\,Cut \quad \leadsto \quad \dfrac{\begin{matrix}\vdots \\ \Gamma; \vdash A\end{matrix} \quad \begin{matrix}\vdots \\ \Gamma, A; \vdash B\end{matrix}}{\Gamma; \vdash B}\,C_M$$

Thus we have an interpretation of LJ into LJT. However, following this interpretation, cut-free proofs in LJ may no more be cut-free in LJT.

3 The $\overline{\lambda}$-calculus

3.1 The $\overline{\lambda}$-expressions

We assume the existence of an infinite set $\mathcal{V}$ of which the elements are called **term variables names** and here denoted by the letters x, y, z, ...

The set of $\overline{\lambda}$-expressions, including the **$\overline{\lambda}$-terms** (or shortly **terms**) and the **lists of arguments** are mutually defined by the following grammar for which x ranges over $\mathcal{V}$

$$\begin{array}{ll} \textbf{Terms:} & t ::= (x\ l) \mid (\lambda x.t) \mid (t\ l) \mid (t[x := t]) \\ \textbf{Argument lists:} & l ::= [\,] \mid [t :: l] \mid (l\ @\ l) \mid l[x := t] \end{array}$$

We use the letters t, u, v,... to denote terms and the (possibly quoted) letter l to denote lists of arguments.

The notation $[\,]$ stands for the empty list of arguments and $[t :: l]$ stands for the adjunction of the term t to the list of arguments l, while $(l\ @\ l')$ stands for the explicit concatenation of the lists l and l' of arguments.

The syntax $(t[x := u])$ stands for an operator of explicit substitution in terms (a "`let x=u in t`" operator) and $(l[x := u])$ stands for an operator of explicit substitution in lists of arguments.

We usually abbreviate an argument list $[t_1 :: [... :: [t_n :: [\,]]...]]$ by $[t_1; ...; t_n]$. Terms such as $((...(t\ t_1)\ ...)\ t_n)$ are abbreviated $(t\ t_1\ ...\ t_n)$. Sometimes $(x\ [\,])$ is shortened into x. Also, the expressions $(\lambda x.t)$, $(t[x := u])$ and $(l\ @\ l')$ may be written respectively $\lambda x.t$, $t[x := u]$ and $l\ @\ l'$ when there is no ambiguity.

Subexpressions of $\overline{\lambda}$-expressions are defined as usual, but, in our case, by a simultaneous recursion on terms and arguments lists.

Bound variables are defined as usual. We say that two $\overline{\lambda}$-expressions are α-equal if they differ only in the names (assumed distinct the one from the others) of bound variables. This notion of equality does not affect the structure of expressions and, in the sequel, we consider $\overline{\lambda}$-expressions up to this α-equality.

3.2 Normal $\overline{\lambda}$-expressions

A $\overline{\lambda}$-expression is **normal** if and only if it does not contain any operator of explicit concatenation or explicit substitution and if all applicative subterms are of the form $(x\ l)$ with l normal.

Otherwise said, a $\overline{\lambda}$-expression, is normal if it is construed using this restricted grammar:

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

An approximation of normality is weak normality. A $\overline{\lambda}$-expression is called **weakly normal** if it is of the form $(x\ l)$ or $\lambda x.t$ or $[\,]$ or $[t :: l]$, where t and l denotes respectively any term and any list of arguments.

Remark: Usual λ-calculus can be embedded in $\overline{\lambda}$-calculus, since there is, in $\overline{\lambda}$-calculus, the possibility to consider terms of the form $(...(x\ [u_1])\ ...\ [u_n])$ having a structure similar to the structure of applicative terms in λ-calculus. However, such a $\overline{\lambda}$-term is not normal. Indeed, its normal form is $(x\ [u_1; ...; u_n])$.

3.3 Reduction Rules

The presence of explicit substitution and concatenation operators entails the presence of appropriated reduction rules:

- β-reduction

$$\begin{aligned} (\lambda x.u\ [v :: l]) &\xrightarrow{r} (u[x := v]\ l) && \beta_{cons} \\ (\lambda x.u\ [\,]) &\xrightarrow{r} \lambda x.u && \beta_{nil} \end{aligned}$$

- concatenation of the arguments of a term

$$((x\ l)\ l') \xrightarrow{r} (x\ (l\ @\ l'))\ C_{var}$$

– concatenation computation rules

$$\begin{array}{ll} [u :: l] \ @\ l' \xrightarrow{r} [u :: (l \ @\ l')] & C_{cons} \\ [\,] \ @\ l' \xrightarrow{r} l' & C_{nil} \end{array}$$

– propagation of substitution through weakly normal terms

$$\begin{array}{rll} (x\ l)[x := v] & \xrightarrow{r} (v\ l[x := v]) & S_{yes} \\ (y\ l)[x := v] & \xrightarrow{r} (y\ l[x := v]) & S_{no} \\ (\lambda y.u)[x := v] & \xrightarrow{r} \lambda y.(u[x := v]) & S_{\lambda} \end{array}$$

warning to a possible variable capture in rule S_λ

– propagation of substitution through weakly normal arguments

$$\begin{array}{rll} [\,][x := v] & \xrightarrow{r} [\,] & S_{nil} \\ [u :: l][x := v] & \xrightarrow{r} [u[x := v] :: l[x := v]] & S_{cons} \end{array}$$

If $u \xrightarrow{r} v$, then u is called a **redex**. We note $\xrightarrow{1}$ the one step reduction obtained from $\xrightarrow{r}$ by congruence. Since the system of reduction rules is left linear (if one takes an infinite family of rules S_{yes}, S_{no} and S_λ, one for each possible combination of distinct x and y in $\mathcal{V}$) and without critical pairs, according to Huet [7], $\xrightarrow{1}$ is confluent. We stay unprecised about the α-equality problem stemming from the rule S_λ. Solutions exist, for instance by adding an extra explicit renaming rule to the rewriting system.

Remark: The absence of critical pairs may be quite restricting. For instance, it is not possible to simulate usual β-reduction using these rules for the reason that substitutions are not allowed to go through β-redexes. However, the set of rules is enough to reach a normal form, when this one exists.

4 Cut-elimination in the Calculus LJT

We say that two proofs are **equal** if they differ only by the names of formula in the proved sequent or by addition of irrelevant formulas to the left part of the proved sequents. We consider proofs up to this notion of equality. In particular, if p is a proof of $\Gamma; \Pi \vdash A$, then, for any named formula B not in Γ, p is a proof of $\Gamma, B; \Pi \vdash A$, even if it becomes necessary to change the name of another similarly named occurrence of B throughout p.

4.1 Cut-elimination

Proposition 1. (Strong and confluent cut-elimination)
There exists a confluent system of rewriting rules which allows to derive a cut-free proof of $\Gamma; \Pi \vdash A$ from any proof of the same sequent.

Such a system of rewriting rules is listed hereafter. It is easy to see that it is complete, since it exhausts all possible patterns having a cut rule as head symbol. Its confluence comes from its left linearity (if one takes one different rule for each different variable name) and from the absence of critical pairs, as for the system of reduction rules of the $\overline{\lambda}$-calculus. As for its strong termination, the proof is done in a next section.

Reduction of C_H Rules.

– logical counterpart of β-reduction

$$\dfrac{\dfrac{\Gamma, A; \vdash B}{\Gamma; \vdash A \to B}\,I_R \qquad \dfrac{\Gamma; \vdash A \quad \Gamma; B \vdash C}{\Gamma; A \to B \vdash C}\,I_L}{\Gamma; \vdash C}\,C_H \quad \overset{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\dfrac{\Gamma; \vdash A \quad \Gamma, A; \vdash B}{\Gamma; \vdash B}\,C_M \qquad \Gamma; B \vdash C}{\Gamma; \vdash C}\,C_H$$

$$\dfrac{\dfrac{\Gamma, A; \vdash B}{\Gamma; \vdash A \to B}\,I_R \qquad \dfrac{}{\Gamma; A \to B \vdash A \to B}\,Ax}{\Gamma; \vdash A \to B}\,C_H \quad \overset{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\Gamma, A; \vdash B}{\Gamma; \vdash A \to B}\,I_R$$

– logical counterpart of concatenation of the arguments of a term

$$\dfrac{\dfrac{\Gamma, B; B \vdash A}{\Gamma, B; \vdash A}\,Cont \qquad \Gamma, B; A \vdash C}{\Gamma, B; \vdash C}\,C_H \quad \overset{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\dfrac{\Gamma, B; B \vdash A \quad \Gamma, B; A \vdash C}{\Gamma, B; B \vdash C}\,C_H}{\Gamma, B; \vdash C}\,Cont$$

– logical counterpart of concatenation computation rules

$$\dfrac{\dfrac{\Gamma; \vdash D \quad \Gamma; B \vdash A}{\Gamma; D \to B \vdash A}\,I_L \qquad \Gamma; A \vdash C}{\Gamma; D \to B \vdash C}\,C_H \quad \overset{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\Gamma; \vdash D \qquad \dfrac{\Gamma; B \vdash A \quad \Gamma; A \vdash C}{\Gamma; B \vdash C}\,C_H}{\Gamma; D \to B \vdash C}\,I_L$$

$$\dfrac{\dfrac{}{\Gamma; A \vdash A}\,Ax \qquad \Gamma; A \vdash C}{\Gamma; A \vdash C}\,C_H \quad \overset{\mathrm{LJT}}{\longrightarrow} \quad \Gamma; A \vdash C$$

Reduction of C_M Rules.

– logical counterpart of propagation of substitutions through weakly normal terms

$$\dfrac{\Gamma; \vdash A \qquad \dfrac{\Gamma, A; A \vdash C}{\Gamma, A; \vdash C}\,Cont}{\Gamma; \vdash C}\,C_M \quad \overset{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\Gamma; \vdash A \qquad \dfrac{\Gamma; \vdash A \quad \Gamma, A; A \vdash C}{\Gamma; A \vdash C}\,C_M}{\Gamma; \vdash C}\,C_H$$

$$\dfrac{\Gamma, B; \vdash A \qquad \dfrac{\Gamma, A, B; B \vdash C}{\Gamma, A, B; \vdash C}\,Cont}{\Gamma, B; \vdash C}\,C_M \quad \stackrel{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\dfrac{\Gamma, B; \vdash A \qquad \Gamma, A, B; B \vdash C}{\Gamma, B; B \vdash C}\,C_M}{\Gamma, B; \vdash C}\,Cont$$

$$\dfrac{\Gamma; \vdash A \qquad \dfrac{\Gamma, A, B; \vdash C}{\Gamma, A; \vdash B \to C}\,I_R}{\Gamma; \vdash B \to C}\,C_M \quad \stackrel{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\dfrac{\Gamma, B; \vdash A \qquad \Gamma, A, B; \vdash C}{\Gamma, B; \vdash C}\,C_M}{\Gamma; \vdash B \to C}\,I_R$$

note that, if B already occurs with the same name somewhere in the proof of $\Gamma; \vdash A$, then this latter name has to be changed throughout the proof.

– logical counterpart of propagation of substitution through weakly normal list of arguments

$$\dfrac{\Gamma; \vdash A \qquad \dfrac{}{\Gamma, A; B \vdash B}\,Ax}{\Gamma; B \vdash B}\,C_M \quad \stackrel{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{}{\Gamma; B \vdash B}\,Ax$$

$$\dfrac{\Gamma; \vdash A \qquad \dfrac{\Gamma, A; \vdash B \qquad \Gamma, A; C \vdash D}{\Gamma, A; B \to C \vdash D}\,I_L}{\Gamma; B \to C \vdash D}\,C_M$$

$$\stackrel{\mathrm{LJT}}{\longrightarrow} \quad \dfrac{\dfrac{\Gamma; \vdash A \qquad \Gamma, A; \vdash B}{\Gamma; \vdash B}\,C_M \qquad \dfrac{\Gamma; \vdash A \qquad \Gamma, A; C \vdash D}{\Gamma; C \vdash D}\,C_M}{\Gamma; B \to C \vdash D}\,I_L$$

5 The Assignment of LJT Proofs by $\overline{\lambda}$-expressions

Proofs of LJT are isomorphic to $\overline{\lambda}$-expressions. We show it by first assigning $\overline{\lambda}$-expressions to proofs of LJT. It remains just to check that, through this assignment, the reduction rules for $\overline{\lambda}$-expressions are in exact correspondence with the rewriting rules for proofs of LJT.

To describe the assignment, we identify the set of formula names with the set of $\overline{\lambda}$-term variable names and we write the named formulas under the form $x\!:\!A$. It is also cumbersome to consider arguments lists as applicative contexts:

An **applicative context** is a list of arguments written under the form $(.\ l)$ where $.$ is a special notational symbol. Also, we call **hole declaration** a formula written under the form $.\ : A$.

We express the assignment by judgments.

A **judgement** is something of the form $\Gamma; \Pi \vdash t : A$. In this writing Π is either nothing, in which case t is a term, or a hole declaration in which case t is an applicative context.

Otherwise said, in the assignment, proofs of sequents with an empty stoup are interpreted by terms while proofs of sequents with a non empty stoup are interpreted by applicative contexts.

Applicative context formation | **Term formation**

$$\frac{}{\Gamma;.:A\vdash(.\ [\,]):A}\ Ax \qquad \frac{\Gamma,x:A;.:A\vdash(.\ l):B}{\Gamma,x:A;\vdash(x\ l):B}\ Cont$$

$$\frac{\Gamma;\vdash u:A \quad \Gamma;.:B\vdash(.\ l):C}{\Gamma;.:A\rightarrow B\vdash(.\ [u::l]):C}\ I_L \qquad \frac{\Gamma,x:A;\vdash u:B}{\Gamma;\vdash\lambda x.u:A\rightarrow B}\ I_R$$

$$\frac{\Gamma;.:C\vdash(.\ l):A \quad \Gamma;.:A\vdash(.\ l'):B}{\Gamma;.:C\vdash(.\ (l\ @\ l')):B}\ C_H \qquad \frac{\Gamma;\vdash u:A \quad \Gamma;.:A\vdash(.\ l):B}{\Gamma;\vdash(u\ l):B}\ C_H$$

$$\frac{\Gamma;\vdash u:A \quad \Gamma,x:A;.:C\vdash(.\ l):B}{\Gamma;.:C\vdash(.\ l[x:=u]):B}\ C_M \qquad \frac{\Gamma;\vdash u:A \quad \Gamma,x:A;\vdash v:B}{\Gamma;\vdash v[x:=u]:B}\ C_M$$

Remark: The rules with an non empty stoup are polymorphic in the role of the formula in the stoup. So, there is a strong relation between a judgement

$$\Gamma;.:A_1\rightarrow\ldots\rightarrow A_n\rightarrow B\vdash(.\ [u_1;\ldots;u_n]):B$$

and a judgement

$$\Gamma\vdash[u_1;\ldots;u_n]:A_1\wedge\ldots\wedge A_n$$

where $A_1\wedge\ldots\wedge A_n$ is defined as $\forall B.(A_1\rightarrow\ldots\rightarrow A_n\rightarrow B)\rightarrow B$ (encoding of tuples in second order λ-calculus).

A $\overline{\lambda}$-expression e such that we have $x_1:A_1,\ldots,x_n:A_n\vdash e:A$ for some term variable names x_1, ..., x_n and for some formulas A_1, ..., A_n, A is said **simply-typed of type** A, or shortly, **typable by** A.

6 Strong Termination

By the isomorphism, the strong termination of cut-elimination for LJT (using the above rewriting system) and the strong termination of reduction for typable $\overline{\lambda}$-expressions are equivalent. We show hereafter the strong termination for typable $\overline{\lambda}$-expressions.

Proof of Strong Termination. Let e be a $\overline{\lambda}$-expression and $\stackrel{R}{\rightarrow}$ a notion of reduction. We say that e is strongly normalisable w.r.t. $\stackrel{R}{\rightarrow}$ in the following cases:

- e is not reducible w.r.t. $\stackrel{R}{\rightarrow}$
- for all e' such that $e\stackrel{R}{\rightarrow}e'$, we have e' strongly normalisable

Let e be a $\overline{\lambda}$-expression. If e is typable, then it is strongly normalisable w.r.t. the reduction $\xrightarrow{1}$. To prove that, we prove something stronger, the strong E-normalisability. This latter is preserved by the various operations of $\overline{\lambda}$-expressions construction.

We define a notion of reduction $\xrightarrow{h}$ which removes the head constructor of a $\overline{\lambda}$-expression. The reduction $\xrightarrow{h}$ is defined by the following cases:

$$\begin{array}{ll} \lambda x.u \xrightarrow{h} u & [u :: l] \xrightarrow{h} u \\ (x\ l) \xrightarrow{h} l & [u :: l] \xrightarrow{h} l \end{array}$$

where u ranges over the set of $\overline{\lambda}$-terms and l over the set of argument lists.

We note $\xrightarrow{E}$, and we call E-reduction, the notion of reduction defined by $e \xrightarrow{E} e'$ either because $e \xrightarrow{1} e'$ or because $e \xrightarrow{h} e'$ (without considering the closure of $\xrightarrow{h}$ by congruence). We say that e is **strongly E-normalisable** (shortly SEN) if it is strongly normalisable w.r.t. $\xrightarrow{E}$.

Lemma 2. *If the $\overline{\lambda}$-term u and the argument list l are SEN then $\lambda x.u$, $(x\ l)$ and $[u :: l]$ are SEN.*

Proof. By induction on the proof that u is SEN then by induction on the proof that l is SEN. Let us treat of the case $[u :: l]$. If $[u :: l] \xrightarrow{E} e'$ then, either e' is u or l, in which case, by hypothesis, e' is SEN, or e' is $[u' :: l]$ with $u \xrightarrow{1} u'$, or $[u :: l']$ with $l \xrightarrow{1} l'$ in which cases e' is SEN by induction hypothesis. Therefore, in any case, e reduces to a SEN $\overline{\lambda}$-expression. This implies that e is itself SEN.

Lemma 3. *Let e and u be SEN $\overline{\lambda}$-expressions. If, for all l SEN, the typability of $(u\ l)$ implies that $(u\ l)$ is SEN, then, also the typability of $e[x := u]$ implies that $e[x := u]$ is SEN.*

Proof. It works by induction on the proof that e is SEN then by induction on the proof that u is SEN.

Let us assume that $e[x := u] \xrightarrow{1} w$. If the reduction touchs a redex in u then w has the form $e[x := u']$ with $u \xrightarrow{1} u'$. The proof of SEN for u' is smaller than the one for u, thus, by induction hypothesis, $e[x := u']$ is SEN. Similarly, if the reduction is in e.

It remains the case where $e[x := u]$ is itself a redex and where it is this redex which is reduced. We look at the different possible forms for e.

- The case where e is $(x\ l')$ – in which case w denotes $(u\ l'[x := u])$ – is the more delicate one. But since $e \xrightarrow{h} l'$, the proof of SEN for l' is smaller than the one for e. Therefore, by induction hypothesis, $l'[x := u]$ is SEN. And since we have assumed that for all l SEN, $(u\ l)$ was SEN, we infer that $(u\ l'[x := u])$ is SEN.

- If e is $(y\ l)$ then w is $(y\ l[x := u])$. Here again, $l[x := u]$ is SEN by induction hypothesis. Then, by lemma 2, we get that w is SEN.
- If e is the term $\lambda y.v$, up to a change of the variable name y in $\lambda y.v$ – and this does not change the structure of the proof of SEN –, we may assume that y and x are distinct variable names. We may then affirm that w is $\lambda y.(v[x := u])$. Since $\lambda y.v \xrightarrow{E} v$, by induction hypothesis, $(v[x := u])$ is SEN and by lemma 2, w is SEN.
- If e is $[v :: l]$ then w denotes $[v[x := u] :: l[x := u]]$. But we have both $[v :: l] \xrightarrow{E} v$ and $[v :: l] \xrightarrow{E} l$. Therefore, by induction hypothesis, we have that $v[x := u]$ and $l[x := u]$ are SEN. Then, by lemma 2, we get that w is also SEN.
- If e is $[\,]$ then w is $[\,]$ which is directly SEN.

Thus, whatever the form of e, the reducts of $e[x := u]$ are all SEN. This is enough to say that $e[x := u]$ is SEN.

Lemma 4. *Let A be a formula. Let e be a $\overline{\lambda}$-expression, SEN and typable by A. Let l be a SEN arguments list. If the expression $(e\ l)$ (if e is a $\overline{\lambda}$-term) or the expression e @ l (if e is an arguments list) is typable, then it is SEN.*

Proof. We proceed by induction on A, then on the proof that e is SEN, then on the proof that l is SEN.

Let us assume that $(e\ l) \xrightarrow{1} w$ (if e is a $\overline{\lambda}$-term) or e @ $l \xrightarrow{1} w$ (if e is an arguments list).

If the reduction affects a redex in e then w has the form $(e'\ l)$ or e' @ l with $e \xrightarrow{1} e'$. Since the proof of SEN for e' is smaller that the one for e, by induction hypothesis, w is SEN. Similarly if the reduction is in l.

It may also happen that $(e\ l)$ or e @ l is a redex and that this redex is the reduced one.

- The more delicate case is when e has the form $\lambda x.u$ while l has the form $[v :: l']$. In this case, the type of A has the form $B \to C$, the $\overline{\lambda}$-term v is typable by B and w denotes $(u[x := v]\ l')$. Since B is smaller than A, by induction hypothesis, the typability of $(v\ l'')$ implies that it is SEN whatever l'' SEN. It is then possible to use lemma 3 in order to infer that $u[x := v]$ is SEN. But this latter is typable by C which is also smaller than A. By induction hypothesis, again, $(u[x := v]\ l')$ is SEN.
- If e is $(x\ l')$ then w denotes $(x\ (l'$ @ $l))$. But $(x\ l') \xrightarrow{E} l'$, therefore, by induction hypothesis, $(l'$ @ $l)$ is SEN. By lemma 2, w is SEN.
- If e is $\lambda x.u$ and l denotes $[\,]$ then w is e which, by hypothesis, is SEN.
- If e is $[\,]$ then w denotes l which is directly SEN.
- If e is $[v :: l']$ then w denotes $[v :: (l'$ @ $l)]$. But $[v :: l'] \xrightarrow{E} l'$, therefore, by induction hypothesis, l' @ l is SEN. As for v, it is also SEN by induction hypothesis. Then, by lemma 2, w is SEN.

Thus, whatever reduction of $(e\ l)$ or $e\ @\ l$ we consider, we get a SEN $\overline{\lambda}$-expression. This means that $(e\ l)$ (if e is a $\overline{\lambda}$-term) or $e\ @\ l$ (if e is an arguments list) is SEN.

Proposition 5. *Typable $\overline{\lambda}$-expressions are SEN.*

Proof. Let e be a typable $\overline{\lambda}$-expression. The proof works by induction on e. The cases λv, $(p\ l)$ and $(v :: l)$ come directly from the lemma 2. The cases $(u\ l)$ and $(l\ @\ l')$ come from the lemma 4. As for the cases $v[x := u]$ and $l[x := u]$, they come from the lemma 3 applied to the lemma 4.

The strong E-normalisability directly implies the strong normalisability.

Corollary 6. *Simply-typed $\overline{\lambda}$-expressions are strongly normalizable.*

Remarks: 1) A similar proof has been done by Dragalin [3] for the system of reduction rules given in the seminal paper of Gentzen on the cut-elimination theorem for LK. The difference is that Dragalin's proof does not work by structural induction on the proof of strong E-normalisability, but rather by induction on the length of these proofs. Our proof has been done independently, extending a proof from Coquand that the elimination of cuts according to an outermost strategy of reduction terminates.

Note that this kind of strong cut-elimination proof applies also to non-confluent systems of reduction rules (it is the case of Gentzen's system of reduction rules) but not to system including rules affecting the order of cuts. This is contrast with the cut-elimination procedures that Zucker or Pottinger and have considered.

2) An interesting result would be to prove the strong normalisation of the simply-typed $\overline{\lambda}$-calculus with the additional reduction rule $(\lambda x.t\ u)[y := v] \xrightarrow{r} ((\lambda x.t)[y := v]\ u[y := v])$. As a corollary of this result, we would get the strong normalisation of the usual simply-typed λ-calculus and even the strong normalisation for the simply-typed λ-calculus with an explicit "let _ in _"-like substitution operator (see for instance Lescanne [8]).

Conclusion

The isomorphism known as the Curry-Howard isomorphism expresses a structural correspondence between Hilbert-like axiomatic systems and combinatory logic and between natural deduction and λ-calculus. The isomorphism between LJT and the $\overline{\lambda}$-calculus can be seen as the extension of this correspondence into the framework of sequent calculi and this shows that sequent calculus is no less related to functional features than natural deduction.

Among the different forms of sequent calculi, the calculus LJT has clearly a special place. Since the Modus Ponens rule of intuitionistic natural deduction can be split into a head-cut rule and an implication left introduction rule, LJT can even be seen as a strict refinement of natural deduction. Similarly the $\overline{\lambda}$-calculus can be seen as a strict refinement of the usual λ-calculus, but, in order to make more precise this embedding relation, it would be necessary to extend the strong normalisation of the simply-typed $\overline{\lambda}$-calculus by considering the extra reduction rule $(\lambda x.t\ u)[y := v] \xrightarrow{r} ((\lambda x.t)[y := v]\ u[y := v])$.

Acknowledgements

Simplifications in the proof of strong normalisation are due to Thierry Coquand. I thank also the Paris 7 computer science logic group, Phil Wadler and Viviana Bono for echoes on this work.

References

1. V. Breazu Tanen, D. Kesner, L. Puel: "A typed pattern calculus", *IEEE Symposium on Logic in Computer Science*, Montréal, Canada, June 1993, pp 262-274.
2. V. Danos, J-B. Joinet, H. Schellinx: "LKQ and LKT: Sequent calculi for second order logic based upon dual linear decompositions of classical implication", in *Proceedings of the Workshop on Linear Logic, Cornell*, edited by J-Y. Girard, Y. Lafont, L. Régnier, 1993.
3. A. G. Dragalin: *Mathematical Intuitionism: Introduction to Proof Theory*, Translations of mathematical monographs, Vol 67, Providence, R.I.: American Mathematical Society, 1988.
4. J. Gallier: "Constructive logics, part I: A tutorial on proof systems and typed λ-calculi", *Theoretical Computer Science*, Vol 110, 1993, pp 249-339.
5. J.-Y. Girard: "A new constructive logic: classical logic", *Mathematical Structures in Computer Science*, Vol 1, 1991, pp 255-296.
6. J-Y. Girard: "On the Unity of Logic", *Annals of Pure and Applied Logic*, Vol 59, 1993, pp 201-217.
7. G. Huet: "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems", *Journal of the Association for Computing Machinery*, Vol 27, 1980, pp 797-821.
8. Z. Benaissa, D. Briaud, P. Lescanne, J. Rouyer-Degli, "λv, a calculus of explicit substitutions which preserves strong normalisation", submitted to *Journal of Functional Programming*, 1995.
9. G. Mints: "Normal forms for sequent derivations", Private communication, 1994.
10. G. Pottinger: "Normalization as a homomorphic image of cut-elimination", *Annals of mathematical logic*), Vol 12, 1977, pp 323-357.
11. D. Prawitz: *Natural Deduction, a Proof-Theoretical Study*, Almquist and Wiksell, Stockholm, 1965, pp 90-91
12. W. A. Howard, "The Formulae-as-Types Notion of Constructions", in J.P. Seldin and J.R. Hindley Eds, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980 (unpublished manuscript of 1969).
13. P. Wadler: "A Curry-Howard isomorphism for sequent calculus", Private communication, 1993.
14. J. I. Zucker: "Correspondence between cut-elimination and normalization, part I and II", *Annals of mathematical logic*, Vol 7, 1974, pp 1-156.