

Investigations into the Duality of Computation

Hugo Herbelin
(INRIA-Futurs)

Marseille, 23 novembre 2007

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

Context

proof theory

theory of computation

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

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

Which meaning to assign to cut-elimination in classical logic?

Non-determinism (as in Dragalin)?

Breaking the left-right symmetry? On which criterion?

Non substitution-based cut-elimination: cross-cuts and assimilated?

Semantics of classical logic: the linear logic's school

Girard's LC [1991]: constraining the rules of classical logic so that basic algebraic properties hold: negation a duality and not a connective, associativity and distributivity for \wedge and \vee , ...

Danos-Joinet-Schellinx's LKT, LKQ [1993]: two dual asymmetric deterministic semantics of LK:

$$\begin{array}{c} \text{one-sided} \\ \overbrace{\text{LC}(\wedge/\vee)} \\ \approx \\ \overbrace{\text{LKT}(\wedge_a, \vee_m, \rightarrow, \leftarrow)} \\ \approx \\ \overbrace{\text{LKQ}(\wedge_m, \vee_a, \backslash, /)} \end{array}$$

Especially: $\text{LKT}(\rightarrow)$ is a subset of $\text{LC}(\wedge/\vee)$ but $\text{LKQ}(\rightarrow)$ is not: $(A \wedge_m B) \rightarrow C \not\approx A \rightarrow B \rightarrow C$ in $\text{LKQ}(\rightarrow, \wedge_m)$.

Danos-Joinet-Schellinx's LK_{tq} [1994]: investigation of all semantics of LK that are definable in LL:

$$\begin{array}{c} \text{LK}_{tq}(\rightarrow) \approx \text{LK}_t(\wedge_m, \vee_m, \bar{\wedge}_m, \bar{\vee}_m, \rightarrow, \leftarrow, \backslash, /) \approx \text{LK}_q(\wedge_m, \vee_m, \bar{\wedge}_m, \bar{\vee}_m, \rightarrow, \leftarrow, \backslash, /) \\ \cup \qquad \qquad \qquad \cup \qquad \qquad \qquad \cup \\ \text{LK}_{tq}^n(\rightarrow) \approx \text{LK}_T(\wedge_m, \vee_m, \bar{\wedge}_m, \bar{\vee}_m, \rightarrow, \leftarrow, \backslash, /) \approx \text{LK}_Q(\wedge_m, \vee_m, \bar{\wedge}_m, \bar{\vee}_m, \rightarrow, \leftarrow, \backslash, /) \end{array}$$

Laurent's LLP [1999]:

$$\overbrace{\text{LLP}(!/?, \oplus/\&, \otimes/\wp)}^{\text{one-sided}} \approx \overbrace{\text{LJ}(\neg, \vee_a, \wedge_m)}^{\text{two-sided}}$$

Semantics of classical logic: the Curry-Howard-style school

Herbelin's $\bar{\lambda}$ -calculus for denoting LJ_{\top} and LK_{\top} proofs [1994].

Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}$ -calculus for denoting proofs (of slight variants) of LK_{\top} and LK_{Q} [2000]

- explicitly connect LK_{\top} to call-by-name and LK_{Q} to call-by-value
- exhibit a deep duality between terms and evaluation contexts
- LK_{\top} and LK_{Q} are subcalculi of the more general $LK_{\mu\tilde{\mu}}$

Sequent calculus $LK_{\mu\tilde{\mu}}(\rightarrow)$

- two axioms
- no contraction: simulated by cuts with the axioms
- three kinds of sequents $\left\{ \begin{array}{l} \text{distinguished formula on the right} \\ \text{distinguished formula on the left} \\ \text{no distinguished formula} \end{array} \right.$

$$\frac{}{\Gamma, A \vdash A; \Delta} Ax_R \quad \frac{}{\Gamma; A \vdash A, \Delta} Ax_L$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A; \Delta} Act_R \quad \frac{\Gamma, x : A \vdash \Delta}{\Gamma; A \vdash \Delta} Act_L$$

$$\frac{\Gamma \vdash A; \Delta \quad \Gamma; A \vdash \Delta}{\Gamma \vdash \Delta} Cut$$

$$\frac{\Gamma, A \vdash B; \Delta}{\Gamma \vdash A \rightarrow B; \Delta} \rightarrow_R \quad \frac{\Gamma \vdash A; \Delta \quad \Gamma; B \vdash \Delta}{\Gamma; A \rightarrow B \vdash \Delta} \rightarrow_L$$

we loose the slogan “normal = cut-free” but ...

LK _{$\mu\tilde{\mu}$} (\rightarrow) in context-free form

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

$$\frac{[A \vdash] \quad \vdots \quad \vdash}{\vdash A} \mu \qquad \frac{[\vdash A] \quad \vdots \quad \vdash}{A \vdash} \tilde{\mu}$$

$$\frac{\vdash A \quad A \vdash}{\vdash} \textit{Cut}$$

$$\frac{[\vdash A] \quad \vdots \quad \vdash B}{\vdash A \rightarrow B} \rightarrow_R \qquad \frac{\vdash A \quad B \vdash}{A \rightarrow B \vdash} \rightarrow_L$$

LK_{μ̃}(→) as a typed λ-calculus

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

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

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

$$\frac{\Gamma, x : A \vdash v : B; \Delta}{\Gamma \vdash \lambda x.v : A \rightarrow B; \Delta} \rightarrow_R \quad \frac{\Gamma \vdash v : A; \Delta \quad \Gamma; e : B \vdash \Delta}{\Gamma; v \cdot e : A \rightarrow B \vdash \Delta} \rightarrow_L$$

Underlying language ($\bar{\lambda}\mu\tilde{\mu}$)

commands

$$c ::= \langle v \| e \rangle$$

terms

$$v ::= x \mid \mu\alpha.c \mid \lambda x.v$$

evaluation contexts

$$e ::= \alpha \mid \tilde{\mu}x.c \mid v \cdot e$$

Let's now focus on the pure (untyped) computational content of $LK_{\mu\tilde{\mu}} \dots$

Outline

An analysis of the underlying term structure of sequent calculus

- System $\mu\tilde{\mu}$: the core of computation
- Adding connectives
- 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 and 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

Delimited control

- The completeness properties of delimited control
- $\lambda\mu\hat{\mu}\tau\rho$: a fine-grain analysis of Danvy-Filinski *shift/reset* calculus
- Call-by-name delimited continuations and Saurin's complete extension of de Groote's variant of $\lambda\mu$ -calculus
- System $\bar{\lambda}\mu\tilde{\mu}\hat{\mu}\tau\rho$: delimited continuations on top of system $\bar{\lambda}\mu\tilde{\mu}$

Summary

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
- 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.

<i>Syntax</i>	
Commands	$c ::= \langle v \ e \rangle$
Terms	$v ::= \mu\alpha.c \mid x$
Evaluation contexts	$e ::= \tilde{\mu}x.c \mid \alpha$
<i>Semantics</i>	
(μ)	$\langle \mu\alpha.c \ e \rangle \rightarrow c[\alpha \leftarrow e]$
$(\tilde{\mu})$	$\langle v \ \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow v]$

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

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

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

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

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

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

Adding constructions: the example of abstraction and application

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

Commands $c ::= \langle v \parallel e \rangle$
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

(μ) $\langle \mu\alpha.c \parallel e \rangle \rightarrow c[\alpha \leftarrow e]$
 $(\tilde{\mu})$ $\langle v \parallel \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow v]$
 (\rightarrow) $\langle \lambda x.v \parallel v' \cdot e \rangle \rightarrow \langle v' \parallel \tilde{\mu}x.\langle v \parallel e \rangle \rangle$

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

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

Other examples of connectives

Syntax of $\mu\tilde{\mu}^{\rightarrow\wedge_a\vee_a\top\perp}$

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 $\mu\tilde{\mu}^{\rightarrow\wedge_a\vee_a\top\perp}$

(μ)	$\langle \mu\alpha.c \ e \rangle$	\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	$\langle v' \ \tilde{\mu}x.\langle v \ e \rangle \rangle$
(\wedge_a)	$\langle (v_1, v_2) \ \pi_j \cdot e \rangle$	\rightarrow	$\langle v_j \ e \rangle$
(\vee_a)	$\langle \iota_j(v) \ [e_1, e_2] \rangle$	\rightarrow	$\langle v \ e_j \rangle$

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

(the critical dilemma of computation)

Syntax

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

Semantics

(μ)	$\langle \mu\alpha.c \ e \rangle \rightarrow c[\alpha \leftarrow e]$
$(\tilde{\mu})$	$\langle v \ \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow v]$

The critical pair

call-by-value	$\langle \mu\alpha.c \ \tilde{\mu}x.c' \rangle$	call-by-name
$\swarrow (\mu)$		$\searrow (\tilde{\mu})$
$c[\alpha \leftarrow \tilde{\mu}x.c']$		$c'[x \leftarrow \mu\alpha.c]$

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

(the critical dilemma of computation)

Syntax

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

Semantics

(μ)	$\langle \mu\alpha.c \ e \rangle$	\rightarrow	$c[\alpha \leftarrow e]$
$(\tilde{\mu})$	$\langle v \ \tilde{\mu}x.c \rangle$	\rightarrow	$c[x \leftarrow v]$

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

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

(the critical dilemma of computation)

Syntax

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

Semantics

(μ) $\langle \mu\alpha.c \| e \rangle \rightarrow c[\alpha \leftarrow e]$
 $(\tilde{\mu})$ $\langle v \| \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow v]$

The critical pair

call-by-value $\langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle$ call-by-name
 $\swarrow (\mu)$ $(\tilde{\mu}) \searrow$
 $c[\alpha \leftarrow \tilde{\mu}x.c']$ $c'[x \leftarrow \mu\alpha.c]$

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

(the call-by-name confluent restriction)

Syntax

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

Semantics

(μ_n)	$\langle \mu\alpha.c \ E \rangle \rightarrow c[\alpha \leftarrow E]$
$(\tilde{\mu})$	$\langle v \ \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow v]$

The solved critical pair

call-by-value	$\langle \mu\alpha.c \ \tilde{\mu}x.c' \rangle$	call-by-name
$\swarrow (\mu)$		$\searrow (\tilde{\mu})$
$c[\alpha \leftarrow \tilde{\mu}x.c']$		$c'[x \leftarrow \mu\alpha.c]$

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	$e ::= \tilde{\mu}x.c \mid \alpha \mid \dots$
Values	$V ::= x \mid \dots$

Semantics

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

The solved critical pair

$$\begin{array}{ccc}
 \text{call-by-value} & \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle & \text{call-by-name} \\
 \swarrow (\mu) & & \searrow (\tilde{\mu}) \\
 c[\alpha \leftarrow \tilde{\mu}x.c'] & & c'[x \leftarrow \mu\alpha.c]
 \end{array}$$

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

(two confluent symmetric restrictions)

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

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

(μ_n)	$\langle \mu\alpha.c \ E \rangle$	\rightarrow	$c[\alpha \leftarrow E]$
$(\tilde{\mu})$	$\langle v \ \tilde{\mu}x.c \rangle$	\rightarrow	$c[x \leftarrow v]$

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

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

(μ)	$\langle \mu\alpha.c \ e \rangle$	\rightarrow	$c[\alpha \leftarrow e]$
$(\tilde{\mu}_v)$	$\langle V \ \tilde{\mu}x.c \rangle$	\rightarrow	$c[x \leftarrow V]$

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

(two confluent symmetric restrictions - the typed point of view - informally)

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

Commands	$!?\Gamma \vdash ?!\Delta$
Terms	$!?\Gamma \vdash ?!\Delta; ?!A$
Linear ev. contexts	$!?\Gamma; A \vdash ?!\Delta;$
Evaluation contexts	$!?\Gamma; !?A \vdash ?!\Delta;$

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

Commands	$!\Gamma \vdash !?\Delta$
Linear terms (= values)	$!\Gamma \vdash !?\Delta; A$
Terms	$!\Gamma \vdash !?\Delta; !?A$
Evaluation contexts	$!\Gamma; !?A \vdash !?\Delta$

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)



The Curry-Howard connection



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

Parigot [1992]
a "clean" variant to λ_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 $\neg\neg$ -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 $\bar{\lambda}$ -calculus [1994]: LJT and LKT as a λ -calculus
- 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 $\bar{\lambda}\mu\tilde{\mu}$ -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 $\bar{\lambda}\mu\tilde{\mu}$ -calculus based on disjunctions and conjunctions [2003]

$$\begin{array}{c}
 \frac{[\alpha : A \vdash] \quad \vdots \quad c}{\vdash \mu\alpha.c : A} \mu \qquad \frac{[\vdash x : A] \quad \vdots \quad c}{\tilde{\mu}x.c : A \vdash} \tilde{\mu} \\
 \\
 \frac{\vdash v : A \quad e : A \vdash}{\langle v \| e \rangle} \text{Cut} \\
 \\
 \frac{[\vdash x : A] \quad \vdots \quad \vdash v : B}{\vdash \lambda x.v : A \rightarrow B} \rightarrow_R \qquad \frac{\vdash v : A \quad e : B \vdash}{v \cdot e : A \rightarrow B \vdash} \rightarrow_L
 \end{array}$$

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

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

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

(η -conversions)

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

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

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

Some η -conversions are consistent only with some subsystems of $\mu\tilde{\mu}$ (expressed at the type level: semantics of LK_t is not stable by η_{\rightarrow} but semantics of LK_T is)

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

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

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

Reduction

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

η -reduction (with usual constraints)

$$\begin{aligned} (\eta_\mu) \quad \mu\alpha.\langle v \| \alpha \rangle &\rightarrow v \\ (\eta_{\rightarrow n}^R) \quad v &\rightarrow \lambda x.\mu\alpha.\langle v \| x \cdot \alpha \rangle \end{aligned}$$

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

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

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

Reduction

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

η -reduction (with usual constraints)

$$\begin{aligned} (\eta_{\tilde{\mu}}) \quad \tilde{\mu}x.\langle x \| e \rangle &\rightarrow e \\ (\eta_{\rightarrow v}^R) \quad \lambda x.\mu\alpha.\langle V \| x \cdot \alpha \rangle &\rightarrow V \end{aligned}$$

Prop: $\bar{\lambda}\tilde{\mu}_v$ 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.

Can the duality foster further theoretical research on call-by-value?

Böhm theorem, standardisation, complete confluent systems, ...

The complex call-by-value theory of $\lambda\mu$ -calculus

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

$$(\beta_v) \quad (\lambda x.v) V \quad \rightarrow \quad v[x \leftarrow V]$$

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

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

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

$$\begin{array}{lll} (\eta_v) & \lambda x.(V x) & \rightarrow V \quad x \text{ not free in } V \\ (\eta_{let}) & (\lambda x.E[x]) v & \rightarrow E[v] \quad x \text{ not free in } E \end{array}$$

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

$$\begin{array}{lll} (\mu_v) & F[\mu\alpha.c] & \rightarrow \mu\alpha.c[\alpha \leftarrow [\alpha]F] \\ (\mu_{var}) & [\alpha]\mu\beta.v & \rightarrow v[\beta \leftarrow [\alpha][\]] \end{array}$$

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

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

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

$$(\eta_\mu) \quad \mu\alpha.[\alpha]v \quad \rightarrow \quad v \quad \alpha \text{ not free in } V$$

Extra rule for confluence

$$(\mu_v^\eta) \quad v \mu\alpha.c \quad \rightarrow \quad (\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*)

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

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)

$$V ::= \dots \mid \mathbf{nil} \mid \mathbf{cons}(v, v) \mid \nu_x.V_c$$

$$E ::= \dots \mid [\mathbf{nil}.c, \mathbf{cons}(x_a, x_l).c]$$

(**nil**) $\langle \mathbf{nil} \parallel [\mathbf{nil}.c, \mathbf{cons}(x_a, x_l).c] \rangle$

(**cons**) $\langle \mathbf{cons}(v_a, v_l) \parallel [\mathbf{nil}.c, \mathbf{cons}(x_a, x_l).c] \rangle$

(ν) $\langle \nu_x.V_c \parallel E_c \rangle \rightarrow \langle V_c[x \leftarrow \nu_x.V_c] \parallel E_c \rangle$

An inductive type (lists)

$$V ::= \dots \mid \mathbf{nil} \mid \mathbf{cons}(v, v)$$

$$E ::= \dots \mid [\mathbf{nil}.c, \mathbf{cons}(x_a, x_l).c] \mid \tilde{\nu}_\alpha.E_c$$

$\rightarrow c$

$\rightarrow \langle v_a \parallel \tilde{\mu}x_a. \langle v_l \parallel \tilde{\mu}x_l.c' \rangle \rangle$

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

V_c means constructed V

E_c means constructed E

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])

$\langle x \parallel E \rangle = \text{"(" (question) with possible reactions determined by } E$

$\langle V \parallel \alpha \rangle = \text{"}" (answer) with possible reactions determined by } V$

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 (μPCF)
(Herbelin [1997], Laird [1997])

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

$$c ::= \langle x_i^j \| v_1 \cdot \dots \cdot v_p \cdot [\mathbf{n} \mapsto c_n] \rangle \mid \langle \mathbf{n} \| \alpha_i \rangle$$

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

$$c ::= \begin{array}{c} \begin{array}{ccc} & \begin{array}{c} (j) \\ \swarrow \quad \searrow \end{array} & \\ \begin{array}{c} [0] \\ \downarrow \end{array} & \dots & \begin{array}{c} [0] \\ \downarrow \end{array} \end{array} \begin{array}{c} \mathbf{n} \\ \downarrow \end{array} \end{array} \mid \begin{array}{c} \mathbf{n} \\ \downarrow \end{array}$$

Initial state

$$\langle x \mid \overbrace{v_1 \cdot \dots \cdot v_p \cdot [\mathbf{n} \mapsto c_n]}^{\text{Opponent}} \rangle \quad [x \leftarrow \overbrace{\psi}^{\text{Player}}]$$

Interaction rules

$$\begin{array}{l} (\rightarrow \mu_n) \quad \langle x_i^j \| \vec{v} \cdot [\mathbf{n} \mapsto c_n] \rangle \quad [\sigma] \rightarrow c \quad [\vec{x} \leftarrow \vec{v}; \alpha \leftarrow [\mathbf{n} \mapsto c_n]; \sigma'] \\ (\mathbb{N}) \quad \langle \mathbf{n} \| \alpha_i \rangle \quad [\sigma] \rightarrow c_n \quad [\sigma'] \\ \sigma(x_i^j) = (\lambda \vec{x}. \mu \alpha. c)[\sigma'] \quad \sigma(\alpha_i) = ([\mathbf{n} \mapsto c_n])[\sigma'] \end{array}$$

Simply-typed $\mu \tilde{\mu}_v^{\rightarrow \mathbb{N}}$ -system (μPCF_v)
(Abramsky-McCusker [1997], Honda-Yoshida [1997], Laird [1998])

$P \rightarrow P$ interpreted as $P^\perp \wp! P$
 \mathbb{N} interpreted as $? \oplus_n 1$
maximal $\eta_{\rightarrow v}$ -expansion and $\eta_{\tilde{\mu}}$ -expansion
needs new constructions $\llbracket \mathbf{n}. v_n$ and $\mathbf{n} \cdot e$

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

where $V_\lambda ::= \lambda x. \mu \alpha. c \mid \llbracket \mathbf{n}. \mu \alpha. c_n$
 $e ::= \tilde{\mu} x. c \mid [\mathbf{n} \mapsto c_n]$

$$c ::= \begin{array}{c} \begin{array}{ccc} & \begin{array}{c} (\lambda) \\ \swarrow \quad \searrow \end{array} & \\ \begin{array}{c} [0] \\ \downarrow \end{array} & \dots & \begin{array}{c} [0] \\ \downarrow \end{array} \end{array} \begin{array}{c} \mathbf{n} \\ \downarrow \end{array} \end{array} \mid \begin{array}{c} \mathbf{n} \\ \downarrow \end{array} \mid \begin{array}{c} \lambda \\ \downarrow \end{array} \mid \begin{array}{c} \mathbf{n} \\ \downarrow \end{array}$$

Initial states

$$\begin{array}{cc} \begin{array}{c} \text{Player} \\ \langle [\mathbf{n}] \| \alpha \rangle \\ \langle \lambda x. \mu \alpha. c \| \alpha \rangle \\ \langle \llbracket \mathbf{n}. \mu \alpha. c_n \| \alpha \rangle \end{array} & \begin{array}{c} \text{Opponent} \\ \langle \alpha \leftarrow [\mathbf{n} \mapsto c_n] \rangle \\ \langle \alpha \leftarrow V_\lambda \cdot e \rangle \\ \langle \alpha \leftarrow \mathbf{n} \cdot e \rangle \end{array} \end{array}$$

Interaction rules

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

$$\begin{array}{ll} \sigma(x_i) = (\lambda x. \mu \alpha. c)[\sigma'] & \sigma(\alpha_i) = (\tilde{\mu} x. c)[\sigma'] \\ \sigma(x_i') = (\llbracket \mathbf{n}. \mu \alpha. c_n)[\sigma'] & \sigma(\alpha_i') = ([\mathbf{n} \mapsto c_n])[\sigma'] \end{array}$$

Ludics syntax (a slight generalization)

Cut-free syntax

Commands $c ::= \langle x \parallel n \ p \ t_1 \dots t_n \rangle \mid \Omega \mid \dagger$
Terms $t ::= \lambda n. \lambda p. \lambda x_1 \dots x_n. c$
Substitutions $[\sigma]$

Ludics syntax (a slight generalization)

Syntax with cuts

Commands	c	$::=$	$\langle x \parallel n \ p \ t_1 \dots t_n \rangle \mid \Omega \mid \dagger \mid c[\sigma]$
Terms	t	$::=$	$\lambda n. \lambda p. \lambda x_1 \dots x_n. c \mid t[\sigma]$
Substitutions	$[\sigma]$	$::=$	$[t_1/x_1 \dots t_n/x_n]$

Semantics

$$\begin{aligned}
 \langle x \parallel n \ p \ t_1 \dots t_n \rangle [\sigma] &\rightarrow c_p^n [t_1/x_1 \dots t_n/x_n, \sigma] && \text{if } [\sigma] \text{ is } [\dots \lambda n. \lambda p. \lambda x_1 \dots x_n. c_p^n / x \dots] \text{ and } c_p^n \text{ not } \Omega \\
 \langle x \parallel n \ p \ t_1 \dots t_n \rangle [\sigma] &\rightarrow \langle x \parallel n \ p \ t_1[\sigma] \dots t_n[\sigma] \rangle && \text{if } x \text{ not bound in } \sigma \\
 (\lambda n. \lambda p. \lambda x_1 \dots x_n. c_p^n) [\sigma] &\rightarrow \lambda n. \lambda p. \lambda x_1 \dots x_n. (c_p^n [\sigma])
 \end{aligned}$$

“Dessein” = x 's occur linearly and p branching is degenerated when $n = 0$ (since there is only one empty subset while there are infinitely many finite subset of non-zero cardinal – p is the index of a subset in an enumeration of finite subsets of cardinal n)

Remark: To avoid variable captures, names are made of sequences of integers. In $\langle x \parallel n \ p \ t_1 \dots t_n \rangle$, if t_i is $\lambda n. \lambda p. \lambda x_1 \dots x_n. c_p^n$ then x_j is the concatenation of j to the name x .

Remark: The generalization allows for instance to represent $true := \langle x \parallel 0 \ 0 \rangle : 1 \oplus 1$ and $false := \langle x \parallel 0 \ 1 \rangle : 1 \oplus 1$ (where x is the name of the formula $1 \oplus 1$).

Ludics as a sequent calculus (a slight generalization)

Types

$$N ::= \&_{I \in \mathcal{I}} \wp_{i \in I} N_{I_i}^\perp \quad \text{where } \mathcal{I} \subset \mathcal{P}_{fin}(\mathbb{N})$$

Typing

$$\frac{\Delta \vdash t_1 : N_{I_0 i_1} \quad \dots \quad \Delta \vdash t_n : N_{I_0 i_n}}{\langle x \parallel n (\#_n I_0) t_1 \dots t_n \rangle : (\Delta, x : \&_{I \in \mathcal{I}} \wp_{i \in I} N_{I_i}^\perp \vdash)} \quad I_0 = \{i_1 \dots i_n\} \quad \frac{\forall I = \{i_1 \dots i_n\} \in \mathcal{I} \quad c_{\#_n I}^n : (\Delta, x_1 : N_{I_{i_1}}, \dots, x_n : N_{I_{i_n}} \vdash)}{\Delta \vdash \lambda n. \lambda p. \lambda x_1 \dots x_n. c_p^n : \&_{I \in \mathcal{I}} \wp_{i \in I} N_{I_i}^\perp} \quad \mathcal{I}}{\text{where } c_{\#_n I}^n \text{ is } \Omega \text{ if } I \notin \mathcal{I}}$$

$$\frac{}{\dagger : (\Delta \vdash)} \quad \frac{\dots \Delta \vdash t_i : N_i \quad \dots \quad c : (\Delta, x_1 : N_1, \dots, x_n : N_n \vdash)}{c[t_1/x_1 \dots t_n/x_n] : (\Delta \vdash)}$$

$$\frac{\dots \Delta \vdash t_i : N_i \quad \dots \quad \Delta, x_1 : N_1, \dots, x_n : N_n \vdash t : N}{\Delta \vdash t[t_1/x_1 \dots t_n/x_n] : N}$$

($\#_n I$ is the index of $I \in \mathbb{N}^n$ in an enumeration of \mathbb{N}^n – with infinitely many copies of \emptyset)

Ludics as a $\mu\tilde{\mu}^{(\oplus\otimes)}(\&\wp)$ -calculus

Syntax

Commands	$c ::= \langle t \parallel e \rangle \mid \dagger$
Terms	$t ::= x \mid \lambda n. \lambda p. \lambda x_1 \dots x_n. c$
Evaluation contexts	$e ::= n \ p \ t_1 \dots t_n \mid \tilde{\mu} x. c$

Semantics

$\langle \lambda n. \lambda p. \lambda x_1 \dots x_n. c_p^n \parallel n \ p \ t_1 \dots t_n \rangle$	$\rightarrow c_p^n[t_1/x_1 \dots t_n/x_n]$ if c_p^n is defined
$\langle t \parallel \tilde{\mu} x. c \rangle$	$\rightarrow c[t/x]$

Typing

$\frac{\Delta \vdash t_1 : N_{I_0 i_1} \quad \dots \quad \Delta \vdash t_n : N_{I_0 i_n}}{\Delta; n \ (\#_n I_0) \ t_1 \dots t_n : \&_{I \in \mathcal{I}} \wp_{i \in I} N_{I_i}^\perp \vdash}$	$\frac{\forall I = \{i_1, \dots, i_n\} \in \mathcal{I} \quad c_{\#_n I}^n : (\Delta, x_1 : N_{I_{i_1}}, \dots, x_n : N_{I_{i_n}} \vdash)}{\Delta \vdash \lambda n. \lambda p. \lambda x_1 \dots x_n. c_p^n : \&_{I \in \mathcal{I}} \wp_{i \in I} N_{I_i}^\perp \vdash}$ <p style="text-align: center; margin-top: -10px;"><i>where $c_{\#_n I}^n$ is Ω if $I \notin \mathcal{I}$</i></p>
$\frac{c : (\Delta, x : N \vdash)}{\Delta; \tilde{\mu} x. c : N \vdash}$	$\frac{}{\dagger : (\Delta \vdash)}$
	$\frac{\Delta \vdash t : N \quad \Delta; e : N \vdash}{\langle t \parallel e \rangle : (\Delta \vdash)}$

($\#_n I$ is the index of $I \in \mathbb{N}^n$ in an enumeration of \mathbb{N}^n)

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])	

Analysis from the point of view of linear logic

Naive approach: interpret sequents as $! \Gamma \vdash ? \Delta$

$$\frac{}{! \Gamma, ! A \vdash ? A, ? \Delta} Ax \qquad \frac{\frac{blabla}{blabla}}{! \Gamma, ! A \vdash ? A, ? \Delta} Ax$$

$$\frac{! \Gamma \vdash ? A, ? \Delta}{! \Gamma \vdash ? A, ? \Delta} \mu \qquad \frac{! \Gamma, ! A \vdash ? \Delta}{! \Gamma, ! A \vdash ? \Delta} \tilde{\mu}$$

$$\frac{! \Gamma \vdash ? A, ? \Delta \quad ! \Gamma, ! A \vdash ? \Delta}{! \Gamma \vdash ? \Delta} Cut$$

The ?-? critical pair

Analysis from the point of view of linear logic

Call-by-name calculus with η -constraint: interpret sequents as $!?\Gamma \vdash ?\Delta$

$$\begin{array}{c}
 \frac{\frac{}{!?\Gamma, ?A \vdash ?A, ?\Delta} Ax}{!?\Gamma, !?A \vdash ?A, ?\Delta} !_L \quad \frac{\frac{}{!?\Gamma, ?A \vdash ?A, ?\Delta} Ax}{!?\Gamma, !?A \vdash ?A, ?\Delta} !_L \\
 \\
 \frac{!?\Gamma \vdash ?A, ?\Delta}{!?\Gamma \vdash ?A, ?\Delta} \mu \quad \frac{!?\Gamma, !?A \vdash ?\Delta}{!?\Gamma, !?A \vdash ?\Delta} \tilde{\mu} \\
 \\
 \frac{\frac{!?\Gamma \vdash ?A, ?\Delta}{!?\Gamma \vdash !?A, ?\Delta} !_R \quad !?\Gamma, !?A \vdash ?\Delta}{!?\Gamma \vdash ?\Delta} Cut
 \end{array}$$

This corresponds to marking all formulas t in LK_{tq}^η (see Danos-Joinet-Schellinx "A new deconstructive logic: LL")

$$\frac{\frac{\frac{!?\Gamma, !?A \vdash ?B, ?\Delta}{!?\Gamma, !?A \vdash !?B, ?\Delta} !_R}{!?\Gamma \vdash ?(!?A \multimap !?B), ?\Delta} \rightarrow_R}{!?\Gamma \vdash ?(!?A \multimap !?B), ?\Delta} ?_R \quad \frac{\frac{\frac{!?\Gamma \vdash ?A, ?\Delta}{!?\Gamma \vdash !?A, ?\Delta} !_R \quad !?\Gamma, !?B \vdash ?\Delta}{!?\Gamma, !?A \multimap !?B \vdash ?\Delta} \rightarrow_L}{!?\Gamma, !?(?!?A \multimap !?B) \vdash ?\Delta} !?_L$$

Analysis from the point of view of linear logic

Call-by-name calculus with η -constraint: interpret sequents as $!?\Gamma \vdash ?\Delta$

$$\begin{array}{c}
 \frac{\frac{}{!?\Gamma, ?A \vdash ?A, ?\Delta} Ax}{!?\Gamma, !?A \vdash ?A, ?\Delta} !_L \quad \frac{\frac{}{!?\Gamma, ?A \vdash ?A, ?\Delta} Ax}{!?\Gamma, !?A \vdash ?A, ?\Delta} !_L \\
 \\
 \frac{!?\Gamma \vdash ?A, ?\Delta}{!?\Gamma \vdash ?A, ?\Delta} \mu \quad \frac{!?\Gamma, !?A \vdash ?\Delta}{!?\Gamma, !?A \vdash ?\Delta} \tilde{\mu} \\
 \\
 \frac{\frac{!?\Gamma \vdash ?A, ?\Delta}{!?\Gamma \vdash !?A, ?\Delta} !_R \quad !?\Gamma, !?A \vdash ?\Delta}{!?\Gamma \vdash ?\Delta} Cut
 \end{array}$$

This corresponds to marking all formulas t in LK_{tq}^η (see Danos-Joinet-Schellinx "A new deconstructive logic: LL")

$$\frac{\frac{\frac{!?\Gamma, !?A \vdash ?B, ?\Delta}{!?\Gamma, !?A \vdash !?B, ?\Delta} !_R}{!?\Gamma \vdash ?(!?A \multimap !?B), ?\Delta} \rightarrow_R}{!?\Gamma \vdash ?(!?A \multimap !?B), ?\Delta} ?_R \quad \frac{\frac{\frac{!?\Gamma \vdash ?A, ?\Delta}{!?\Gamma \vdash !?A, ?\Delta} !_R \quad !?\Gamma, !?B \vdash ?\Delta}{!?\Gamma, !?A \multimap !?B \vdash ?\Delta} \rightarrow_L}{!?\Gamma, !?(?!?A \multimap !?B) \vdash ?\Delta} !?_L$$

Analysis from the point of view of linear logic

Call-by-value calculus with η -constraint: interpret sequents as $!\Gamma \vdash ?!\Delta$

$$\begin{array}{c}
 \frac{\frac{}{!\Gamma, !A \vdash !A, ?!\Delta} Ax}{!\Gamma, !A \vdash ?!A, ?!\Delta} ?_R \quad \frac{\frac{}{!\Gamma, !A \vdash !A, ?!\Delta} Ax}{!\Gamma, !A \vdash ?!A, ?!\Delta} ?_R \\
 \\
 \frac{!\Gamma \vdash ?!A, ?!\Delta}{!\Gamma \vdash ?!A, ?!\Delta} \mu \quad \frac{!\Gamma, !A \vdash ?!\Delta}{!\Gamma, !A \vdash ?!\Delta} \tilde{\mu} \\
 \\
 \frac{!\Gamma \vdash ?!A, ?!\Delta \quad \frac{!\Gamma, !A \vdash ?!\Delta}{!\Gamma, ?!A \vdash ?!\Delta} ?_L}{!\Gamma \vdash ?!\Delta} Cut
 \end{array}$$

This corresponds to marking all formulas q in $LK_{\tau q}^\eta$ (see Danos-Joinet-Schellinx "A new deconstructive logic: LL")

$$\frac{\frac{!\Gamma, !A \vdash ?!B, ?!\Delta}{!\Gamma, ?!A \vdash ?!B, ?!\Delta} ?_L}{!\Gamma \vdash ?!A \multimap ?!B, ?!\Delta} \rightarrow_R \quad \frac{\frac{!\Gamma, !B \vdash ?!\Delta}{!\Gamma, ?!B \vdash ?!\Delta} ?_L \quad \frac{!\Gamma \vdash ?!A, ?!\Delta}{!\Gamma, ?!A \multimap ?!B \vdash ?!\Delta} \rightarrow_L}{!\Gamma, !(?!A \multimap ?!B) \vdash ?!\Delta} !_L$$

Delimited control

Delimited continuations (support for composing continuations)

- Felleisen [1987]: delimiter $\#$ and operator \mathcal{F} (*prompt* and *control*).
 - semantics in direct style ($C[\#E[\mathcal{F}(\lambda k.M)]] \mapsto C[\#M[\lambda x.E[x]/k]]$) but complex to understand
 - continuation passing style semantics achieved by Shan [2004]
- Danvy-Filinski [1989]: delimiter $\langle \rangle$ and operator \mathcal{S} (*reset* and *shift*)
 - *reset* similar to *prompt*
 - semantics historically in continuation passing style
 - easy semantics in direct style: $C[\#E[\mathcal{S}(\lambda k.M)]] \mapsto C[\#M[\lambda x.\#E[x]/k]]$
- The completeness properties of *shift/reset*:
 - Sitaram-Felleisen [1990]: *reset* (+ *parallel or*) provides completeness of λ_c wrt Plotkin's domains
 - Filinski [1994]: *shift/reset* can simulate all concrete monads in *direct style* (references, exceptions, ...)
- Ariola-Herbelin-Sabry [2004,2007]: calculus $\lambda_{c\hat{t}p}$ (aka $\lambda\mu\hat{\mu}tp$)
 - is call-by-value $\lambda\mu$ -calculus extended with one dynamic continuation variable
 - is equivalent to Danvy-Filinski *shift/reset*
 - alternative characterization: $\lambda\mu$ -calculus with *raise* and *abort*
- Call-by-name $\lambda\mu\hat{\mu}tp =$ Saurin's $\lambda\mu$ -calculus [2007]
 - Saurin's $\lambda\mu$ -calculus is untyped de Groote's variant of $\lambda\mu$ -calculus
 - Saurin's $\lambda\mu$ -calculus is observationally complete [2005] while untyped $\lambda\mu$ -calculus is not (David-Py [2001])

Call-by-value $\lambda\mu\hat{\mu}\text{tp}$ -calculus (a fine-grain *shift/reset* calculus)

Syntax

$$\begin{aligned} t, u &::= V \mid tt \mid \mu\alpha.c \mid \hat{\mu}\text{tp}.c \\ c &::= [\beta]t \mid [\text{tp}]t \\ V &::= x \mid \lambda x.t \end{aligned}$$

Reduction semantics

$$\begin{aligned} (\beta_v) \quad (\lambda x.t) V &\rightarrow t[x \leftarrow V] \\ (\mu_{app}) \quad (\mu\alpha.c) u &\rightarrow \mu\alpha.c[\alpha \leftarrow [\alpha]([\] u))] \\ (\mu'_{app}) \quad V (\mu\alpha.c) &\rightarrow \mu\alpha.c[\alpha \leftarrow [\alpha](V [\])] \\ (\mu_{var}) \quad [\beta]\mu\alpha.c &\rightarrow c[\alpha \leftarrow \beta] && \text{also if } \beta \text{ is tp} \\ (\eta_{\hat{\mu}v}) \quad \hat{\mu}\text{tp}.[\text{tp}]V &\rightarrow V && \text{even if tp occurs in } t \\ (\hat{\mu}_{var}) \quad [\text{tp}]\hat{\mu}\text{tp}.c &\rightarrow c \end{aligned}$$

Expressiveness

$$\begin{array}{lll} (\textit{shift}) & \mathcal{S}(\lambda k.t) \triangleq \mu\alpha.[\text{tp}](t[\lambda x.\hat{\mu}\text{tp}.[\alpha]x/k]) & (\textit{reset/prompt/“try”}) \quad \langle t \rangle \triangleq \hat{\mu}\text{tp}.[\text{tp}]t \\ (\textit{Felleisen’s } \mathcal{C}) & \mathcal{C}(\lambda k.t) \triangleq \mu\alpha.[\text{tp}](t[\lambda x.\mu_.[\alpha]x/k]) & (\textit{abort/raise}) \quad \mathcal{A}t \triangleq \mu_.[\text{tp}]t \\ (\textit{“callcc”}) & \mathcal{K}(\lambda k.t) \triangleq \mu\alpha.[\alpha](t[\lambda x.\mu_.[\alpha]x/k]) & \\ (\textit{memory read}) & \text{read} \triangleq \lambda().\mathcal{S}(\lambda k.\lambda x.\lambda s.(k \ s \ s)) & \\ (\textit{memory write}) & \text{write} \triangleq \lambda s.\mathcal{S}(\lambda k.\lambda x.\lambda_.(k \ () \ s)) & \end{array}$$

Call-by-name $\lambda\mu\hat{\mu}\text{tp}$ -calculus (a fine-grain *shift/reset* calculus)

Syntax

$$\begin{aligned} t, u &::= V \mid tt \mid \mu\alpha.c \mid \hat{\mu}\text{tp}.c \\ c &::= [\beta]t \mid [\text{tp}]t \\ V &::= x \mid \lambda x.t \end{aligned}$$

Reduction semantics

$$\begin{aligned} (\beta) \quad & (\lambda x.t) u \rightarrow t[x \leftarrow u] \\ (\mu_{app}) \quad & (\mu\alpha.c) u \rightarrow \mu\alpha.c[\alpha \leftarrow [\alpha]([] u)] \\ (\mu_{var}) \quad & [\beta]\mu\alpha.c \rightarrow c[\alpha \leftarrow \beta] \quad \beta \neq \text{tp} \\ (\hat{\mu}\text{tp}) \quad & \hat{\mu}\text{tp}.[\text{tp}]t \rightarrow t \quad \text{even if tp occurs in } t \\ (\eta_{\hat{\mu}}) \quad & [\text{tp}]\hat{\mu}\text{tp}.c \rightarrow c \end{aligned}$$

Expressiveness

Call-by-name $\lambda\mu\hat{\mu}\text{tp}$ -calculus (de Groot's style syntax)

Syntax

$$\begin{aligned} t, u &::= V \mid tt \mid \mu\alpha.t \mid [\alpha]t \\ V &::= x \mid \lambda x.t \end{aligned}$$

Reduction semantics

$$\begin{aligned} (\beta) \quad (\lambda x.t) u &\rightarrow t[x \leftarrow u] \\ (\mu_{app}) \quad (\mu\alpha.c) u &\rightarrow \mu\alpha.c[\alpha \leftarrow [\alpha]([] u)] \\ (\mu_{var}) \quad [\beta]\mu\alpha.c &\rightarrow c[\alpha \leftarrow \beta] \end{aligned}$$

Correspondence with full syntax

$$\begin{aligned} \mu\alpha.t &\triangleq \mu\alpha.[\text{tp}]t \\ [\alpha]t &\triangleq \hat{\mu}\text{tp}.[\alpha]t \end{aligned}$$

No such light syntax for call-by-value!

$\bar{\lambda}\mu\tilde{\mu}\hat{\mu}\text{tp}$ -calculus

(work in progress)

Syntax

Commands	$c ::= \langle v \ e \rangle$
Values	$V ::= x \mid \lambda x.v$
Terms	$v ::= V \mid \mu\alpha.c \mid \hat{\mu}\text{tp}.c$
Linear ev. contexts	$E ::= \alpha \mid v \cdot e$
Evaluation contexts	$e ::= E \mid \tilde{\mu}x.c \mid \text{tp}$

Call-by-name reduction semantics

(\rightarrow)	$\langle \lambda x.v_1 \ v_2 \cdot e \rangle \rightarrow \langle v_2 \ \tilde{\mu}x.\langle v_1 \ e \rangle \rangle$
(μ)	$\langle \mu\alpha.c \ E \rangle \rightarrow c[\alpha \leftarrow E]$
$(\tilde{\mu})$	$\langle v \ \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow v]$
$(\hat{\mu}^{var})$	$\langle \hat{\mu}\text{tp}.c \ \text{tp} \rangle \rightarrow c$
$(\eta_{\hat{\mu}})$	$\hat{\mu}\text{tp}.\langle v \ \text{tp} \rangle \rightarrow v$

$\bar{\lambda}\mu\tilde{\mu}\hat{\mu}\text{tp}$ -calculus

(work in progress)

Syntax

Commands	$c ::= \langle v \ e \rangle$
Values	$V ::= x \mid \lambda x.v$
Terms	$v ::= V \mid \mu\alpha.c \mid \hat{\mu}\text{tp}.c$
Linear ev. contexts	$E ::= \alpha \mid v \cdot e$
Evaluation contexts	$e ::= E \mid \tilde{\mu}x.c \mid \text{tp}$

Call-by-value reduction semantics

(\rightarrow)	$\langle \lambda x.v_1 \ v_2 \cdot e \rangle \rightarrow \langle v_2 \ \tilde{\mu}x.\langle v_1 \ e \rangle \rangle$
(μ)	$\langle \mu\alpha.c \ e \rangle \rightarrow c[\alpha \leftarrow e]$
$(\tilde{\mu})$	$\langle V \ \tilde{\mu}x.c \rangle \rightarrow c[x \leftarrow V]$
$(\hat{\mu}^{var})$	$\langle \hat{\mu}\text{tp}.c \ \text{tp} \rangle \rightarrow c$
$(\eta_{\hat{\mu}})$	$\hat{\mu}\text{tp}.\langle V \ \text{tp} \rangle \rightarrow V$

Summary

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

The duality finds limits in dependent type theory.

Incidentally: how to interpret Gentzen's cross-cuts?

Delimiters of continuations are promising (completeness properties, side-effects)

French manuscript available at <http://pauillac.inria.fr/~herbelin/habilitation/memoire.ps>

(but cps for call-by-name dynamically binding is buggy)

Cross-cuts reminded

$$\begin{array}{c}
 \frac{\frac{\Gamma \vdash A, \bar{A}, \Delta \quad \Gamma, \bar{A}, \bar{A} \vdash \Delta}{\Gamma \vdash A, \Delta} \quad \frac{\Gamma \vdash \underline{A}, \underline{A}, \Delta \quad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma, A \vdash \Delta}}{\Gamma \vdash \Delta} \\
 \\
 \uparrow \text{ (Cross-cuts) } \\
 \frac{\Gamma \vdash A, A, \Delta \quad \Gamma, A, A \vdash \Delta}{\Gamma \vdash \Delta} \\
 \begin{array}{cc}
 \text{(CBN) } \swarrow & \searrow \text{ (CBV) }
 \end{array} \\
 \frac{\frac{\Gamma \vdash \underline{A}, \underline{A}, \Delta \quad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma \vdash A, \bar{A}, \Delta} \quad \frac{\Gamma, \bar{A} \vdash \Delta}{\Gamma, \bar{A} \vdash \Delta}}{\Gamma \vdash A, \Delta} \quad \frac{\Gamma \vdash \underline{A}, \underline{A}, \Delta \quad \Gamma, \underline{A}, A \vdash \Delta}{\Gamma, A \vdash \Delta} \quad \frac{\Gamma \vdash A, \bar{A}, \Delta \quad \Gamma, \bar{A}, \bar{A} \vdash \Delta}{\Gamma \vdash A, \Delta} \quad \frac{\Gamma, \underline{A}, A \vdash \Delta}{\Gamma, \underline{A}, A \vdash \Delta}}{\Gamma \vdash \Delta} \quad \frac{\Gamma \vdash A, \bar{A}, \Delta \quad \Gamma, \bar{A}, \bar{A} \vdash \Delta}{\Gamma \vdash A, \Delta} \quad \frac{\Gamma, \underline{A}, A \vdash \Delta}{\Gamma, \underline{A}, A \vdash \Delta}}{\Gamma \vdash \Delta}
 \end{array}$$