The duality of computation

Hugo Herbelin

INRIA-Futurs, Palaiseau, France

(based on joint work with Pierre-Louis Curien)

WoLP 2004 - Créteil

Contents

Introduction

Call-by-name vs Call-by-value

Usual λ-calculus is *call-by-name* λ-calculus

$$t, u \ ::= \ x \mid \lambda x.t \mid t\ u \qquad \textit{Terms}$$

Call-by-name reduction

$$(\lambda x.t)\ u \ \longrightarrow_\beta \ t[x := u]$$

Call-by-value λ-calculus distinguishes values and non-values

$$\begin{array}{lcll} V & ::= & x \mid \lambda x.t & \textit{Values} \\ t, u & ::= & V \mid t\ u & \textit{Terms} \end{array}$$

Call-by-value reduction: arguments must be evaluated before application

$$(\lambda x.t)\ V \ \longrightarrow_{\beta_v} \ t[x := V]$$

Plotkin 1975 defined the call-by-value λ-theory: the λ_v-calculus. It has more normal forms than usual (call-by-name) λ-calculus. E.g.

$$(\lambda x.t)\ (y\ u)$$

is normal as soon as t and u are.

Moggi 1989 defined the computational λ-calculus and introduced a `let-in` construct to characterize the equational call-by-value λ-theory. E.g.

$$NF((\lambda x.t)(y\ u)) \ \text{ is } \ \texttt{let}\ x = y\ u\ \texttt{in}\ t$$

Call-by-value is commonly considered dual to call-by-name, how to syntactically characterize this duality?

Computational content of Gentzen's sequent calculus

Sequent calculus	Natural deduction
$\dfrac{}{\Gamma, A \vdash A}\,Ax$	$\dfrac{}{\Gamma, x : A \vdash x : A}\,Ax$
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to_R$	$\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,\to_I$
$\dfrac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}\,\to_L$	$\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash u : A \to B}{\Gamma \vdash t\ u : B}\,\to_E$
$\dfrac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\,Cont_L$	
$\dfrac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\,Cut$	

Breazu-Tanen–Kesner–Puel 1993: left introduction rules build patterns
Herbelin 1994: left introduction rules (essentially) build *evaluation contexts*

The symmetry of classical Gentzen's sequent calculus

Sequent calculus for classical logic:

- axiom and cut rules are symmetric

- introduction rules for disjunction and conjunction are dual

How does this symmetry show up on the computational interpretation?

See Barbanera-Berardi λ_{sym}-calculus which implements a one-sided version of sequent calculus with disjunction and conjunction (1996).

How does natural deduction interpret classical proofs: Parigot's classical natural deduction

The computational interpretation of classical natural deduction distinguishes the active formula on the right-hand-side of the sequent.

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

$$\frac{\Gamma, x : A \vdash t : B \,;\, \Delta}{\Gamma \vdash \lambda x.t : A \to B \,;\, \Delta} \to_R \qquad \frac{\Gamma \vdash t : A \,;\, \Delta \qquad \Gamma \vdash u : A \to B \,;\, \Delta}{\Gamma \vdash t\ u : B \,;\, \Delta} \to_L$$

$$\frac{\Gamma \vdash t : A \,;\, \alpha : A, \Delta}{\Gamma \vdash [\alpha]t \,;\, \alpha : A, \Delta} Cont_R \qquad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \,;\, \Delta} \mu$$

$$\begin{array}{lll} t, u & ::= & x \mid \lambda x.t \mid t\ u \mid \mu\alpha.c \qquad \textit{Terms} \\ c & ::= & [\alpha]t \qquad\qquad\qquad\qquad \textit{Commands} \end{array}$$

Can we distinguish a formula to the left-hand-side of the sequents?

Sequent calculus has more proofs than natural deduction

$$\mathsf{LJ} \quad \stackrel{\text{many-one}}{\longrightarrow} \quad \mathsf{LJT} \quad \longleftrightarrow \quad \mathsf{NJ}$$

LJT: the image of NJ within LJ

Implicitly considered by Mints.
Identified and named by Danos-Joinet-Schellinx 1995.

To what correspond the extra proofs of LJ?

Results of this talk

- Left introduction rules build evaluation contexts
- Syntactic duality between terms and evaluation contexts

  Syntactic duality between call-by-name and call-by-value
- The “dark” side of sequent calculus is the call-by-value side
- A tree-like presentation of LJ and LK

References: Herbelin 1994 and Curien-Herbelin 2000

I - A computational content for LJ

Interpreting left introduction of implication

How to interpret implication left introduction rule?

$$\frac{\Gamma \vdash t : A \qquad \Gamma, x : B \vdash u : C}{\Gamma, y : A \to B \vdash ? : C}$$

Introduce a distinguished formula on the left-hand-side!
Two kinds of sequents:

$$\begin{array}{rl} \Gamma \vdash A & \text{for terms} \\ \Gamma \, ; A \vdash B & \text{for lists of arguments} \end{array}$$

Interpret a sequent with a distinguished formula on the left-hand-side as a typing rule for list of arguments.

$$\frac{\Gamma \vdash t : A \qquad \Gamma \, ; l : B \vdash C}{\Gamma \, ; t\, l : A \to B \vdash C} \to_L$$

Read $\Gamma \, ; l : A \vdash B$ as the typing derivation of an evaluation context: $\Gamma \, ; [\,] : A \vdash [\,]\, l : B$.

Interpreting the cut rule

The new kind of sequent suggests the following new version of the cut rule

$$\frac{\Gamma \vdash t : A \qquad \Gamma \,; l : A \vdash B}{\Gamma \vdash t \cdot l : B}\,Cut$$

I.e. the cut rule makes a function t interact with a list of arguments l.

Interpreting the axiom rule

The axiom rule naturally builds an empty list of arguments, that we write $\star$.

$$\dfrac{}{\Gamma\,;\star : A \vdash A}\,Ax$$

Interpreting the left contraction rule

We have two choices for interpreting the left contraction rule.

1st choice (Herbelin 1994)

The contraction rule builds normal applicative terms $x\ l$.

$$\frac{\Gamma, x : A\,; l : A \vdash B}{\Gamma, x : A \vdash x \cdot l : B}\,Cont$$

The corresponding syntax of terms ($\overline{\lambda}$-calculus in Herbelin 1994) is

$$\begin{array}{lcll} t, u & ::= & x\ l \mid \lambda x.t \mid t \cdot l & \textit{Terms} \\ l & ::= & \star \mid t\,l & \textit{Lists of arguments} \end{array}$$

Advantage: "normal = cut-free"
Slight disadvantage: x is not a term any longer. Its natural encoding is $x()$.

Interpreting the left contraction rule

2nd choice (Curien-Herbelin 2000)

Add a second axiom rule

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

and derive contraction from cut

$$\frac{\Gamma, A\,;A \vdash B}{\Gamma, A \vdash B} Cont \quad \triangleq \quad \frac{\dfrac{}{\Gamma, x : A \vdash x : A} Ax_R \qquad \Gamma, x : A; l : A \vdash B}{\Gamma, x : A \vdash x \cdot l : B} Cut$$

The corresponding syntax of terms is

$$\begin{array}{lll} t, u & ::= & x \mid \lambda x.t \mid t \cdot l \qquad\qquad \textit{Terms} \\ l & ::= & \star \mid t\, l \qquad\qquad\qquad\quad \textit{Lists of arguments} \end{array}$$

Disadvantage: normal $\neq$ cut-free. We only have: "normal $=$ constructive cut only" where a constructive cut is a cut between an axiom and an introduction rule or between two axiom rules.

Advantage 1: closer from natural deduction and enjoys a tree-form representation.

Advantage 2: scales to a dual classical version

The sequent calculus LJT

The resulting calculus is essentially LJT.

Example: the encoding of $\lambda x.(y\ x) : A \to B$ is $\lambda x.(y \cdot (x\star))$

$$\dfrac{\dfrac{}{\Gamma', x : A \vdash y : A \to B}\,Ax_R \qquad \dfrac{\dfrac{}{\Gamma', x : A \vdash x : A}\,Ax_R \qquad \dfrac{}{\Gamma', x : A\,;\star : B \vdash B}\,Ax_L}{\Gamma', x : A\,; x\star : A \to B \vdash B}\to_L}{\dfrac{\Gamma', x : A \vdash y \cdot (x\star) : B}{\Gamma' \vdash \lambda x.(y \cdot (x\star)) : B}\to_R}\,Cut$$

where Γ' is $\Gamma, y : A \to B$.

LJT is the image of NJ within LJ. Its normal forms are isomorphic to the normal forms of natural deduction.

$$\begin{array}{ccc} x & \longleftrightarrow & x \\ \lambda x.t & \longleftrightarrow & \lambda x.t \\ (...(x\ t_1)\ \ldots t_n) & \longleftrightarrow & x \cdot (t_1 \ldots t_n\,\star) \end{array}$$

Non-normal forms map canonically

$$(...((\lambda x.t)\ t_1)\ \ldots t_n) \longleftrightarrow (\lambda x.t) \cdot (t_1 \ldots t_n\,\star)$$

but there are more non-normal forms in LJT than in NJ

The reduction rules of LJT

$$
\begin{array}{llll}
(\lambda x.t) \cdot (u\, l) & \longrightarrow & t[x := u] \cdot l & \beta\text{-reduction} \\
(t \cdot l) \cdot l' & \longrightarrow & t \cdot (l l') & \text{list concatenation} \\
x \cdot \star & \longrightarrow & x & \text{simplification} \\
\lambda x.t \cdot \star & \longrightarrow & \lambda x.t & \text{simplification}
\end{array}
$$

The tree-like presentation of LJT

Two kinds of nodes

$\vdash t : A$ for terms

$l : A \vdash$ for lists of arguments

Leaves have the form $\vdash t : A$ or $\star : A \vdash$. They can be "bracketed".

Construction rules are

$$\frac{\begin{array}{c}[\vdash x : A]\\ \vdots \\ \vdash t : B\end{array}}{\vdash \lambda x.t : A \to B}\to_R \qquad \frac{\vdash t : A \qquad l : B \vdash}{t\, l : A \to B \vdash}\to_L$$

$$\frac{\vdash t : A \qquad \begin{array}{c}[\star : B \vdash]\\ \vdots \\ l : A \vdash\end{array}}{\vdash t \cdot l : B}\,Cut$$

The tree-like presentation of LJT

Example: the encoding of $\lambda x.(y\ x) : A \to B$ is $\lambda x.y \cdot (x\star)$

$$\dfrac{\vdash y : A \to B \qquad \dfrac{[\vdash x : A] \qquad [\star : B \vdash]}{x\star : A \to B \vdash}\to_L}{\dfrac{\vdash y \cdot (x\star) : B}{\vdash \lambda x.(y \cdot (x\star)) : B}\to_R}Cut$$

The missing part of LJ

Normal forms of LJT have the following form

$$
\dfrac{\dfrac{}{\Gamma \vdash y : \vec{B} \to A}\,Ax_R \qquad \begin{array}{c} \dfrac{}{\Gamma\,;\star : A \vdash A}\,Ax_L \\ \vdots\ \to_L \\ \Gamma\,;l : \vec{B} \to A \vdash A \end{array}}{\begin{array}{c}\Gamma \vdash y \cdot l : A \\ \vdots\ \to_R \\ \Gamma' \vdash \lambda x_1 \ldots x_n . y \cdot l : \vec{C} \to A\end{array}}\,Cut
$$

There is no way to interpret

$$
\dfrac{\Gamma \vdash t : A \qquad \Gamma, x : B \vdash u : C}{\Gamma, y : A \to B \vdash ? : C}
$$

The missing rule is a rule "activating" an hypothesis of the context

$$
\dfrac{\Gamma, x : A \vdash B}{\Gamma\,; ? : A \vdash B}
$$

Such a rule implements a `let-in` evaluation context.

$$
\dfrac{\Gamma, x : A \vdash t : B}{\Gamma\,; \texttt{let}\ x = [\,]\ \texttt{in}\ t : A \vdash B}\,Letin
$$

Then we can derive full LJ implication left introduction.

$$
\dfrac{\Gamma \vdash t : A \qquad \Gamma, x : B \vdash u : C}{\Gamma, y : A \to B \vdash \texttt{let}\ x = y \cdot t\ \texttt{in}\ u : C}
$$

The `let-in` evaluation context

The `let-in` constructor is dual to Parigot's μ operator. Let's write $\tilde{\mu}x.u$ for `let` $x = [\,]$ `in` u. This notation allows to concatenate evaluation contexts and avoids to have to nest them.

The full LJ implication left introduction rewrites as follows:

$$\dfrac{\dfrac{}{\Gamma, y : A \to B \vdash y : A \to B}\,Ax_R \qquad \dfrac{\Gamma \vdash t : A \qquad \dfrac{\Gamma, x : B \vdash u : C}{\Gamma\,; \tilde{\mu}x.u : B \vdash C}\,Letin}{\Gamma\,; t\,\tilde{\mu}x.u : A \to B \vdash C}\,\to_L}{\Gamma, y : A \to B \vdash y \cdot t\,\tilde{\mu}x.u : C}\,Cut$$

The extended syntax is

$$\begin{array}{lcll} t, u & ::= & x \mid \lambda x.t \mid t \cdot l & \textit{Terms} \\ l & ::= & \star \mid t\,l \mid \tilde{\mu}x.t & \textit{Lists of arguments} \end{array}$$

Separating call-by-name and call-by-value reduction

Thanks to the `let-in` construction, β-reduction can be refined into β-contraction and `let-in`-reduction.

$$
\begin{array}{lcll}
(\lambda x.t) \cdot (u\, l) & \longrightarrow & (u \cdot \tilde{\mu} x.t) \cdot l & \beta\text{-contraction} \\
(t \cdot l) \cdot l' & \longrightarrow & t \cdot (l l') & \text{concatenation of evaluation contexts} \\
x \cdot \star & \longrightarrow & x & \text{simplification} \\
\lambda x.t \cdot \star & \longrightarrow & \lambda x.t & \text{simplification} \\
u \cdot \tilde{\mu} x.t & \longrightarrow & t[x := u] & \tilde{\mu}
\end{array}
$$

But there is now a critical pair!

$$
\begin{array}{ccc}
 & (t \cdot l) \cdot \tilde{\mu} x.u & \\
\swarrow & & \searrow \\
t \cdot (l \tilde{\mu} x.u) & & u[x := (t \cdot l)]
\end{array}
$$

Giving preference to the left-hand side implements call-by-value: only values can be substituted within `let-in`, hence, full β-reduction (not just β-contraction) applies only to values.

Giving preference to the right-hand side implements call-by-name as before.

The tree-like presentation of `let-in`

Expressed in tree-form, the `let-in` rule is

$$\dfrac{\begin{array}{c}[\vdash x : A] \\ \vdots \\ \vdash t : B\end{array} \qquad \star : B \vdash}{\tilde{\mu}x.t : A \vdash}\ Letin$$

Thanks to `let-in`, new normal forms occur that are call-by-value normal form. E.g., the derivation of `let` $x = y\ u$ `in` t which indeed corresponds to the full form of implication left introduction.

Remark: For t and u normal, $(y \cdot t\,\star) \cdot \tilde{\mu}x.u$ is not normal, it reduces to the normal form $y \cdot (t\,\tilde{\mu}x.u)$ along call-by-value reduction and to $u[x := (y \cdot t\,\star)]$ along call-by-name reduction.

II - A computational content for LK

Sequent calculus with multiple conclusions

As in Parigot's $\lambda\mu$-calculus, we

- name conclusions with lowercase Greek letters

- distinguish a formula on the right for the rules building terms

- introduce a sequent $c : (\Gamma \vdash \Delta)$ for *commands*, with no distinguished formulae at all

Let's re-interpret the previous formulation of LJ with multiple conclusions.

The *implication right introduction rule* builds a term, it extends directly to multiple conclusions as in $\lambda\mu$-calculus.

$$\frac{\Gamma, x : A \vdash t : B; \Delta}{\Gamma \vdash \lambda x.t : A \to B; \Delta} \to_R$$

The *implication left introduction rule* and the *right axiom rule* extend easily.

$$\frac{\Gamma \vdash t : A\,;\Delta \qquad \Gamma\,; l : B \vdash \Delta}{\Gamma\,; t\,l : A \to B \vdash \Delta} \to_L$$

$$\frac{}{\Gamma, x : A \vdash x : A, \Delta} Ax_R$$

Sequent calculus with multiple conclusions: the Cut rule

The *cut rule* used to build a term. In a symmetric calculus, there is no reason it builds a term rather than an evaluation context. It should build a command.

$$\frac{\Gamma \vdash t : A; \Delta \qquad \Gamma\, ; l : A \vdash \Delta}{t \cdot l : (\Gamma \vdash \Delta)}\, Cut$$

Sequent calculus with multiple conclusions: many continuation variables

With multiple conclusions, it is possible to switch the evaluation contexts.
Let's add the μ rule as in $\lambda\mu$-calculus.

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \,;\, \Delta}\,\mu$$

We now need to re-interpret the meaning of the empty list of arguments. In LJ, it tells that there are no more argument to apply and that the computation can go on with the surrounding context. The direct generalisation to sequent calculus with multiple conclusion is to go on with a given *named* evaluation context denoted by a continuation variable.

This leads to the following rephrasing of the left axiom rule, dual to the right axiom rule.

$$\frac{}{\Gamma; \alpha : A \vdash \alpha : A, \Delta}\,Ax_L$$

Finally, we can restate the `let-in` rule. By symmetry with the μ rule, we make it take a command as argument.

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \,;\, \tilde{\mu}x.c : A \vdash \Delta}\,\tilde{\mu}$$

Re-interpreting LJ within LK

LJ is LK with a single continuation variable, written $\star$ (like the empty list of argument). LK gives a finer analysis of the previous constructions as follows:

LJ	LK
x	x
$\lambda x.t$	$\lambda x.t$
$t \cdot l$	$\mu \star .(t \cdot l)$
$\star$	$\star$
$\tilde{\mu} x.t$	$\tilde{\mu} x.(t \cdot \star)$

Injecting natural deduction

Injecting usual λ-calculus is as before except that we cannot just use the single continuation variable $\star$. We need a fresh continuation variable α.

NJ	LK
x	x
$\lambda x.t$	$\lambda x.t$
$x\, t_1 \dots t_n$	$\mu\alpha.(x \cdot t_1 \dots t_n\, \alpha)$
$(\lambda x.t)\, t_1 \dots t_n$	$\mu\alpha.((\lambda x.t) \cdot t_1 \dots t_n\, \alpha)$

On normal forms, the injection from $\lambda\mu$-calculus is

CND	LK
$\lambda x_1 \dots x_m.\mu\gamma[\beta](y\, t_1 \dots t_n)$	$\lambda x_1 \dots x_m.\mu\gamma.(y \cdot (t_1 \dots t_n\, \beta))$
$\lambda x_1 \dots x_m.(y\, t_1 \dots t_n)$	$\lambda x_1 \dots x_m.\mu\alpha.(y \cdot (t_1 \dots t_n\, \alpha))$

Reduction rules

The reduction rules are symmetrical in LK:

$$
\begin{array}{lcll}
(\lambda x.t) \cdot (u\, l) & \longrightarrow & u \cdot \tilde{\mu} x.(t \cdot l) & \beta\text{-contraction} \\
(\mu \alpha.t) \cdot l & \longrightarrow & t[\alpha := l] & \mu \\
u \cdot (\tilde{\mu} x.t) & \longrightarrow & t[x := u] & \tilde{\mu} \\
\mu \alpha.(V \cdot \alpha) & \longrightarrow & V & \text{simplification } (\alpha \text{ not free in } V) \\
\tilde{\mu} x.(x \cdot L) & \longrightarrow & L & \text{simplification } (x \text{ not free in } L)
\end{array}
$$

where

$$
\begin{array}{lcll}
V & ::= & x \mid \lambda x.t & \textit{Values} \\
L & ::= & \alpha \mid tl & \textit{Applicative evaluation contexts}
\end{array}
$$

Note that the substitution work is done by the rules μ and $\tilde{\mu}$ only.
The critical pair now expresses symmetrically.

$$
\begin{tikzcd}
& (\mu \alpha.c) \cdot (\tilde{\mu} x.c') \arrow[dl] \arrow[dr] & \\
c[\alpha := \tilde{\mu} x.c'] & & c'[x := \mu \alpha.c]
\end{tikzcd}
$$

Call-by-value and call-by-name

Giving preference to the left-hand side implements call-by-value. Typically, a configuration like

$$(\lambda x.t)(\mu\alpha.c)$$

is interpreted in LK as

$$\mu\beta.((\lambda x.t)\cdot((\mu\alpha.c)\beta))$$

which reduces to

$$\begin{array}{ccc} & \mu\beta.((\mu\alpha.c)\cdot\tilde{\mu}x.(t\cdot\beta)) & \\ CBV \swarrow & & \searrow CBN \\ \mu\beta.c[\alpha := \tilde{\mu}x.(t\cdot\beta)] & & t[x := \mu\alpha.c] \end{array}$$

Example: reduction of $\mu\beta.[\beta]((\lambda x.3)(\mu\alpha.[\beta]4))$

$$\begin{array}{ccc} & \mu\beta.(\lambda x.3\cdot((\mu\alpha.[\beta]4)\,\beta)) & \\ CBV \swarrow & & \searrow CBN \\ 4 & & 3 \end{array}$$

Summary

The resulting sequent calculus is essentially LK in the sense that there is a correspondence with Gentzen's LK that preserves the structure of introduction rules.

$$\frac{}{\Gamma; \alpha : A \vdash \alpha : A, \Delta}\,Ax_L \qquad \frac{}{\Gamma, x : A \vdash x : A, \Delta}\,Ax_R$$

$$\frac{\Gamma \vdash t : A \,;\Delta \qquad \Gamma\,; l : B \vdash \Delta}{\Gamma\,; t\,l : A \to B \vdash \Delta}\,{\to_L} \quad \frac{\Gamma, x : A \vdash t : B; \Delta}{\Gamma \vdash \lambda x.t : A \to B; \Delta}\,{\to_R}$$

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma\,; \tilde{\mu}x.c : A \vdash \Delta}\,\tilde{\mu} \qquad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A\,; \Delta}\,\mu$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma\,; l : A \vdash \Delta}{t \cdot l : (\Gamma \vdash \Delta)}\,Cut$$

The $\overline{\lambda}\mu\tilde{\mu}$-calculus

$$\begin{array}{lcll}
t, u & ::= & x \mid \lambda x.t \mid \mu\alpha.c & \textit{Terms} \\
l & ::= & \alpha \mid t\,l \mid \tilde{\mu}x.c & \textit{Evaluation contexts} \\
c & ::= & t \cdot l & \textit{Commands}
\end{array}$$

The tree-like presentation of LK

The resulting sequent calculus is essentially LK in the sense that removing the ";" yields a proof of LK extended with the identity rule. It enjoys a tree-like presentation.

$$\frac{\vdash t : A \qquad l : B \vdash}{t\, l : A \to B \vdash} \to_L \qquad \frac{\begin{array}{c}[\vdash x : A] \\ \vdots \\ \vdash t : B\end{array}}{\vdash \lambda x.t : A \to B} \to_R$$

$$\frac{\begin{array}{c}[\vdash x : A] \\ \vdots \\ c\end{array}}{\tilde{\mu}x.c : A \vdash} \tilde{\mu} \qquad \frac{\begin{array}{c}[\alpha : A \vdash] \\ \vdots \\ c\end{array}}{\vdash \mu\alpha.c : A} \mu$$

$$\frac{\vdash t : A \qquad l : A \vdash}{t \cdot l} Cut$$