Investigations into the Duality of Computation

Hugo Herbelin
(INRIA-Futurs)

International Workshop on Higher-Order Rewriting

Seattle, August 15, 2006

(based on joint work with Pierre-Louis Curien [ICFP' 00])

Outline

Introduction

An analysis of the underlying term structure of sequent calculus

- System $\mu\tilde{\mu}$: the core of computation
- Adding connectives
- Typing system $\mu\tilde{\mu}$
- Sequent calculus presented with implicit context
- Back to (untyped) system $\mu\tilde{\mu}$: the call-by-name vs call-by-value dilemma

Further investigations

- “Canonical” call-by-name and call-by-value λ-calculi
- Applications to the study of call-by-value
- A dual to call-by-need
- Sequent calculus and the proof-as-strategy approach
- Sequent calculus and abstract machines
- A limit to duality: dependent types

Conclusion and problems

General research framework

proof theory

theory of computation

proof-as-program correspondence
Curry [1958] Howard [1969] (de Bruijn [1968])

intuitionistic natural deduction Gentzen [1936] = (call-by-name) λ-calculus Church [1936]

↓

classical logic = control operators

↓

sequent calculus Gentzen [1936] = **?**

left/right symmetry

strategies in game-theoretic models of computation

abstract machines (redex at root)

duality ⟶ *a tool to revisit proof and computation theories*

The $\mu\tilde{\mu}$-subsystem: the core of computation

Computational properties

- syntactic emphasis of the call-by-name vs call-by-value duality,
- syntactic emphasis of a duality between term and evaluation contexts that interact to produce results,
- accepts extensions made by specific sets of interacting constructors,
- condenses all the computational aspects of its extensions

Syntax

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha$

Semantics

$$\begin{array}{llll} (\mu) & \langle \mu\alpha.c \| e \rangle & \rightarrow & c[\alpha \leftarrow e] \\ (\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v] \end{array}$$

- a good tool to better understand the theory of call-by-value λ-calculus
- a uniform analysis of η-conversion,
- a close connection with abstract environment machines (redexes are at the head of the expressions),

Proof-theoretical properties

- full Curry-Howard correspondence for sequent calculus (left introduction rules build evaluation contexts),
- the “dark side” of sequent calculus is the call-by-value side,
- context-implicit tree-like representation of sequent calculus.

The $\mu\tilde{\mu}$-subsystem

Syntax of $\mu\tilde{\mu}^{\rightarrow\wedge}$ (aka $\overline{\lambda}\mu\tilde{\mu}$)

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \lambda x.v$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid v \cdot e$

Semantics

$$
\begin{array}{llcl}
(\mu) & \langle \mu\alpha.c \| e \rangle & \rightarrow & c[\alpha \leftarrow e] \\
(\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v] \\
(\rightarrow) & \langle \lambda x.v \| v' \cdot e \rangle & \rightarrow & \langle v' \| \tilde{\mu}x.\langle v \| e \rangle \rangle
\end{array}
$$

Equivalent syntax in $\lambda\mu$-calculus

Commands	c	$::=$	$e[v]$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \lambda x.v$
Evaluation contexts	$e[\,]$	$::=$	$\mathbf{let}\ x = [\,]\ \mathbf{in}\ c \mid [\alpha]([\,]) \mid e[[\,]\,v]$

Rules in the syntax of $\lambda\mu$-calculus

$$
\begin{array}{llcl}
(\mu) & e[\mu\alpha.c] & \rightarrow & c[[\alpha]v \leftarrow e[v]] \\
(\tilde{\mu}) & \mathbf{let}\ x = v\ \mathbf{in}\ c & \rightarrow & c[x \leftarrow v] \\
(\rightarrow) & e[(\lambda x.v)v'] & \rightarrow & \mathbf{let}\ x = v'\ \mathbf{in}\ e[v]
\end{array}
$$

Adding constructions: the example of abstraction and application

Syntax of $\mu\tilde{\mu}^{\rightarrow\wedge}$ (aka $\overline{\lambda}\mu\tilde{\mu}$)

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \lambda x.v$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid v \cdot e$

Semantics

$$
\begin{array}{lllll}
(\mu) & \langle \mu\alpha.c \| e \rangle & \rightarrow & c[\alpha \leftarrow e] \\
(\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v] \\
(\rightarrow) & \langle \lambda x.v \| v' \cdot e \rangle & \rightarrow & \langle v' \| \tilde{\mu}x.\langle v \| e \rangle \rangle
\end{array}
$$

Equivalent syntax in $\lambda\mu$-calculus

Commands	c	$::=$	$e[v]$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \lambda x.v$
Evaluation contexts	$e[\,]$	$::=$	$\mathbf{let}\ x = [\,]\ \mathbf{in}\ c \mid [\alpha]([\,]) \mid e[[\,]\,v]$

Rules in the syntax of $\lambda\mu$-calculus

$$
\begin{array}{llll}
(\mu) & e[\mu\alpha.c] & \rightarrow & c[[\alpha]v \leftarrow e[v]] \\
(\tilde{\mu}) & \mathbf{let}\ x = v\ \mathbf{in}\ c & \rightarrow & c[x \leftarrow v] \\
(\rightarrow) & e[(\lambda x.v)v'] & \rightarrow & \mathbf{let}\ x = v'\ \mathbf{in}\ e[v]
\end{array}
$$

Other examples of connectives

Syntax of $\mu\tilde{\mu}^{\to\wedge\vee\top\bot}$

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \lambda x.v \mid (v, v) \mid \iota_1(v) \mid \iota_2(v) \mid 1$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid v \cdot e \mid \pi_1 \cdot e \mid \pi_2 \cdot e \mid [e, e] \mid 0$

Semantics of $\mu\tilde{\mu}^{\to\wedge\vee\top\bot}$

$$
\begin{array}{lllll}
(\mu) & \langle \mu\alpha.c \| e \rangle & \to & c[\alpha \leftarrow e] \\
(\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \to & c[x \leftarrow v] \\
(\to) & \langle \lambda x.v \| v' \cdot e \rangle & \to & \langle v' \| \tilde{\mu}x.\langle v \| e \rangle \rangle \\
(\wedge) & \langle (v_1, v_2) \| \pi_j \cdot e \rangle & \to & \langle v_j \| e \rangle \\
(\vee) & \langle \iota_j(v) \| [e_1, e_2] \rangle & \to & \langle v \| e_j \rangle
\end{array}
$$

Typing the $\mu\tilde{\mu}$-subsystem

(a sequent calculus structure)

- two axioms
- no contraction: simulated by cuts with the axioms
- three kinds of sequents:
  - terms: distinguished formula on the right
  - ev. contexts: distinguished formula on the left
  - commands: no distinguished formula

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

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

$$\frac{\Gamma \vdash v : A \,|\, \Delta \quad \Gamma \,|\, e : A \vdash \Delta}{\langle v \| e \rangle : (\Gamma \vdash \Delta)} Cut$$

Typing extensions (e.g. implication connective)

$$\frac{\Gamma, x : A \vdash v : B \,|\, \Delta}{\Gamma \vdash \lambda x.v : A \to B \,|\, \Delta} \qquad \frac{\Gamma \vdash v : A \,|\, \Delta \qquad \Gamma \,|\, e : B \vdash \Delta}{\Gamma \,|\, v \cdot e : A \to B \vdash \Delta}$$

Typing the $\mu\tilde{\mu}$-subsystem

(sequent calculus in context-free form)

Thanks to the absence of contraction, sequent calculus proofs can be represented *à la* natural deduction

$$\dfrac{\begin{array}{c}[A \vdash] \\ \vdots \\ \vdash\end{array}}{\vdash A}\,\mu \qquad\qquad \dfrac{\begin{array}{c}[\vdash A] \\ \vdots \\ \vdash\end{array}}{A \vdash}\,\tilde{\mu}$$

$$\dfrac{\vdash A \qquad A \vdash}{\vdash}\,Cut$$

$$\dfrac{\begin{array}{c}[\vdash A] \\ \vdots \\ \vdash B\end{array}}{\vdash A \to B}\,\to_R \qquad\qquad \dfrac{\vdash A \qquad B \vdash}{A \to B \vdash}\,\to_L$$

The $\mu\tilde{\mu}$-subsystem

(the critical dilemma of computation)

Syntax

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid \ldots$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$

Semantics

$$
\begin{array}{llcl}
(\mu) & \langle \mu\alpha.c \| e \rangle & \rightarrow & c[\alpha \leftarrow e] \\
(\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v]
\end{array}
$$

The critical pair

call-by-value $\qquad \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \qquad$ call-by-name

$$
c[\alpha \leftarrow \tilde{\mu}x.c'] \;\overset{(\mu)}{\swarrow}\; \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \;\overset{(\tilde{\mu})}{\searrow}\; c'[x \leftarrow \mu\alpha.c]
$$

The $\mu\tilde{\mu}$-subsystem

(the critical dilemma of computation)

Syntax

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid \ldots$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$

Semantics

$$\begin{array}{lccc} (\mu) & \langle \mu\alpha.c \| e \rangle & \rightarrow & c[\alpha \leftarrow e] \\ (\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v] \end{array}$$

The critical pair in the syntax of $\lambda\mu$-calculus

call-by-value — $\mathbf{let}\ x = \mu\alpha.c\ \mathbf{in}\ c'$ — call-by-name

$$c[[\alpha]v \leftarrow \mathbf{let}\ x = v\ \mathbf{in}\ c'] \ \swarrow (\mu) \qquad (\tilde{\mu}) \searrow \ c'[x \leftarrow \mu\alpha.c]$$

The $\mu\tilde{\mu}$-subsystem

(the critical dilemma of computation)

Syntax

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid \ldots$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$

Semantics

$$
\begin{array}{llll}
(\mu) & \langle \mu\alpha.c \| e \rangle & \to & c[\alpha \leftarrow e] \\
(\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \to & c[x \leftarrow v]
\end{array}
$$

The critical pair

call-by-value $\qquad \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \qquad$ call-by-name

$$
c[\alpha \leftarrow \tilde{\mu}x.c'] \;\overset{(\mu)}{\swarrow} \qquad \overset{(\tilde{\mu})}{\searrow}\; c'[x \leftarrow \mu\alpha.c]
$$

The $\mu\tilde{\mu}$-subsystem

(the call-by-name confluent restriction)

Syntax

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid E$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$

Semantics

$$\begin{array}{lccl} (\mu_n) & \langle \mu\alpha.c \| E \rangle & \rightarrow & c[\alpha \leftarrow E] \\ (\tilde{\mu}) & \langle v \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v] \end{array}$$

The solved critical pair

call-by-value $\qquad \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \qquad$ call-by-name

$$\swarrow (\mu) \qquad\qquad (\tilde{\mu}) \searrow$$

$$c[\alpha \leftarrow \tilde{\mu}x.c'] \qquad\qquad\qquad c'[x \leftarrow \mu\alpha.c]$$

The $\mu\tilde{\mu}$-subsystem

(the call-by-value confluent restriction)

Syntax

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid V$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid \ldots$
Values	V	$::=$	$x \mid \ldots$

Semantics

$$\begin{array}{lll} (\mu) & \langle \mu\alpha.c \| e \rangle & \rightarrow & c[\alpha \leftarrow e] \\ (\tilde{\mu}_v) & \langle V \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow V] \end{array}$$

The solved critical pair

call-by-value $\quad \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \quad$ call-by-name

$$\swarrow (\mu) \qquad\qquad (\tilde{\mu}) \searrow$$

$$c[\alpha \leftarrow \tilde{\mu}x.c'] \qquad\qquad c'[x \leftarrow \mu\alpha.c]$$

The $\mu\tilde{\mu}$-subsystem

(two confluent symmetric restrictions)

$\mu_n\tilde{\mu}$-subsystem

Commands	c	$::=$	$\langle v \Vert e \rangle$
Terms	v	$::=$	$\mu\alpha.c \mid x \mid \ldots$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid E$

$$
\begin{array}{llll}
(\mu_n) & \langle \mu\alpha.c \Vert E \rangle & \rightarrow & c[\alpha \leftarrow E] \\
(\tilde{\mu}) & \langle v \Vert \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow v]
\end{array}
$$

$\mu\tilde{\mu}_v$-subsystem

Commands	c	$::=$	$\langle v \Vert e \rangle$
Linear terms (= values)	V	$::=$	$x \mid \ldots$
Terms	v	$::=$	$\mu\alpha.c \mid V$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid \alpha \mid \ldots$

$$
\begin{array}{llll}
(\mu) & \langle \mu\alpha.c \Vert e \rangle & \rightarrow & c[\alpha \leftarrow e] \\
(\tilde{\mu}_v) & \langle V \Vert \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow V]
\end{array}
$$

The principle of duality

Static duality

term	/	evaluation context
value	/	linear ev. context
term variable	/	ev. context variable
Parigot's μ	/	let-in context ($\tilde{\mu}$)

Semantical duality

call-by-value	/	call-by-name
$(\tilde{\mu}_v)$	/	(μ_n)
(μ)	/	$(\tilde{\mu})$

Connective-level duality

conjunction	/	disjunction
true	/	false
implication	/	subtraction

A tool to transfer results from one side to the other side...

The computational-classical-logic lineage of $\mu\tilde{\mu}^{\rightarrow}$

The programming side

Landin's J [1964]
an operator to translate goto and labels

Reynolds' `escape` [1972]
a syntactical variant of `call-cc`

Scheme's `catch/throw` [1975]
a static variant of Lisp's `catch/throw`

Felleisen *et al*'s C [1986]
abstract study of λ-calculus with control

The proof theory side

Prawitz [1965]
normalisation of natural deduction + $\neg\neg A \rightarrow A$
(no Curry-Howard)

$\searrow$ $\swarrow$

The Curry-Howard connection

Griffin [1990]
typing C of type $\neg\neg A \rightarrow A$

Parigot [1992]
a "clean" variant to $\lambda_{\mathcal{C}}$-calculus: $\lambda\mu$-calculus

Connected works

Computational symmetry

- Barbanera-Berardi's symmetric λ-calculus [1996]

- Filinski's symmetric λ-calculus syntax [1988]

Call-by-name/call-by-value duality

- Filinski's dual cont.-passing-style call-by-name and call-by-value semantics of his symmetric λ-calculus [1988]

- Selinger's dual control and co-control categories modelling call-by-name and call-by-value $\lambda\mu$-calculi [2000]

The computational-content-of-sequent-calculus lineage of $\mu\tilde{\mu}^{\rightarrow}$

- Gentzen's LK sequent calculus [1935]:
  various (implicitly weak) cut-elimination strategies of LK
- Dragalin [1979]: strong cut-elimination of LK

  Griffin's stimulus: classical logic computes in real life

- Girard's LC [1991]: associativity-and-commutativity-preserving ¬¬-translation of LK
- Danos-Joinet-Schellinx's LKT/LKQ fragments of LK [1994]:
  intuition of a connection to call-by-name and call-by-value
- Barbanera-Berardi's λ_{sym}-calculus [1996]: non-deterministic computational content (implicitly?) of sequent calculus
- Herbelin's $\overline{\lambda}$-calculus [1994]: LJT and LKT as a λ-calculus

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

$$\dfrac{\vdash v : A \qquad e : A \vdash}{\langle v \| e \rangle}\ Cut$$

$$\dfrac{\begin{array}{c}[\vdash x : A] \\ \vdots \\ \vdash v : B\end{array}}{\vdash \lambda x.v : A \rightarrow B}\ \rightarrow_R \qquad \dfrac{\vdash v : A \qquad e : B \vdash}{v \cdot e : A \rightarrow B \vdash}\ \rightarrow_L$$

- Danos-Joinet-Schellinx's analysis of the LK to LL embeddings [1995,1997]
- Urban-Bierman's extension of Barbanera-Berardi's strong normalisation method to an LK syntax [2000]
- Curien-Herbelin $\overline{\lambda}\mu\tilde{\mu}$-calculus [2000]
- Lengrand's λ_{ξ}-calculus [2003]:
  application of the duality-of-computation paradigm to Urban-Bierman's LK syntax
- Wadler's variant of $\overline{\lambda}\mu\tilde{\mu}$-calculus based on disjunctions and conjunctions [2003]

The intuitionistic constraint

evaluation context variables are bound linearly

In extensions that don't bind ev. context variables (e.g. the constructors of implication), this implies one can take a unique ev. context variable, say $\star$.

Example:

Syntax of intuitionistic $\mu\tilde{\mu}^{\rightarrow}$

Commands	c	$::=$	$\langle v \| e \rangle$
Terms	v	$::=$	$\mu \star . c \mid x \mid \lambda x . v$
Evaluation contexts	e	$::=$	$\tilde{\mu} x . c \mid \star \mid v \cdot e$

The intuitionistic constraint

evaluation context variables are bound linearly

In extensions that don't bind ev. context variables (e.g. the constructors of implication), this implies one can take a unique ev. context variable, say $\star$.

Example:

Syntax of intuitionistic $\mu\tilde{\mu}^{\rightarrow}$
(after reorganisation of the grammar)

Terms	v	$::=$	$v(e) \mid x \mid \lambda x.v$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.v \mid v \mid v \cdot e$

In the intuitionistic case, $=_v$ is a strict subset of $=_n$.

The $\mu\tilde{\mu}$-subsystem

(η-conversions)

$$\begin{array}{llcll} (\eta_\mu) & \mu\alpha.\langle V \| \alpha\rangle & = & V & \alpha \text{ not free in } V \\ (\eta_{\tilde{\mu}}) & \tilde{\mu}x.\langle x \| E\rangle & = & E & x \text{ not free in } E \end{array}$$

Each extension comes with its own η-conversions. E.g., for the constructors of implication, we have

$$(\eta_\rightarrow) \quad \lambda x.\mu\alpha.\langle y \| x \cdot \alpha\rangle \ = \ y$$

from which we derive

$$\begin{array}{llcll} (\eta_{\rightarrow n}) & \lambda x.\mu\alpha.\langle v \| x \cdot \alpha\rangle & = & v & x \text{ and } \alpha \text{ not free in } v \\ (\eta_{\rightarrow v}) & \lambda x.\mu\alpha.\langle V \| x \cdot \alpha\rangle & = & V & x \text{ and } \alpha \text{ not free in } V \end{array}$$

Focus on $\overline{\lambda}\mu_n$-calculus and $\overline{\lambda}\tilde{\mu}_v$-calculus

$\overline{\lambda}\mu_n$-calculus

$$\begin{array}{lcl} c & ::= & \langle v \| E \rangle \\ v & ::= & \mu\alpha.c \mid x \mid \lambda x.v \\ E & ::= & \alpha \mid v \cdot E \end{array}$$

Reduction

$$\begin{array}{llcl} (\mu_n) & \langle \mu\alpha.c \| E \rangle & \rightarrow & c[\alpha \leftarrow E] \\ (\rightarrow_n^\beta) & \langle \lambda x.v \| v' \cdot E \rangle & \rightarrow & \langle v[x \leftarrow v'] \| E \rangle \end{array}$$

η-reduction (with usual constraints)

$$\begin{array}{llcl} (\eta_\mu) & \mu\alpha.\langle v \| \alpha \rangle & \rightarrow & v \\ (\eta^R_{\rightarrow n}) & v & \rightarrow & \lambda x.\mu\alpha.\langle v \| x \cdot \alpha \rangle \end{array}$$

Prop: $\overline{\lambda}\mu_n$, David-Py $\lambda\mu$, and $\mu_n\tilde{\mu}^{\rightarrow}$-calculi have isomorphic equations on commands and terms.

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

$$\begin{array}{lcl} c & ::= & \langle V \| e \rangle \\ V & ::= & x \mid \lambda x.\mu\alpha.c \\ e & ::= & \alpha \mid V \cdot e \mid \tilde{\mu}x.c \end{array}$$

Reduction

$$\begin{array}{llcl} (\tilde{\mu}_v) & \langle V \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow V] \\ (\rightarrow_v^\beta) & \langle \lambda x.\mu\alpha.c \| V \cdot e \rangle & \rightarrow & c[x \leftarrow V][\alpha \leftarrow e] \end{array}$$

η-reduction (with usual constraints)

$$\begin{array}{llcl} (\eta_{\tilde{\mu}}) & \tilde{\mu}x.\langle x \| e \rangle & \rightarrow & e \\ (\eta^R_{\rightarrow v}) & \lambda x.\mu\alpha.\langle V \| x \cdot \alpha \rangle & \rightarrow & V \end{array}$$

Prop: $\overline{\lambda}\tilde{\mu}_v$ and $\mu\tilde{\mu}_v^{\rightarrow}$-calculi have isomorphic equations on commands, values and contexts.

Consequences for call-by-value λ-calculus

Call-by-value interprets the “dark side” of sequent calculus.

Conversely, sequent calculus gives to call-by-value its nobility.

Call-by-value λ-calculus (natural deduction style) is complex to study (see next page).

Can the duality foster further theoretical research on call-by-value?
Böhm theorem, standardisation, complete confluent systems, ...

A complete reduction system

cbv λ-calculus operational rule (Plotkin [1975])

(β_v)	$(\lambda x.v)\,V$	$\rightarrow$	$v[x \leftarrow V]$	

cbv λ-calculus sub-operational rule (Moggi [1988])

(let_{lift})	$F[(\lambda x.v)\,v']$	$\rightarrow$	$(\lambda x.F[v])\,v'$	

cbv λ-calculus observational rules (Moggi [1988])

(η_v)	$\lambda x.(V\,x)$	$\rightarrow$	V	x not free in V
(η_{let})	$(\lambda x.E[x])\,v$	$\rightarrow$	$E[v]$	x not free in E

cbv $\lambda\mu$-calculus extra operational rules ($_*$)

(μ_v)	$F[\mu\alpha.c]$	$\rightarrow$	$\mu\alpha.c[\alpha \leftarrow [\alpha]F]$	
(μ_{var})	$[\alpha]\mu\beta.v$	$\rightarrow$	$v[\beta \leftarrow [\alpha][\,]]$	

cbv $\lambda\mu$-calculus extra sub-operational rule ($_*$)

(μ_{let})	$(\lambda x.\mu\alpha.[\beta]v)\,v'$	$\rightarrow$	$\mu\alpha.[\beta]((\lambda x.v)\,v')$	

cbv $\lambda\mu$-calculus extra observational rule ($_*$)

(η_μ)	$\mu\alpha.[\alpha]v$	$\rightarrow$	v	α not free in V

Extra rule for confluence

(μ_v^η)	$v\,\mu\alpha.c$	$\rightarrow$	$(\lambda x.\mu\alpha.c[\alpha \leftarrow [\alpha](x\,[\,])])\,v$	

($_*$) inspired by Sabry-Felleisen [1993], Hofmann [1995] and Selinger [2000] complete axiomatics

A dual to call-by-need ?

lazy call-by-value (call-by-need)	*lazy call-by-name*

Lazy call-by-value (call-by-need)

Commands	c	$::=$	$\langle v \| e \rangle$
Linear terms (= values)	V	$::=$	$x \mid \ldots$
Terms	v	$::=$	$\mu\alpha.c \mid V$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid E$

(μ_n)	$\langle \mu\alpha.c \| E \rangle$	$\rightarrow_{lv}$	$c[\alpha \leftarrow E]$	
$(\tilde{\mu}_v)$	$\langle V \| \tilde{\mu}x.c \rangle$	$\rightarrow_{lv}$	$c[x \leftarrow V]$	
(μ_{lv})	$\langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle$	$\rightarrow_{lv}$	$c[\alpha \leftarrow \tilde{\mu}x.c']$	($_*$)
$(\tilde{\mu}_{lv})$	$\langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle$	$\rightarrow_{lv}$	c'	($_{**}$)

($_*$) if x “needed” in c'
($_{**}$) if $x \notin FV(c')$

intuitionistic restriction
observationally collapses to call-by-name

Lazy call-by-name

Commands	c	$::=$	$\langle v \| e \rangle$
Linear terms (= values)	V	$::=$	$x \mid \ldots$
Terms	v	$::=$	$\mu\alpha.c \mid V$
Linear ev. contexts	E	$::=$	$\alpha \mid \ldots$
Evaluation contexts	e	$::=$	$\tilde{\mu}x.c \mid E$

(μ_n)	$\langle \mu\alpha.c \| E \rangle$	$\rightarrow_{ln}$	$c[\alpha \leftarrow E]$	
$(\tilde{\mu}_v)$	$\langle V \| \tilde{\mu}x.c \rangle$	$\rightarrow_{ln}$	$c[x \leftarrow V]$	
(μ_{ln})	$\langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle$	$\rightarrow_{ln}$	c	($_{**}$)
$(\tilde{\mu}_{ln})$	$\langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle$	$\rightarrow_{ln}$	$c'[x \leftarrow \mu\alpha.c]$	($_*$)

($_*$) if α “needed” in c
($_{**}$) if $\alpha \notin FV(c)$

intuitionistic restriction
operationally collapses to call-by-name

with control, all four reduction systems are observationally different

The proof-as-strategy approach

strategies in game model of computations are polarised normal proofs in some sequent calculus

Lorenz-Lorenzen's game semantics of provability
strategies = cut-free proofs in LJQ/LKQ (Herbelin's PhD [1995])

different possible polarisations of $\mu\tilde{\mu}$-systems normal proofs

decompose normal proofs along commands: yields (abstract) Böhm trees (Curien-Herbelin [1998])

$\langle x \| E \rangle$ = "(" (question) with possible reactions determined by E

$\langle V \| \alpha \rangle$ = "]" (answer) with possible reactions determined by V

game interaction = head reduction in an abstract machine (Danos-Herbelin-Regnier [1996])
= head reduction in $\mu\tilde{\mu}$

intuitionistic restriction = well-bracketed parentheses
(Lorenzen's school [see Felscher 1986])

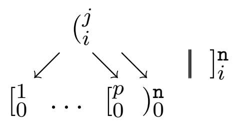
Simply-typed $\mu_n\tilde{\mu}^{\rightarrow\mathbb{N}}$-system ($\mu$PCF)

(Herbelin [1997], Laird [1997])

$N \rightarrow N$ interpreted as $?N^{\perp}\parr N$
$\mathbb{N}$ interpreted as $?\oplus_n 1$
maximal $\eta_{\rightarrow n}$-expansion
maximal η_{μ}-expansion of atoms

$$c \quad ::= \quad \langle x_i^j \| v_1 \cdot \ldots \cdot v_p \cdot [\mathtt{n} \mapsto c_{\mathtt{n}}] \rangle \mid \langle \mathtt{n} \| \alpha_i \rangle$$

where $v ::= \lambda x_1 \ldots x_n.\mu\alpha.c$

$c \quad ::=$ 

Initial state

$$\langle x \| \overbrace{v_1 \cdot \ldots \cdot v_p \cdot [\mathtt{n} \mapsto c_{\mathtt{n}}]}^{Opponent} \rangle \quad [x \leftarrow \overbrace{\;|v|\;}^{Player}]$$

Interaction rules

$$\begin{array}{llllll} (\rightarrow \mu_n) & \langle x_i^j \| \vec{v} \cdot [\mathtt{n} \mapsto c_{\mathtt{n}}] \rangle & [\sigma] & \rightarrow & c & [\vec{x} \leftarrow \vec{v}; \alpha \leftarrow [\mathtt{n} \mapsto c_{\mathtt{n}}]; \sigma'] \\ (\mathbb{N}) & \langle \mathtt{n} \| \alpha_i \rangle & [\sigma] & \rightarrow & c_{\mathtt{n}} & [\sigma'] \end{array}$$

$$\sigma(x_i^j) = (\lambda \vec{x}.\mu\alpha.c)[\sigma'] \quad \sigma(\alpha_i) = ([\mathtt{n} \mapsto c_{\mathtt{n}}])[\sigma']$$

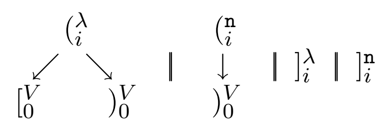
Simply-typed $\mu\tilde{\mu}_v^{\rightarrow\mathbb{N}}$-system ($\mu$PCF$_v$)

(Abramsky-McCusker [1997], Honda-Yoshida [1997], Laird [1998])

$P \rightarrow P$ interpreted as $P^{\perp}\parr !P$
$\mathbb{N}$ interpreted as $?\oplus_n 1$
maximal $\eta_{\rightarrow v}$-expansion and $\eta_{\tilde{\mu}}$-expansion
needs new constructions $\lambda\!\!\lambda\mathtt{n}.v_{\mathtt{n}}$ and $\mathtt{n} \cdot e$

$$c \quad ::= \quad \langle x_i \| V_\lambda \cdot e \rangle \mid \langle x_i \| \mathtt{n} \cdot e \rangle \mid \langle V_\lambda \| \alpha_i \rangle \mid \langle \mathtt{n} \| \alpha_i \rangle$$

where $V_\lambda ::= \lambda x.\mu\alpha.c \mid \lambda\!\!\lambda\mathtt{n}.\mu\alpha.c_{\mathtt{n}}$
$e ::= \tilde{\mu}x.c \mid [\mathtt{n} \mapsto c_{\mathtt{n}}]$

$c \quad ::=$ 

Initial states

$$\begin{array}{ll} \langle \overbrace{[\;\mathtt{n}\;]}^{Player} \| \alpha \rangle & [\alpha \leftarrow \overbrace{[\mathtt{n} \mapsto c_{\mathtt{n}}]}^{Opponent}] \\ \langle \lambda x.\mu\alpha.c \| \alpha \rangle & [\alpha \leftarrow V_\lambda \cdot e] \\ \langle \lambda\!\!\lambda\mathtt{n}.\mu\alpha.c_{\mathtt{n}} \| \alpha \rangle & [\alpha \leftarrow \mathtt{n} \cdot e] \end{array}$$

Interaction rules

$$\begin{array}{llllll} (\rightarrow \mu) & \langle x_i \| V_\lambda \cdot e \rangle & [\sigma] & \rightarrow & c & [x \leftarrow V_\lambda; \alpha \leftarrow e; \sigma'] \\ (\rightarrow^{\mathbb{N}} \mu) & \langle x_i' \| \mathtt{n} \cdot e \rangle & [\sigma] & \rightarrow & c_{\mathtt{n}} & [\alpha \leftarrow e; \sigma'] \\ (\tilde{\mu}_v) & \langle V_\lambda \| \alpha_i \rangle & [\sigma] & \rightarrow & c & [x \leftarrow V_\lambda; \sigma'] \\ (\mathbb{N}) & \langle \mathtt{n} \| \alpha_i' \rangle & [\sigma] & \rightarrow & c_{\mathtt{n}} & [\sigma'] \end{array}$$

$$\sigma(x_i) = (\lambda x.\mu\alpha.c)[\sigma'] \qquad \sigma(\alpha_i) = (\tilde{\mu}x.c)[\sigma']$$
$$\sigma(x_i') = (\lambda\!\!\lambda\mathtt{n}.\mu\alpha.c_{\mathtt{n}})[\sigma'] \qquad \sigma(\alpha_i') = ([\mathtt{n} \mapsto c_{\mathtt{n}}])[\sigma']$$

Abstract Computing Devices

(collecting the ingredients)

Hardin-Maranget-Pagano [1996]
relevance of explicit substitution to represent environments in abstract devices

Abstract machines characterised by reducing at the root of the computation

The $\mu\tilde{\mu}$-system:

- has a primitive notion of evaluation contexts (“stacks”)
- has a primitive notion of “states” (the commands)
- redex of non head normal states are at the root

Add an (ad hoc) variable numbering schemes

All ingredients are here to simulate abstract machines

Duality and dependent types

(when left-right symmetry meets left-right dependency)

	call-by-name	call-by-value
implicit dependent product (intersection)	OK	OK (but restricted to lin. ev. ctx.)
implicit dependent sum (union)	OK (but restricted to values - cf Pierce [1991])	OK
explicit dependent sum (stand-alone domain)	OK	OK
explicit dependent sum (stand-alone domain)	OK	OK
explicit dependent product (type-theoretic)	OK (but bypass $\tilde{\mu}$)	OK (but applied to value only)
explicit dependent sum (type-theoretic)	classical case degenerated (Herbelin [2005])	

Conclusions and problems

The $\mu\tilde{\mu}$-subsystem is an elegant tool to investigate the duality properties of the computation, and to revisit the foundations of λ-calculus.

The $\overline{\lambda}\tilde{\mu}_v$-calculus, good candidate for studying classical call-by-value computation.

Can we give a symmetric (non substitution-based) semantics (see next page) for system $\mu\tilde{\mu}$ (see e.g. Došen-Petrić categories of distributive lattices – see also Führmann-Pym and Lamarche-Straßburger “boolean” categories)?

How to interpret Gentzen’s cross-cuts (see two pages further)? Which connection with Coquand-Herbelin symmetric product, Raghunandan-Summers’s symmetric cut-elimination of a primitive “iff” connective?

The duality finds limits in dependent type theory.

Manuscript (in French) available at
`http://pauillac.inria.fr/˜herbelin/habilitation/memoire.ps`

Focus on symmetric reduction

Add command $\{c, c\}$ and take rules

$$
\begin{array}{lrcl}
(\cup) & \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle & \rightarrow & \{c[\alpha \leftarrow \tilde{\mu}x.c'], c'[x \leftarrow \mu\alpha.c]\} \\
(\mu_n) & \langle \mu\alpha.c \| E \rangle & \rightarrow & c[\alpha \leftarrow E] \\
(\tilde{\mu}_v) & \langle V \| \tilde{\mu}x.c \rangle & \rightarrow & c[x \leftarrow V]
\end{array}
$$

Which valid equations about $\{c, c'\}$ (associativity, commutativity, distributivity)?

Focus on Gentzen's cross-cuts, compared to CBN and CBV reductions

$$\dfrac{\dfrac{\Gamma \vdash A, \overline{A}, \Delta \qquad \Gamma, \overline{A}, \overline{A} \vdash \Delta}{\Gamma \vdash A, \Delta} \qquad \dfrac{\Gamma \vdash \underline{A}, \underline{A}, \Delta \qquad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma, A \vdash \Delta}}{\Gamma \vdash \Delta}$$

$\uparrow$ (Cross-cuts)

$$\dfrac{\Gamma \vdash A, A, \Delta \qquad \Gamma, A, A \vdash \Delta}{\Gamma \vdash \Delta}$$

(CBN) $\swarrow$ $\qquad\qquad$ $\searrow$ (CBV)

$$\dfrac{\dfrac{\Gamma \vdash A, \overline{A}, \Delta \qquad \dfrac{\Gamma \vdash \underline{A}, \underline{A}, \Delta \quad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma, \overline{A} \vdash \Delta}}{\Gamma \vdash A, \Delta} \qquad \dfrac{\Gamma \vdash \underline{A}, \underline{A}, \Delta \qquad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma, A \vdash \Delta}}{\Gamma \vdash \Delta}$$

$$\dfrac{\dfrac{\Gamma \vdash A, \overline{A}, \Delta \qquad \Gamma, \overline{A}, \overline{A} \vdash \Delta}{\Gamma \vdash A, \Delta} \qquad \dfrac{\dfrac{\Gamma \vdash A, \overline{A}, \Delta \quad \Gamma, \overline{A}, \overline{A} \vdash \Delta}{\Gamma \vdash \underline{A}, \Delta} \qquad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma, A \vdash \Delta}}{\Gamma \vdash \Delta}$$

Extra material

Coinductive and inductive types

(Paulin-Mohring's fixpoint-based decomposition)

Coinductive type
= term constructors
+ ev. ctx. constructors
+ term recursion guarded by a term constructor

Inductive type
= term constructors
+ ev. ctx. constructors
+ ev. ctx. recursion guarded by an ev. ctx. constructor

Examples

A coinductive type (possibly infinite lists)

$$\begin{array}{lcl} V & ::= & \ldots \mid \texttt{nil} \mid \texttt{cons}(v,v) \mid \nu_x.V_c \\ E & ::= & \ldots \mid [\texttt{nil}.c, \texttt{cons}(x_a,x_l).c] \end{array}$$

An inductive type (lists)

$$\begin{array}{lcl} V & ::= & \ldots \mid \texttt{nil} \mid \texttt{cons}(v,v) \\ E & ::= & \ldots \mid [\texttt{nil}.c, \texttt{cons}(x_a,x_l).c] \mid \tilde{\nu}_\alpha.E_c \end{array}$$

$$\begin{array}{lll} (\texttt{nil}) & \langle \texttt{nil} \| [\texttt{nil}.c, \texttt{cons}(x_a,x_l).c'] \rangle & \rightarrow\ c \\ (\texttt{cons}) & \langle \texttt{cons}(v_a,v_l) \| [\texttt{nil}.c, \texttt{cons}(x_a,x_l).c'] \rangle & \rightarrow\ \langle v_a \| \tilde{\mu} x_a. \langle v_l \| \tilde{\mu} x_l.c' \rangle \rangle \end{array}$$

$$(\nu)\quad \langle \nu_x.V_c \| E_c \rangle \ \rightarrow\ \langle V_c[x \leftarrow \nu_x.V_c] \| E_c \rangle$$

$$(\tilde{\nu})\quad \langle V_c \| \tilde{\nu}_\alpha.E_c \rangle \ \rightarrow\ \langle V_c \| E_c[\alpha \leftarrow \tilde{\nu}_\alpha.E_c] \rangle$$

V_c means constructed V
E_c means constructed E