The Call-by-Value λ-Calculus from a Linear Logic Perspective

Giulio Guerrieri

LIS, Aix-Marseille Université (Marseille, France)

Workshop in honour of Thomas Ehrhard's 60 years

Paris, 30 September 2022

Outline

Outline

A specific λ-calculus among a plethora of λ-calculi

The λ-calculus is the model of computation underlying

- functional programming languages (Haskell, OCaml, ...)
- proof assistants (Coq, Isabelle/Hol, ...).

Actually, there are many λ-calculi, depending on

- the evaluation mechanism (e.g., call-by-name, call-by-value, call-by-need);
- computational feature the calculus aims to model (e.g., pure, non-deterministic);
- the type system (e.g. untyped, simply typed, second order).

In this talk: pure untyped call-by-value λ-calculus (mainly).

A specific λ-calculus among a plethora of λ-calculi

The λ-calculus is the model of computation underlying

- functional programming languages (Haskell, OCaml, ...)
- proof assistants (Coq, Isabelle/Hol, ...).

Actually, there are many λ-calculi, depending on

- the evaluation mechanism (e.g., call-by-name, call-by-value, call-by-need);
- computational feature the calculus aims to model (e.g., pure, non-deterministic);
- the type system (e.g. untyped, simply typed, second order).

In this talk: pure untyped call-by-value λ-calculus (mainly).

A specific λ-calculus among a plethora of λ-calculi

The λ-calculus is the model of computation underlying

- functional programming languages (Haskell, OCaml, ...)
- proof assistants (Coq, Isabelle/Hol, ...).

Actually, there are many λ-calculi, depending on

- the evaluation mechanism (e.g., call-by-name, call-by-value, call-by-need);
- computational feature the calculus aims to model (e.g., pure, non-deterministic);
- the type system (e.g. untyped, simply typed, second order).

In this talk: pure untyped call-by-value λ-calculus (mainly).

Call-by-Name vs. Call-by-Value (for dummies)

- Call-by-Name (CbN): pass the argument to the calling function before evaluating it.
- Call-by-Value (CbV): pass the argument to the calling function after evaluating it.

Call-by-Name vs. Call-by-Value (for dummies)

- Call-by-Name (CbN): pass the argument to the calling function before evaluating it.
- Call-by-Value (CbV): pass the argument to the calling function after evaluating it.

$$\begin{tikzcd}
& (\lambda x.\, x + x)(2 * 3) \arrow[dl, "CbN"'] \arrow[dr, "CbV"] & \\
(2 * 3) + (2 * 3) \arrow[d, "CbN"'] & & (\lambda x.\, x + x)6 \arrow[d, "CbV"] \\
6 + (2 * 3) \arrow[d, "CbN"'] & & 6 + 6 \arrow[d, "CbV"] \\
6 + 6 \arrow[d, "CbN"'] & & 12 \\
12 & &
\end{tikzcd}$$

Call-by-Name vs. Call-by-Value (for dummies)

- Call-by-Name (CbN): pass the argument to the calling function before evaluating it.
- Call-by-Value (CbV): pass the argument to the calling function after evaluating it.

$$
\begin{tikzcd}
 & (\lambda y.\lambda x.\,x)\Omega \arrow[dl, "CbN"'] \arrow[dr, "CbV"] & \\
\lambda x.x & & (\lambda y.\lambda x.\,x)\Omega \arrow[d, "CbV"] \\
 & & \vdots
\end{tikzcd}
$$

(Ω is a diverging program)

Summing up, CbV is eager, that is,

1. CbV is smarter than CbN when the argument must be duplicated;
2. CbV is sillier than CbN when the argument must be discarded.

Plotkin's Call-by-Value λ-calculus [Plo75]

Terms: $s, t, u ::= v \mid tu$ Values: $v ::= x \mid \lambda x.t$

CbV reduction: $(\lambda x.t)v \to_{\beta_v} t\{v/x\}$ (restriction to—CbN—β-rule)

Why? Closer to real implementation of most programming languages & proof assistants.

CbN and CbV λ-calculi have different operational and denotational semantics
$\rightsquigarrow$ in general, it is impossible to derive a property for CbV from CbN, or vice versa.

Examples, with $I := \lambda z.z$ (identity) and $\delta := \lambda z.zz$ (duplicator):

1. $(\lambda y.I)(\delta\delta)$ β-normalizes but β_v-diverges

$$(\lambda y.I)(\delta\delta) \to_\beta I \qquad (\lambda y.I)(\delta\delta) \to_{\beta_v} (\lambda y.I)(\delta\delta) \to_{\beta_v} \dots$$

2. $(\lambda x.\delta)(xx)\delta$ is β_v-normal but β-divergent: $(\lambda x.\delta)(xx)\delta \to_\beta \delta\delta \to_\beta \delta\delta \to_\beta \dots$

Plotkin's Call-by-Value λ-calculus [Plo75]

Terms: $s, t, u ::= v \mid tu$ Values: $v ::= x \mid \lambda x.t$

CbV reduction: $(\lambda x.t)v \to_{\beta_v} t\{v/x\}$ (restriction to—CbN—β-rule)

Why? Closer to real implementation of most programming languages & proof assistants.

CbN and CbV λ-calculi have different operational and denotational semantics
$\rightsquigarrow$ in general, it is impossible to derive a property for CbV from CbN, or vice versa.

Examples, with $I := \lambda z.z$ (identity) and $\delta := \lambda z.zz$ (duplicator):

1. $(\lambda y.I)(\delta\delta)$ β-normalizes but β_v-diverges

$$(\lambda y.I)(\delta\delta) \to_\beta I \qquad (\lambda y.I)(\delta\delta) \to_{\beta_v} (\lambda y.I)(\delta\delta) \to_{\beta_v} \dots$$

2. $(\lambda x.\delta)(xx)\delta$ is β_v-normal but β-divergent: $(\lambda x.\delta)(xx)\delta \to_\beta \delta\delta \to_\beta \delta\delta \to_\beta \dots$

Plotkin's Call-by-Value λ-calculus [Plo75]

Terms: $s, t, u ::= v \mid tu$ Values: $v ::= x \mid \lambda x.t$

CbV reduction: $(\lambda x.t)v \to_{\beta_v} t\{v/x\}$ (restriction to—CbN—β-rule)

Why? Closer to real implementation of most programming languages & proof assistants.

CbN and CbV λ-calculi have different operational and denotational semantics
$\rightsquigarrow$ in general, it is impossible to derive a property for CbV from CbN, or vice versa.

Examples, with $I := \lambda z.z$ (identity) and $\delta := \lambda z.zz$ (duplicator):

1. $(\lambda y.I)(\delta\delta)$ β-normalizes but β_v-diverges

$$(\lambda y.I)(\delta\delta) \to_\beta I \qquad (\lambda y.I)(\delta\delta) \to_{\beta_v} (\lambda y.I)(\delta\delta) \to_{\beta_v} \dots$$

2. $(\lambda x.\delta)(xx)\delta$ is β_v-normal but β-divergent: $(\lambda x.\delta)(xx)\delta \to_\beta \delta\delta \to_\beta \delta\delta \to_\beta \dots$

Plotkin's Call-by-Value λ-calculus [Plo75]

Terms: $s, t, u ::= v \mid tu$ Values: $v ::= x \mid \lambda x.t$

CbV reduction: $(\lambda x.t)v \to_{\beta_v} t\{v/x\}$ (restriction to—CbN—β-rule)

Why? Closer to real implementation of most programming languages & proof assistants.

CbN and CbV λ-calculi have different operational and denotational semantics
$\rightsquigarrow$ in general, it is impossible to derive a property for CbV from CbN, or vice versa.

Examples, with $I := \lambda z.z$ (identity) and $\delta := \lambda z.zz$ (duplicator):

1. $(\lambda y.I)(\delta\delta)$ β-normalizes but β_v-diverges

$$(\lambda y.I)(\delta\delta) \to_{\beta} I \qquad (\lambda y.I)(\delta\delta) \to_{\beta_v} (\lambda y.I)(\delta\delta) \to_{\beta_v} \dots$$

2. $(\lambda x.\delta)(xx)\delta$ is β_v-normal but β-divergent: $(\lambda x.\delta)(xx)\delta \to_{\beta} \delta\delta \to_{\beta} \delta\delta \to_{\beta} \dots$

Outline

A symptom that Plotkin's CbV is sick: Contextual equivalence

Def. Terms t, t' are contextually equivalent if they are observably indistinguishable, i.e., for every context $\mathtt{C}$, $\mathtt{C}\langle t\rangle \to_{\beta_v}^* v$ (for some value v) iff $\mathtt{C}\langle t'\rangle \to_{\beta_v}^* v'$ (for some value v')

Consider the terms (with $\delta := \lambda z.zz$ as usual)

$$\delta_1 := (\lambda x.\delta)(xx)\delta \qquad \delta_3 := \delta((\lambda x.\delta)(xx))$$

δ_1 and δ_3 are β_v-normal but contextually equivalent to $\delta\delta$ (which is β_v-divergent)!

The "energy" (i.e. divergence) in δ_1 and δ_3 is only potential, in $\delta\delta$ is kinetic!

Why are δ_1 and δ_3 stuck? Why cannot we transform their potential energy in kinetic?
It seems that in Plotkin's CbV λ-calculus something is missing...

A symptom that Plotkin's CbV is sick: Contextual equivalence

Def. Terms t, t' are contextually equivalent if they are observably indistinguishable, i.e., for every context $\mathtt{C}$, $\mathtt{C}\langle t\rangle \to_{\beta_v}^* v$ (for some value v) iff $\mathtt{C}\langle t'\rangle \to_{\beta_v}^* v'$ (for some value v')

Consider the terms (with $\delta \coloneqq \lambda z.zz$ as usual)

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

δ_1 and δ_3 are β_v-normal but contextually equivalent to $\delta\delta$ (which is β_v-divergent)!

The "energy" (i.e. divergence) in δ_1 and δ_3 is only potential, in $\delta\delta$ is kinetic!

Why are δ_1 and δ_3 stuck? Why cannot we transform their potential energy in kinetic?
It seems that in Plotkin's CbV λ-calculus something is missing...

A symptom that Plotkin's CbV is sick: Contextual equivalence

Def. Terms t, t' are contextually equivalent if they are observably indistinguishable, i.e., for every context $\mathtt{C}$, $\mathtt{C}\langle t\rangle \to^*_{\beta_v} v$ (for some value v) iff $\mathtt{C}\langle t'\rangle \to^*_{\beta_v} v'$ (for some value v')

Consider the terms (with $\delta := \lambda z.zz$ as usual)

$$\delta_1 := (\lambda x.\delta)(xx)\delta \qquad \delta_3 := \delta((\lambda x.\delta)(xx))$$

δ_1 and δ_3 are β_v-normal but contextually equivalent to $\delta\delta$ (which is β_v-divergent)!

The "energy" (i.e. divergence) in δ_1 and δ_3 is only potential, in $\delta\delta$ is kinetic!

Why are δ_1 and δ_3 stuck? Why cannot we transform their potential energy in kinetic? It seems that in Plotkin's CbV λ-calculus something is missing...

A second symptom that Plotkin's CbV is sick: Solvability (1 of 2)

In a calculus X, a term t is solvable if there is a head context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \to_{\mathsf{X}}^{*} \boldsymbol{I}$.

In the CbN λ-calculus, solvability is well-studied and has an elegant theory.

1. Internal operational characterization: t is CbN-solvable iff t is head β-normalizing.
2. Every β-normalizing term is CbN-solvable, but the converse fails (e.g. Y).
3. The λ-theory that equates all CbN-unsolvable terms is consistent.
4. CbN-unsolvable terms represent undefined partial recursive functions.

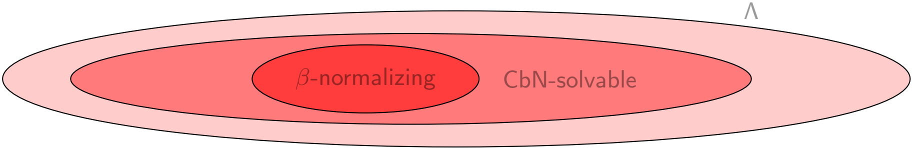
Moral: CbN-solvable terms are all and only the meaningful terms in CbN.
$\rightsquigarrow$ CbN-unsolvable terms are meaningless, and "heavily" divergent.

A second symptom that Plotkin's CbV is sick: Solvability (1 of 2)

In a calculus X, a term t is solvable if there is a head context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \to_{\mathsf{X}}^* I$.

In the CbN λ-calculus, solvability is well-studied and has an elegant theory.

1. Internal operational characterization: t is CbN-solvable iff t is head β-normalizing.
2. Every β-normalizing term is CbN-solvable, but the converse fails (e.g. Y).
3. The λ-theory that equates all CbN-unsolvable terms is consistent.
4. CbN-unsolvable terms represent undefined partial recursive functions.

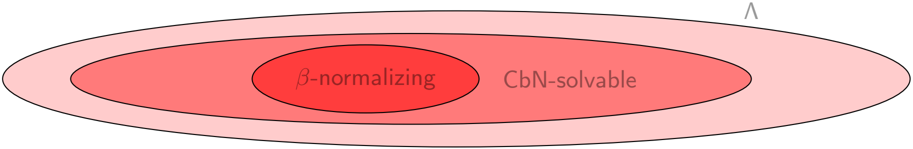


Moral: CbN-solvable terms are all and only the meaningful terms in CbN.
$\rightsquigarrow$ CbN-unsolvable terms are meaningless, and "heavily" divergent.

In a calculus X, a term t is solvable if there is a head context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \to_{\mathsf{X}}^{*} I$.

In the CbN λ-calculus, solvability is well-studied and has an elegant theory.

1. Internal operational characterization: t is CbN-solvable iff t is head β-normalizing.
2. Every β-normalizing term is CbN-solvable, but the converse fails (e.g. Y).
3. The λ-theory that equates all CbN-unsolvable terms is consistent.
4. CbN-unsolvable terms represent undefined partial recursive functions.



Moral: CbN-solvable terms are all and only the meaningful terms in CbN.
$\rightsquigarrow$ CbN-unsolvable terms are meaningless, and "heavily" divergent.

In CbV, all these results are false! In particular,

1. There is no internal operational characterization of CbV-solvability.
2. The set of CbV-solvable terms does not include the set of β_v-normalizing ones.

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

δ_1 and δ_3 are β_v-normal but CbV-unsolvable ($\delta\delta$ is CbV-unsolvable too)!

Moral: If we stick to the idea CbV-solvable = meaningful in CbV, we have two options:

1. We change the notion of CbV-solvability (i.e., we change the semantics of CbV);
2. We change the notion of reduction in Plotkin's CbV (i.e., we change its syntax).

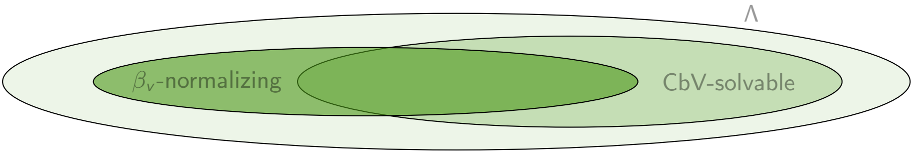
A second symptom that Plotkin's CbV is sick: Solvability (2 of 2)

In CbV, all these results are false! In particular,

1. There is no internal operational characterization of CbV-solvability.
2. The set of CbV-solvable terms does not include the set of β_v-normalizing ones.

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

δ_1 and δ_3 are β_v-normal but CbV-unsolvable ($\delta\delta$ is CbV-unsolvable too)!



Moral: If we stick to the idea CbV-solvable = meaningful in CbV, we have two options:

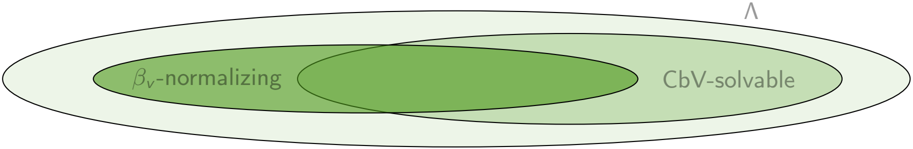
1. We change the notion of CbV-solvability (i.e., we change the semantics of CbV);
2. We change the notion of reduction in Plotkin's CbV (i.e., we change its syntax).

In CbV, all these results are false! In particular,

1. There is no internal operational characterization of CbV-solvability.
2. The set of CbV-solvable terms does not include the set of β_v-normalizing ones.

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

δ_1 and δ_3 are β_v-normal but CbV-unsolvable ($\delta\delta$ is CbV-unsolvable too)!



Moral: If we stick to the idea CbV-solvable = meaningful in CbV, we have two options:

1. We change the notion of CbV-solvability (i.e., we change the semantics of CbV);
2. We change the notion of reduction in Plotkin's CbV (i.e., we change its syntax).

[Ehr12] defined a non-idempotent intersection type system for Plotkin's CbV λ-calculus.

Linear types $L ::= M \multimap N$ Multi types $M, N ::= [L_1, \dots, L_n] \quad n \geq 0$

Idea: $[L, L', L'] \approx L \wedge L' \wedge L' \neq L \wedge L'$ (commutative, associative, non-idempotent $\wedge$).

$$\frac{}{x : [L] \vdash x : L}\,ax \qquad \frac{\Gamma, x : M \vdash t : N}{\Gamma \vdash \lambda x.t : M \multimap N}\,\lambda \qquad \frac{\Gamma_1 \vdash v : L_1 \quad \overset{n \geq 0}{\cdots} \quad \Gamma_n \vdash v : L_n}{\Gamma \vdash v : [L_1, \dots, L_n]}\,! \qquad \frac{\Gamma \vdash t : [M \multimap N] \quad \Delta \vdash s : M}{\Gamma \vdash ts : N}\,@$$

Idea: A term $t : [L, L', L']$ can be used once as a data of type L, twice as a data of type L'.

Rmk: The constructor for multi types (rule !) can be used only by values!

A third symptom that Plotkin's CbV is sick: denotational semantics.

[Ehr12] defined a non-idempotent intersection type system for Plotkin's CbV λ-calculus.

Linear types $L ::= M \multimap N$ Multi types $M, N ::= [L_1, \ldots, L_n] \quad n \geq 0$

Idea: $[L, L', L'] \approx L \wedge L' \wedge L' \neq L \wedge L'$ (commutative, associative, non-idempotent $\wedge$).

$$\frac{}{x : [L] \vdash x : L}\,ax \qquad \frac{\Gamma, x : M \vdash t : N}{\Gamma \vdash \lambda x.t : M \multimap N}\,\lambda \qquad \frac{\Gamma_1 \vdash v : L_1 \quad \overset{n \geq 0}{\ldots} \quad \Gamma_n \vdash v : L_n}{\Gamma \vdash v : [L_1, \ldots, L_n]}\,! \qquad \frac{\Gamma \vdash t : [M \multimap N] \quad \Delta \vdash s : M}{\Gamma \vdash ts : N}\,@$$

Idea: A term $t : [L, L', L']$ can be used once as a data of type L, twice as a data of type L'.

Rmk: The constructor for multi types (rule !) can be used only by values!

Non-idempotent intersection types define a denotational model: relational semantics

$$[\![t]\!]_{\vec{x}} = \{(\Gamma, M) \mid \Gamma \vdash t : M \text{ is derivable}\}$$

Theorem (Invariance, [Ehr12])

If $t \to_{\beta_v} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.

Theorem (Correctness, [Ehr12])

If $[\![t]\!]_{\vec{x}} \neq \emptyset$ then t is normalizing for "weak" β_v-reduction (not reducing under λ's).

The converse (completeness) fail!

$$[\![\delta_1]\!] = \emptyset = [\![\delta_3]\!] \quad (\text{and } [\![\delta\delta]\!] = \emptyset \text{ too!})$$

but δ_1 and δ_3 are β_v-normal, while $\delta\delta$ is β_v-divergent!

Rmk: Not only in relational semantics but also in other denotational models of CbV!

Non-idempotent intersection types define a denotational model: relational semantics

$$[\![t]\!]_{\vec{x}} = \{(\Gamma, M) \mid \Gamma \vdash t : M \text{ is derivable}\}$$

Theorem (Invariance, [Ehr12])

If $t \to_{\beta_v} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.

Theorem (Correctness, [Ehr12])

If $[\![t]\!]_{\vec{x}} \neq \emptyset$ then t is normalizing for "weak" β_v-reduction (not reducing under λ's).

The converse (completeness) fail!

$$[\![\delta_1]\!] = \emptyset = [\![\delta_3]\!] \quad (\text{and } [\![\delta\delta]\!] = \emptyset \text{ too!})$$

but δ_1 and δ_3 are β_v-normal, while $\delta\delta$ is β_v-divergent!

Rmk: Not only in relational semantics but also in other denotational models of CbV!

A third symptom that Plotkin's CbV is sick: denotational semantics. (2 of 2)

Non-idempotent intersection types define a denotational model: relational semantics

$$[\![t]\!]_{\vec{x}} = \{(\Gamma, M) \mid \Gamma \vdash t : M \text{ is derivable}\}$$

Theorem (Invariance, [Ehr12])

If $t \to_{\beta_v} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.

Theorem (Correctness, [Ehr12])

If $[\![t]\!]_{\vec{x}} \neq \emptyset$ then t is normalizing for "weak" β_v-reduction (not reducing under λ's).

The converse (completeness) fail!

$$[\![\delta_1]\!] = \emptyset = [\![\delta_3]\!] \quad \text{(and } [\![\delta\delta]\!] = \emptyset \text{ too!)}$$

but δ_1 and δ_3 are β_v-normal, while $\delta\delta$ is β_v-divergent!

Rmk: Not only in relational semantics but also in other denotational models of CbV!

Summing up: a mismatch between syntax and semantics

In Plotkin's CbV λ-calculus there is a mismatch between syntax and semantics.

There are terms, such as

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

that are β_v-normal but their semantics is the same as $\delta\delta$, which is β_v-divergent!

- semantics: context equivalence, solvability, denotational models, ...

Somehow, in Plotkin's CbV λ-calculus, β_v-reduction is "not enough".

- Can we extend β_v so that δ_1 and δ_3 are divergent?
- But we want to keep a CbV discipline:

$$(\lambda x.I)(\delta\delta) \text{ is } \beta_v\text{-divergent (but } \beta\text{-normalizing)}$$

Idea: Let us see what happens in CbV from a proof-theoretic viewpoint (Curry-Howard).

Summing up: a mismatch between syntax and semantics

In Plotkin's CbV λ-calculus there is a mismatch between syntax and semantics.

There are terms, such as

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

that are β_v-normal but their semantics is the same as $\delta\delta$, which is β_v-divergent!

- semantics: context equivalence, solvability, denotational models, ...

Somehow, in Plotkin's CbV λ-calculus, β_v-reduction is "not enough".

- Can we extend β_v so that δ_1 and δ_3 are divergent?
- But we want to keep a CbV discipline:

$$(\lambda x.I)(\delta\delta) \text{ is } \beta_v\text{-divergent (but } \beta\text{-normalizing)}$$

Idea: Let us see what happens in CbV from a proof-theoretic viewpoint (Curry-Howard).

Summing up: a mismatch between syntax and semantics

In Plotkin's CbV λ-calculus there is a mismatch between syntax and semantics.

There are terms, such as

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

that are β_v-normal but their semantics is the same as $\delta\delta$, which is β_v-divergent!

- semantics: context equivalence, solvability, denotational models, ...

Somehow, in Plotkin's CbV λ-calculus, β_v-reduction is "not enough".

- Can we extend β_v so that δ_1 and δ_3 are divergent?
- But we want to keep a CbV discipline:

$$(\lambda x.I)(\delta\delta) \text{ is } \beta_v\text{-divergent (but } \beta\text{-normalizing)}$$

Idea: Let us see what happens in CbV from a proof-theoretic viewpoint (Curry-Howard).

What I learned from Thomas when I was his PhD student

T: In the eternal fight between syntax and semantics, the semantics always wins.

G: I see.

T: Use linear logic and its semantics as a guideline.

G: Thank you!

Outline

The role of linear logic with respect to λ-calculi

Girard's linear logic (1987) provides new concepts and tools to study λ-calculi:

1. denotational models of linear logic provides denotational models for λ-calculi;
2. clear notion of resource and linear consumption
   $f : A \multimap B \;\approx\; f$ consumes a value of type A and transforms it into a value of type B;
3. quantitative analysis of computation
   - semantic tools to study execution time (De Carvalho *et al.*));
   - "compatible" with cost models (Accattoli *et al.*).
4. ...

LL also hints how to modify syntax and dynamics of λ-calculi to have "good properties".

The role of linear logic with respect to λ-calculi

Girard's linear logic (1987) provides new concepts and tools to study λ-calculi:

1. denotational models of linear logic provides denotational models for λ-calculi;
2. clear notion of resource and linear consumption
   $f : A \multimap B \;\approx\; f$ consumes a value of type A and transforms it into a value of type B;
3. quantitative analysis of computation
   - semantic tools to study execution time (De Carvalho *et al.*));
   - "compatible" with cost models (Accattoli *et al.*).
4. ...

LL also hints how to modify syntax and dynamics of λ-calculi to have "good properties".

The role of linear logic with respect to λ-calculi

Girard's linear logic (1987) provides new concepts and tools to study λ-calculi:

1. denotational models of linear logic provides denotational models for λ-calculi;
2. clear notion of resource and linear consumption
   $f : A \multimap B \;\approx\; f$ consumes a value of type A and transforms it into a value of type B;
3. quantitative analysis of computation
   - semantic tools to study execution time (De Carvalho *et al.*));
   - "compatible" with cost models (Accattoli *et al.*).
4. ...

LL also hints how to modify syntax and dynamics of λ-calculi to have "good properties".

The role of linear logic with respect to λ-calculi

Girard's linear logic (1987) provides new concepts and tools to study λ-calculi:

1. denotational models of linear logic provides denotational models for λ-calculi;
2. clear notion of resource and linear consumption
   $f : A \multimap B \;\approx\; f$ consumes a value of type A and transforms it into a value of type B;
3. quantitative analysis of computation
   - semantic tools to study execution time (De Carvalho *et al.*));
   - "compatible" with cost models (Accattoli *et al.*).
4. ...

LL also hints how to modify syntax and dynamics of λ-calculi to have "good properties".

The role of linear logic with respect to λ-calculi

Girard's linear logic (1987) provides new concepts and tools to study λ-calculi:

1. denotational models of linear logic provides denotational models for λ-calculi;
2. clear notion of resource and linear consumption
   $f : A \multimap B \;\approx\; f$ consumes a value of type A and transforms it into a value of type B;
3. quantitative analysis of computation
   - semantic tools to study execution time (De Carvalho *et al.*));
   - "compatible" with cost models (Accattoli *et al.*).
4. ...

LL also hints how to modify syntax and dynamics of λ-calculi to have "good properties".

The Curry-Howard-Girard correspondence

Logic		Computer Science
formula	$\leftrightsquigarrow$	type
proof	$\leftrightsquigarrow$	program
cut-elimination	$\leftrightsquigarrow$	evaluation
coherence	$\leftrightsquigarrow$	termination
different encodings of intuitionistic arrow in LL	$\leftrightsquigarrow$	different evaluation mechanisms

$\rightsquigarrow$ Tools from intuitionistic linear logic (ILL) can be used to study properties of:

- call-by-name evaluation via Girard's translation $(\cdot)^{\mathtt{N}}$,
- call-by-value evaluation via Girard's translation $(\cdot)^{\mathtt{V}}$.

The Curry-Howard-Girard correspondence

Logic		Computer Science
formula	$\leftrightsquigarrow$	type
proof	$\leftrightsquigarrow$	program
cut-elimination	$\leftrightsquigarrow$	evaluation
coherence	$\leftrightsquigarrow$	termination
different encodings of intuitionistic arrow in LL	$\leftrightsquigarrow$	different evaluation mechanisms

$\rightsquigarrow$ Tools from intuitionistic linear logic (ILL) can be used to study properties of:

- call-by-name evaluation via Girard's translation $(\cdot)^{\mathtt{N}}$,
- call-by-value evaluation via Girard's translation $(\cdot)^{\mathtt{V}}$.

The Curry-Howard-Girard correspondence

Logic		Computer Science
formula	$\leftrightsquigarrow$	type
proof	$\leftrightsquigarrow$	program
cut-elimination	$\leftrightsquigarrow$	evaluation
coherence	$\leftrightsquigarrow$	termination
different encodings of intuitionistic arrow in LL	$\leftrightsquigarrow$	different evaluation mechanisms

$\rightsquigarrow$ Tools from intuitionistic linear logic (ILL) can be used to study properties of:

- call-by-name evaluation via Girard's translation $(\cdot)^{\mathtt{N}}$,
- call-by-value evaluation via Girard's translation $(\cdot)^{\mathtt{V}}$.

The two Girard's translations of IL into ILL (1987)

well-known translation $(\cdot)^{\mathtt{N}}$

$$X^{\mathtt{N}} = X$$
$$(A \to B)^{\mathtt{N}} = !A^{\mathtt{N}} \multimap B^{\mathtt{N}}$$
$$(\Gamma \vdash A)^{\mathtt{N}} = !\Gamma^{\mathtt{N}} \vdash A^{\mathtt{N}}$$

"boring" translation $(\cdot)^{\mathtt{V}}$

$$X^{\mathtt{V}} = X$$
$$(A \to B)^{\mathtt{V}} = !A^{\mathtt{V}} \multimap !B^{\mathtt{V}}$$
$$(\Gamma \vdash A)^{\mathtt{V}} = !\Gamma^{\mathtt{V}} \vdash !A^{\mathtt{V}}$$

$$\begin{tikzcd} \mathsf{IL} \arrow[r, bend left, "(\cdot)^{\mathtt{N}}"] \arrow[r, bend right, "(\cdot)^{\mathtt{V}}"'] & \mathsf{ILL} \end{tikzcd} \quad \text{(and then proof-nets)}$$

simply typed Λ = IL (via Curry-Howard)

(untyped) Λ = IL + unique atomic type o + type identity $o = o \to o$

$$\begin{tikzcd} \text{(untyped) } \Lambda \arrow[r, bend left, "(\cdot)^{\mathtt{N}}", "o=!o\multimap o"'] \arrow[r, bend right, "o=!o\multimap !o", "(\cdot)^{\mathtt{V}}"'] & \mathsf{ILL} \end{tikzcd} \quad \text{(and then proof-nets)}$$

The two Girard's translations of IL into ILL (1987)

well-known translation $(\cdot)^{\mathtt{N}}$

$$X^{\mathtt{N}} = X$$
$$(A \to B)^{\mathtt{N}} = !A^{\mathtt{N}} \multimap B^{\mathtt{N}}$$
$$(\Gamma \vdash A)^{\mathtt{N}} = !\Gamma^{\mathtt{N}} \vdash A^{\mathtt{N}}$$

"boring" translation $(\cdot)^{\mathtt{V}}$

$$X^{\mathtt{V}} = X$$
$$(A \to B)^{\mathtt{V}} = !A^{\mathtt{V}} \multimap !B^{\mathtt{V}}$$
$$(\Gamma \vdash A)^{\mathtt{V}} = !\Gamma^{\mathtt{V}} \vdash !A^{\mathtt{V}}$$

$$\begin{tikzcd} \mathrm{IL} \arrow[r, bend left, "(\cdot)^{\mathtt{N}}"] \arrow[r, bend right, "(\cdot)^{\mathtt{V}}"'] & \mathrm{ILL} \end{tikzcd}$$
(and then proof-nets)

simply typed Λ = IL (via Curry-Howard)

(untyped) Λ = IL + unique atomic type o + type identity $o = o \to o$

$$\begin{tikzcd} \text{(untyped) } \Lambda \arrow[r, bend left, "(\cdot)^{\mathtt{N}}", "o = !o \multimap o"'] \arrow[r, bend right, "o = !o \multimap !o", "(\cdot)^{\mathtt{V}}"'] & \mathrm{ILL} \end{tikzcd}$$
(and then proof-nets)

Girard's first translation: $(\cdot)^{\mathtt{N}}$

$$X^{\mathtt{N}} = X$$
$$(A \to B)^{\mathtt{N}} = {!}A^{\mathtt{N}} \multimap B^{\mathtt{N}}$$
$$(\Gamma \vdash A)^{\mathtt{N}} = {!}\Gamma^{\mathtt{N}} \vdash A^{\mathtt{N}}$$

(natural deduction for IL) (sequent calculus for ILL)

$$\dfrac{}{x : A \vdash x : A}\,ax \quad \rightsquigarrow \quad \dfrac{\dfrac{}{A^{\mathtt{N}} \vdash A^{\mathtt{N}}}\,ax}{{!}A^{\mathtt{N}} \vdash A^{\mathtt{N}}}\,der$$

$$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x\, M : A \to B}\,\to_i \quad \rightsquigarrow \quad \dfrac{{!}\Gamma^{\mathtt{N}}, {!}A^{\mathtt{N}} \vdash B^{\mathtt{N}}}{{!}\Gamma^{\mathtt{N}} \vdash {!}A^{\mathtt{N}} \multimap B^{\mathtt{N}}}\,⅋$$

$$\dfrac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\,\to_e \quad \rightsquigarrow \quad \dfrac{{!}\Gamma^{\mathtt{N}} \vdash {!}A^{\mathtt{N}} \multimap B^{\mathtt{N}} \qquad \dfrac{\dfrac{{!}\Delta^{\mathtt{N}} \vdash A^{\mathtt{N}}}{{!}\Delta^{\mathtt{N}} \vdash {!}A^{\mathtt{N}}}\,! \qquad \dfrac{}{B^{\mathtt{N}} \vdash B^{\mathtt{N}}}\,ax}{{!}\Delta^{\mathtt{N}}, {!}A^{\mathtt{N}} \multimap B^{\mathtt{N}} \vdash B^{\mathtt{N}}}\,\otimes}{{!}\Gamma^{\mathtt{N}}, {!}\Delta^{\mathtt{N}} \vdash B^{\mathtt{N}}}\,cut$$

The translation $(\cdot)^{\mathtt{N}}$ puts a ! in front of every formula on the left-hand side of $\vdash$
$\rightsquigarrow$ the translation of the structural rules is obvious.

Girard's first translation: $(\cdot)^{\mathtt{N}}$

$$\begin{aligned} X^{\mathtt{N}} &= X \\ (A \to B)^{\mathtt{N}} &= {!A^{\mathtt{N}}} \multimap B^{\mathtt{N}} \\ (\Gamma \vdash A)^{\mathtt{N}} &= {!\Gamma^{\mathtt{N}}} \vdash A^{\mathtt{N}} \end{aligned}$$

(natural deduction for IL) (sequent calculus for ILL)

$$\dfrac{}{x : A \vdash x : A}\,ax \quad \rightsquigarrow \quad \dfrac{\dfrac{}{A^{\mathtt{N}} \vdash A^{\mathtt{N}}}\,ax}{!A^{\mathtt{N}} \vdash A^{\mathtt{N}}}\,der$$

$$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x\, M : A \to B}\,\to_i \quad \rightsquigarrow \quad \dfrac{!\Gamma^{\mathtt{N}}, !A^{\mathtt{N}} \vdash B^{\mathtt{N}}}{!\Gamma^{\mathtt{N}} \vdash !A^{\mathtt{N}} \multimap B^{\mathtt{N}}}\,\parr$$

$$\dfrac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\,\to_e \quad \rightsquigarrow \quad \dfrac{!\Gamma^{\mathtt{N}} \vdash !A^{\mathtt{N}} \multimap B^{\mathtt{N}} \qquad \dfrac{\dfrac{!\Delta^{\mathtt{N}} \vdash A^{\mathtt{N}}}{!\Delta^{\mathtt{N}} \vdash !A^{\mathtt{N}}}\,! \qquad \dfrac{}{B^{\mathtt{N}} \vdash B^{\mathtt{N}}}\,ax}{!\Delta^{\mathtt{N}}, !A^{\mathtt{N}} \multimap B^{\mathtt{N}} \vdash B^{\mathtt{N}}}\,\otimes}{!\Gamma^{\mathtt{N}}, !\Delta^{\mathtt{N}} \vdash B^{\mathtt{N}}}\,cut$$

The translation $(\cdot)^{\mathtt{N}}$ puts a ! in front of every formula on the left-hand side of $\vdash$
$\rightsquigarrow$ the translation of the structural rules is obvious.

Girard's second ("boring") translation: $(\cdot)^{\mathtt{v}}$

$$X^{\mathtt{v}} = X$$
$$(A \to B)^{\mathtt{v}} = !A^{\mathtt{v}} \multimap !B^{\mathtt{v}}$$
$$(\Gamma \vdash A)^{\mathtt{v}} = !\Gamma^{\mathtt{v}} \vdash !A^{\mathtt{v}}$$

(natural deduction for IL) (sequent calculus for ILL)

$$\frac{}{x : A \vdash x : A}\,ax \quad\rightsquigarrow\quad \frac{}{!A^{\mathtt{v}} \vdash !A^{\mathtt{v}}}\,ax$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x\, M : A \to B}\,\to_i \quad\rightsquigarrow\quad \frac{\dfrac{!\Gamma^{\mathtt{v}}, !A^{\mathtt{v}} \vdash !B^{\mathtt{v}}}{!\Gamma^{\mathtt{v}} \vdash !A^{\mathtt{v}} \multimap !B^{\mathtt{v}}}\,\parr}{!\Gamma^{\mathtt{v}} \vdash !(!A^{\mathtt{v}} \multimap !B^{\mathtt{v}})}\,!$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\,\to_e \quad\rightsquigarrow\quad \frac{!\Gamma^{\mathtt{v}} \vdash !(!A^{\mathtt{v}} \multimap !B^{\mathtt{v}}) \qquad \dfrac{\dfrac{!\Delta^{\mathtt{v}} \vdash !A^{\mathtt{v}} \qquad \dfrac{}{!B^{\mathtt{v}} \vdash !B^{\mathtt{v}}}\,ax}{!\Delta^{\mathtt{v}}, !A^{\mathtt{v}} \multimap !B^{\mathtt{v}} \vdash !B^{\mathtt{v}}}\,\otimes}{!\Delta^{\mathtt{v}}, !(!A^{\mathtt{v}} \multimap !B^{\mathtt{v}}) \vdash !B^{\mathtt{v}}}\,der}{!\Gamma^{\mathtt{v}}, !\Delta^{\mathtt{v}} \vdash !B^{\mathtt{v}}}\,cut$$

The translation $(\cdot)^{\mathtt{v}}$ puts a ! in front of every formula on the left-hand side of $\vdash$
$\rightsquigarrow$ the translation of the structural rules is obvious.

Girard's second ("boring") translation: $(\cdot)^{\mathtt{v}}$

$$
\begin{aligned}
X^{\mathtt{v}} &= X \\
(A \to B)^{\mathtt{v}} &= \,!A^{\mathtt{v}} \multimap\, !B^{\mathtt{v}} \\
(\Gamma \vdash A)^{\mathtt{v}} &= \,!\Gamma^{\mathtt{v}} \vdash\, !A^{\mathtt{v}}
\end{aligned}
$$

(natural deduction for IL) (sequent calculus for ILL)

$$
\dfrac{}{x : A \vdash x : A}\,ax \quad\rightsquigarrow\quad \dfrac{}{!A^{\mathtt{v}} \vdash\, !A^{\mathtt{v}}}\,ax
$$

$$
\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x\, M : A \to B}\,\to_i \quad\rightsquigarrow\quad \dfrac{\dfrac{!\Gamma^{\mathtt{v}}, !A^{\mathtt{v}} \vdash\, !B^{\mathtt{v}}}{!\Gamma^{\mathtt{v}} \vdash\, !A^{\mathtt{v}} \multimap\, !B^{\mathtt{v}}}\,\invamp}{!\Gamma^{\mathtt{v}} \vdash\, !(!A^{\mathtt{v}} \multimap\, !B^{\mathtt{v}})}\,!
$$

$$
\dfrac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\,\to_e \quad\rightsquigarrow\quad \dfrac{!\Gamma^{\mathtt{v}} \vdash\, !(!A^{\mathtt{v}} \multimap\, !B^{\mathtt{v}}) \qquad \dfrac{\dfrac{!\Delta^{\mathtt{v}} \vdash\, !A^{\mathtt{v}} \qquad \dfrac{}{!B^{\mathtt{v}} \vdash\, !B^{\mathtt{v}}}\,ax}{!\Delta^{\mathtt{v}}, !A^{\mathtt{v}} \multimap\, !B^{\mathtt{v}} \vdash\, !B^{\mathtt{v}}}\,\otimes}{!\Delta^{\mathtt{v}}, !(!A^{\mathtt{v}} \multimap\, !B^{\mathtt{v}}) \vdash\, !B^{\mathtt{v}}}\,der}{!\Gamma^{\mathtt{v}}, !\Delta^{\mathtt{v}} \vdash\, !B^{\mathtt{v}}}\,cut
$$

The translation $(\cdot)^{\mathtt{v}}$ puts a ! in front of every formula on the left-hand side of $\vdash$
$\rightsquigarrow$ the translation of the structural rules is obvious.

An example: from IL (natural deduction)...

$$\pi \quad = \quad \dfrac{\dfrac{\dfrac{}{a:A \vdash a:A}\,ax}{a:A, c:C \vdash a:A}\,w}{a:A \vdash \lambda c\, a : C \to A}\,\to_i \qquad \dfrac{\dfrac{}{x:B \to C \vdash x:B \to C}\,ax \qquad \dfrac{}{b:B \vdash b:B}\,ax}{b:B, x:B \to C \vdash xb:C}\,\to_e$$

$$\overline{\qquad a:A, b:B, x:B \to C \vdash (\lambda c\, a)(xb):A \qquad}\;\to_e$$

$$\downarrow_{\mathsf{cut}}$$

$$\mathsf{nf}(\pi) \quad = \quad \dfrac{\dfrac{\dfrac{}{a:A \vdash a:A}\,ax}{a:A, b:B, \vdash a:A}\,w}{a:A, b:B, x:B \to C \vdash a:A}\,w$$

An example: from IL (natural deduction) . . .

$$\pi \quad = \quad \dfrac{\dfrac{\dfrac{}{a{:}A \vdash a{:}A}\,ax}{a{:}A, c{:}C \vdash a{:}A}\,w}{a{:}A \vdash \lambda c\, a{:}C \to A}\,\to_i \qquad \dfrac{\dfrac{}{x{:}B \to C \vdash x{:}B \to C}\,ax \qquad \dfrac{}{b{:}B \vdash b{:}B}\,ax}{b{:}B, x{:}B \to C \vdash xb{:}C}\,\to_e$$

$$\dfrac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{a{:}A, b{:}B, x{:}B \to C \vdash (\lambda c\, a)(xb){:}A}\,\to_e$$

$$\downarrow_{\mathbf{cut}}$$

$$\mathrm{nf}(\pi) \quad = \quad \dfrac{\dfrac{\dfrac{}{a{:}A \vdash a{:}A}\,ax}{a{:}A, b{:}B, \vdash a{:}A}\,w}{a{:}A, b{:}B, x{:}B \to C \vdash a{:}A}\,w$$

$$(\lambda c\, a)(xb) \to_\beta a$$

An example: from IL (natural deduction) . . .

$$
\begin{tikzcd}
\text{int. logic (IL):} \arrow[d, "Curry-Howard"] & \pi \arrow[rr, "*"', "\mathsf{cut}"] \arrow[d, "\underline{(\cdot)}"] & & \mathrm{nf}(\pi) \arrow[d, "\underline{(\cdot)}"] \\
\lambda\text{-calculus:} & \underline{\pi} \arrow[r, "*", "\beta"'] & \mathrm{nf}(\underline{\pi}) = & \underline{\mathrm{nf}(\pi)}
\end{tikzcd}
$$

An example: ...to ILL via $(\cdot)^{\mathtt{N}}$

$$
\pi^{\mathtt{N}} =
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$A \vdash A$}
\RightLabel{der}
\UnaryInfC{$!A \vdash A$}
\RightLabel{w}
\UnaryInfC{$!A, !C \vdash A$}
\RightLabel{$⅋$}
\UnaryInfC{$!A \vdash !C \multimap A$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!B \multimap C \vdash !B \multimap C$}
\RightLabel{der}
\UnaryInfC{$!(!B \multimap C) \vdash !B \multimap C$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$B \vdash B$}
\RightLabel{der}
\UnaryInfC{$!B \vdash B$}
\RightLabel{$!$}
\UnaryInfC{$!B \vdash !B$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$C \vdash C$}
\RightLabel{$\otimes$}
\BinaryInfC{$!B, !B \multimap C \vdash C$}
\RightLabel{cut}
\BinaryInfC{$!B, !(!B \multimap C) \vdash C$}
\RightLabel{$!$}
\UnaryInfC{$!B, !(!B \multimap C) \vdash !C$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$A \vdash A$}
\RightLabel{$\otimes$}
\BinaryInfC{$!C \multimap A, !B, !(!B \multimap C) \vdash A$}
\RightLabel{cut}
\BinaryInfC{$a : !A, b : !B, x : !(!B \multimap C) \vdash (\lambda c\, a)(xb) : A$}
\end{prooftree}
$$

$$\mathbf{cut}\!\downarrow_+$$

$$
\mathrm{nf}(\pi^{\mathtt{N}}) =
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$a : A \vdash a : A$}
\RightLabel{der}
\UnaryInfC{$a : !A \vdash a : A$}
\RightLabel{w}
\UnaryInfC{$a : !A, b : !B, \vdash a : A$}
\RightLabel{w}
\UnaryInfC{$a : !A, b : !B, x : !(!B \multimap C) \vdash a : A$}
\end{prooftree}
= (\mathrm{nf}(\pi))^{\mathtt{N}}
$$

An example: ...to ILL via $(\cdot)^{\mathtt{N}}$

$$
\begin{array}{ccc}
\text{intuit. logic (IL):} & \qquad & \pi \xrightarrow[\text{cut}]{*} \mathrm{nf}(\pi) \\
\downarrow (\cdot)^{\mathtt{N}} & & \downarrow (\cdot)^{\mathtt{N}} \qquad\qquad\qquad \downarrow (\cdot)^{\mathtt{N}} \\
\text{intuit. LL (ILL):} & & \pi^{\mathtt{N}} \xrightarrow[\text{cut}]{*} \mathrm{nf}(\pi^{\mathtt{N}}) = (\mathrm{nf}(\pi))^{\mathtt{N}}
\end{array}
$$

An example: ... to ILL via $(\cdot)^{\mathtt{N}}$

$$\pi^{\mathtt{N}} = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\mathit{ax}}{!A \vdash A}\,\mathit{der}}{!A, !C \vdash A}\,\mathit{w}}{!A \vdash !C \multimap A}\,⅋ \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!B \multimap C \vdash !B \multimap C}\,\mathit{ax}}{!(!B \multimap C) \vdash !B \multimap C}\,\mathit{der} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B}\,\mathit{ax}}{!B \vdash B}\,\mathit{der}}{!B \vdash !B}\,! \qquad \dfrac{}{C \vdash C}\,\mathit{ax}}{!B, !B \multimap C \vdash C}\,\otimes}{!B, !(!B \multimap C) \vdash C}\,\mathit{cut}}{!B, !(!B \multimap C) \vdash !C}\,! \qquad \dfrac{}{A \vdash A}\,\mathit{ax}}{!C \multimap A, !B, !(!B \multimap C) \vdash A}\,\otimes}{{\color{blue}a}:!A, {\color{blue}b}:!B, {\color{blue}x}:!(!B \multimap C) \vdash {\color{blue}(\lambda c\, a)(xb)}:A}\,\mathit{cut}$$

$$_{\mathbf{cut}}\!\downarrow_{+}$$

$$\mathrm{nf}(\pi^{\mathtt{N}}) \;=\; \dfrac{\dfrac{\dfrac{\dfrac{}{{\color{blue}a}:A \vdash {\color{blue}a}:A}\,\mathit{ax}}{{\color{blue}a}:!A \vdash {\color{blue}a}:A}\,\mathit{der}}{{\color{blue}a}:!A, {\color{blue}b}:!B, \vdash {\color{blue}a}:A}\,\mathit{w}}{{\color{blue}a}:!A, {\color{blue}b}:!B, {\color{blue}x}:!(!B \multimap C) \vdash {\color{blue}a}:A}\,\mathit{w} \;=\; (\mathrm{nf}(\pi))^{\mathtt{N}}$$

$${\color{blue}(\lambda c\, a)(xb)} \to_{\beta} {\color{blue}a}$$

An example: . . . to ILL via $(\cdot)^{\mathtt{v}}$

$$
\pi^{\mathtt{v}} = \quad
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!A \vdash !A$}
\RightLabel{w}
\UnaryInfC{$!A, !C \vdash !A$}
\RightLabel{$⅋$}
\UnaryInfC{$!A \vdash !C \multimap !A$}
\RightLabel{$!$}
\UnaryInfC{$!A \vdash !(!C \multimap !A)$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!(!B \multimap !C) \vdash !(!B \multimap !C)$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!B \vdash !B$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!C \vdash !C$}
\RightLabel{$\otimes$}
\BinaryInfC{$!B, !B \multimap !C \vdash !C$}
\RightLabel{der}
\UnaryInfC{$!B, !(!B \multimap !C) \vdash C$}
\RightLabel{cut}
\BinaryInfC{$!B, !(!B \multimap !C) \vdash !C$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$A \vdash A$}
\RightLabel{$\otimes$}
\BinaryInfC{$!C \multimap A, !B, !(!B \to C) \vdash A$}
\RightLabel{der}
\UnaryInfC{$!(!C \multimap A), !B, !(!B \multimap !C) \vdash A$}
\RightLabel{cut}
\BinaryInfC{$a : !A, b : !B, x : !(!B \multimap !C) \vdash (\lambda c\, a)(xb) : A$}
\end{prooftree}
$$

$$
\mathbf{cut}\downarrow_{+}
$$

$$
\mathrm{nf}(\pi^{\mathtt{v}}) = \quad
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!B \vdash !B$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!A \vdash !A$}
\RightLabel{w}
\UnaryInfC{$!C, !A \vdash !A$}
\RightLabel{$\otimes$}
\BinaryInfC{$!A, !B, !B \multimap !C \vdash !A$}
\RightLabel{der}
\UnaryInfC{$a : !A, b : !B, x : !(!B \multimap !C) \vdash a[xb/c] : !A$}
\end{prooftree}
\quad \neq \quad
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$!A \vdash !A$}
\RightLabel{w}
\UnaryInfC{$!B, !A \vdash !A$}
\RightLabel{w}
\UnaryInfC{$a : !A, b : !B, x : !(!B \multimap !C) \vdash a : !A$}
\end{prooftree}
\quad = (\mathrm{nf}(\pi))^{\mathtt{v}}
$$

An example: ... to ILL via $(\cdot)^{\mathtt{v}}$

$$
\begin{tikzcd}
\text{intuit. logic (IL):} \arrow[d, "(\cdot)^{\mathtt{v}}"] & \pi \arrow[r, "*", "\mathsf{cut}"'] \arrow[d, "(\cdot)^{\mathtt{v}}"] & \mathsf{nf}(\pi) \arrow[d, "(\cdot)^{\mathtt{v}}"] \\
\text{intuit. LL (ILL):} & \pi^{\mathtt{v}} \arrow[r, "*", "\mathsf{cut}"'] & \mathsf{nf}(\pi^{\mathtt{v}}) \neq (\mathsf{nf}(\pi))^{\mathtt{v}}
\end{tikzcd}
$$

An example: ...to ILL via $(\cdot)^{\mathtt{v}}$

$$
\pi^{\mathtt{v}} = \quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{!A \vdash !A}\,ax
}{!A, !C \vdash !A}\,w
}{!A \vdash !C \multimap !A}\,⅋
}{!A \vdash !(!C \multimap !A)}\,!
\qquad
\dfrac{
\dfrac{
\dfrac{}{!(!B \multimap !C) \vdash !(!B \multimap !C)}\,ax
\quad
\dfrac{
\dfrac{
\dfrac{}{!B \vdash !B}\,ax \quad \dfrac{}{!C \vdash !C}\,ax
}{!B, !B \multimap !C \vdash !C}\,\otimes
}{!B, !(!B \multimap !C) \vdash C}\,der
}{!B, !(!B \multimap !C) \vdash !C}\,cut
\quad
\dfrac{}{A \vdash A}\,ax
}{
\dfrac{!C \multimap A, !B, !(!B \to C) \vdash A}{!(!C \multimap A), !B, !(!B \multimap !C) \vdash A}\,der
}\,\otimes
}{\color{blue}{a}:!A, \color{blue}{b}:!B, \color{blue}{x}:!(!B \multimap !C) \vdash \color{blue}{(\lambda c\, a)(xb)}:A}\,cut
$$

$$\mathbf{cut}\downarrow_{+}$$

$$
\mathrm{nf}(\pi^{\mathtt{v}}) = \quad
\dfrac{
\dfrac{
\dfrac{}{!B \vdash !B}\,ax
\quad
\dfrac{\dfrac{}{!A \vdash !A}\,ax}{!C, !A \vdash !A}\,w
}{!A, !B, !B \multimap !C \vdash !A}\,\otimes
}{\color{blue}{a}:!A, \color{blue}{b}:!B, \color{blue}{x}:!(!B \multimap !C) \vdash \color{blue}{a[xb/c]}:!A}\,der
\quad \color{blue}{\neq} \quad
\dfrac{
\dfrac{\dfrac{}{!A \vdash !A}\,ax}{!B, !A \vdash !A}\,w
}{\color{blue}{a}:!A, \color{blue}{b}:!B, \color{blue}{x}:!(!B \multimap !C) \vdash \color{blue}{a}:!A}\,w
\quad = (\mathrm{nf}(\pi))^{\mathtt{v}}
$$

$(\lambda c\, a)(xb) \to_{\beta_v} a[xb/c]$ (i.e. $\mathtt{let}\ c := xb\ \mathtt{in}\ a$) $\approx (\lambda c\, a)(xb)$ ... boring (according to Girard).

But $a[xb/c]$ is β_v-normal!

Call-by-name vs. call-by-value from a Linear Logic point of view

In the λ-calculus there are two evaluation mechanisms:

- *call-by-name* (CbN, β-reduction): no restriction in firing a β-redex;
- *call-by-value* (CbV, β_v-reduction): a β-redex $(\lambda x\, t)s$ can be fired only if s is a value.

ILL (and proof-nets) cut-elimination simulates $\begin{cases} \beta\text{-reduction via the translation } (\cdot)^{\mathtt{N}} \\ \beta_v\text{-reduction via the translation } (\cdot)^{\mathtt{V}} \end{cases}$

- via $(\cdot)^{\mathtt{N}}$ every argument is translated by a box
  $\rightsquigarrow$ every argument can be duplicated or discarded (CbN discipline);
- via $(\cdot)^{\mathtt{V}}$ every (and only) abstraction or variable is translated by a box
  $\rightsquigarrow$ only abstraction or variable can be duplicated or discarded (CbV discipline).

The two Girard's logical translations can explain the two different evaluation mechanisms inside the same setting, bringing them into the scope of the Curry-Howard isomorphism.

Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. John Maraist, Martin Odersky, David Turner, and Philip Wadler. MFPS, 1995.

Call-by-name vs. call-by-value from a Linear Logic point of view

In the λ-calculus there are two evaluation mechanisms:

- *call-by-name* (CbN, β-reduction): no restriction in firing a β-redex;
- *call-by-value* (CbV, β_v-reduction): a β-redex $(\lambda x\, t)s$ can be fired only if s is a value.

ILL (and proof-nets) cut-elimination simulates $\begin{cases} \beta\text{-reduction via the translation } (\cdot)^{\mathtt{N}} \\ \beta_v\text{-reduction via the translation } (\cdot)^{\mathtt{V}} \end{cases}$

- via $(\cdot)^{\mathtt{N}}$ every argument is translated by a box
  $\rightsquigarrow$ every argument can be duplicated or discarded (CbN discipline);
- via $(\cdot)^{\mathtt{V}}$ every (and only) abstraction or variable is translated by a box
  $\rightsquigarrow$ only abstraction or variable can be duplicated or discarded (CbV discipline).

The two Girard's logical translations can explain the two different evaluation mechanisms inside the same setting, bringing them into the scope of the Curry-Howard isomorphism.

Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. John Maraist, Martin Odersky, David Turner, and Philip Wadler. MFPS, 1995.

Call-by-name vs. call-by-value from a Linear Logic point of view

In the λ-calculus there are two evaluation mechanisms:

- *call-by-name* (CbN, β-reduction): no restriction in firing a β-redex;
- *call-by-value* (CbV, β_v-reduction): a β-redex $(\lambda x\, t)s$ can be fired only if s is a value.

ILL (and proof-nets) cut-elimination simulates $\begin{cases} \beta\text{-reduction via the translation } (\cdot)^{\mathtt{N}} \\ \beta_v\text{-reduction via the translation } (\cdot)^{\mathtt{V}} \end{cases}$

- via $(\cdot)^{\mathtt{N}}$ every argument is translated by a box
  $\rightsquigarrow$ every argument can be duplicated or discarded (CbN discipline);
- via $(\cdot)^{\mathtt{V}}$ every (and only) abstraction or variable is translated by a box
  $\rightsquigarrow$ only abstraction or variable can be duplicated or discarded (CbV discipline).

The two Girard's logical translations can explain the two different evaluation mechanisms inside the same setting, bringing them into the scope of the Curry-Howard isomorphism.

Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. John Maraist, Martin Odersky, David Turner, and Philip Wadler. MFPS, 1995.

Outline

What can LL say about the issues in Plotkin's CbV?

The terms

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

are β_v-normal because the β-redex (but not β_v-redex) $(\lambda x.\dots)(xx)$ is stuck.
$\leadsto$ The β-redex prevents the two δ's from interacting!

But if we translate δ_1 and δ_3 into ILL proof-nets, the two δ's can interact.
$\leadsto$ The translations of δ_1 and δ_3 into ILL proof-nets are diverging!

ILL is suggesting a way to extend β_v-reduction in a CbV setting.

Question: How can we internalize ILL behavior into a calculus?

Answer: There are at least two solutions.

What can LL say about the issues in Plotkin's CbV?

The terms

$$\delta_1 \coloneqq (\lambda x.\delta)(xx)\delta \qquad \delta_3 \coloneqq \delta((\lambda x.\delta)(xx))$$

are β_v-normal because the β-redex (but not β_v-redex) $(\lambda x.\dots)(xx)$ is stuck.
$\rightsquigarrow$ The β-redex prevents the two δ's from interacting!

But if we translate δ_1 and δ_3 into ILL proof-nets, the two δ's can interact.
$\rightsquigarrow$ The translations of δ_1 and δ_3 into ILL proof-nets are diverging!

ILL is suggesting a way to extend β_v-reduction in a CbV setting.

Question: How can we internalize ILL behavior into a calculus?

Answer: There are at least two solutions.

What can LL say about the issues in Plotkin's CbV?

The terms

$$\delta_1 := (\lambda x.\delta)(xx)\delta \qquad \delta_3 := \delta((\lambda x.\delta)(xx))$$

are β_v-normal because the β-redex (but not β_v-redex) $(\lambda x.\dots)(xx)$ is stuck.
$\rightsquigarrow$ The β-redex prevents the two δ's from interacting!

But if we translate δ_1 and δ_3 into ILL proof-nets, the two δ's can interact.
$\rightsquigarrow$ The translations of δ_1 and δ_3 into ILL proof-nets are diverging!

ILL is suggesting a way to extend β_v-reduction in a CbV setting.

Question: How can we internalize ILL behavior into a calculus?

Answer: There are at least two solutions.

Solution 1: Value Substitution Calculus [AccPao12]

Terms: $s, t, u ::= v \mid tu \mid t[u/x]$ Values: $v ::= x \mid \lambda x.t$

Substitution contexts: $L ::= [t_1/x_1] \dots [t_n/x_n]$

Reductions: $(\lambda x.t)Ls \to_m t[s/x]L$ $\qquad t[vL/x] \to_e t\{v/x\}L$

1. β_v-reduction can be simulated into VSC.

$$(\lambda x.t)v \to_m t[v/x] \to_e t\{v/x\}$$

2. VSC extends β_v-reduction:

$$\delta_1 = (\lambda x.\delta)(xx)\delta \to_m \delta[xx/x]\delta \to_m (zz)[\delta/z][xx/x] \to_e \delta\delta[xx/x] \to \cdots$$
$$\delta_3 = \delta((\lambda x.\delta)(xx)) \to_m \delta(\delta[xx/x]) \to_m (zz)[\delta[xx/x]/z] \to_e \delta\delta[xx/x] \to \cdots$$

In VSC, δ_1 and δ_3 are divergent as $\delta\delta$!

Solution 1: Value Substitution Calculus [AccPao12]

Terms: $s, t, u ::= v \mid tu \mid t[u/x]$ Values: $v ::= x \mid \lambda x.t$

Substitution contexts: $L ::= [t_1/x_1] \dots [t_n/x_n]$

Reductions: $(\lambda x.t)Ls \to_m t[s/x]L$ $t[vL/x] \to_e t\{v/x\}L$

1. β_v-reduction can be simulated into VSC.

$$(\lambda x.t)v \to_m t[v/x] \to_e t\{v/x\}$$

2. VSC extends β_v-reduction:

$$\delta_1 = (\lambda x.\delta)(xx)\delta \to_m \delta[xx/x]\delta \to_m (zz)[\delta/z][xx/x] \to_e \delta\delta[xx/x] \to \cdots$$
$$\delta_3 = \delta((\lambda x.\delta)(xx)) \to_m \delta(\delta[xx/x]) \to_m (zz)[\delta[xx/x]/z] \to_e \delta\delta[xx/x] \to \cdots$$

In VSC, δ_1 and δ_3 are divergent as $\delta\delta$!

Solution 2: Shuffling Calculus [CarrGue14]

Terms: $s, t, u ::= v \mid tu$ Values: $v ::= x \mid \lambda x.t$

β_v-reduction: $(\lambda x.t)v \to_{\beta_v} t\{v/x\}$

Shuffling reductions: $(\lambda x.t)su \to_{\sigma_1} (\lambda x.tu)s$ $\quad v((\lambda x.t)s) \to_{\sigma_3} (\lambda x.vt)s$

1. The shuffling calculus extends β_v-reduction:

$$\delta_1 = (\lambda x.\delta)(xx)\delta \to_{\sigma_1} (\lambda x.\delta\delta)(xx) \to_{\beta_v} (\lambda x.\delta\delta)(xx) \to_{\beta_v} \cdots$$
$$\delta_3 = \delta((\lambda x.\delta)(xx)) \to_{\sigma_3} (\lambda x.\delta\delta)(xx) \to_{\beta_v} (\lambda x.\delta\delta)(xx) \to_{\beta_v} \cdots$$

In the shuffling calculus, δ_1 and δ_3 are divergent as $\delta\delta$!

Solution 2: Shuffling Calculus [CarrGue14]

Terms: $s, t, u ::= v \mid tu$ Values: $v ::= x \mid \lambda x.t$

β_v-reduction: $(\lambda x.t)v \to_{\beta_v} t\{v/x\}$

Shuffling reductions: $(\lambda x.t)su \to_{\sigma_1} (\lambda x.tu)s$ $\qquad v((\lambda x.t)s) \to_{\sigma_3} (\lambda x.vt)s$

1. The shuffling calculus extends β_v-reduction:

$$\delta_1 = (\lambda x.\delta)(xx)\delta \to_{\sigma_1} (\lambda x.\delta\delta)(xx) \to_{\beta_v} (\lambda x.\delta\delta)(xx) \to_{\beta_v} \cdots$$
$$\delta_3 = \delta((\lambda x.\delta)(xx)) \to_{\sigma_3} (\lambda x.\delta\delta)(xx) \to_{\beta_v} (\lambda x.\delta\delta)(xx) \to_{\beta_v} \cdots$$

In the shuffling calculus, δ_1 and δ_3 are divergent as $\delta\delta$!

VSC vs. Shuffling: the importance of being (linearly) logical

Both VSC and Shuffling (Shuf) calculi are inspired by ILL proof-nets.
It turns out that they are "essentially the same" (termination equivalence)

Theorem (termination equivalence, [AccGue16])

Let t be a term: t is VSC-normalizing iff t is Shuf-normalizing.

Not *ad hoc*: these settings are termination equivalent to other extensions of Plotkin's one

- fireball calculus (Paolini & Ronchi Della Rocca, 1999; Grégoire & Leroy, 2002);
- CbV $\bar{\lambda}\mu\tilde{\mu}$-calculus (Curien & Herbelin, 2000);
- ...

(Introduced with different motivations: implementative, semantic, proof-theoretic, etc.)

Just different syntactic incarnations of the "same" CbV calculus (extending Plotkin's one).

VSC vs. Shuffling: the importance of being (linearly) logical

Both VSC and Shuffling (Shuf) calculi are inspired by ILL proof-nets.
It turns out that they are "essentially the same" (termination equivalence)

Theorem (termination equivalence, [AccGue16])

Let t be a term: t is VSC-normalizing iff t is Shuf-normalizing.

Not *ad hoc*: these settings are termination equivalent to other extensions of Plotkin's one

- fireball calculus (Paolini & Ronchi Della Rocca, 1999; Grégoire & Leroy, 2002);
- CbV $\bar{\lambda}\mu\tilde{\mu}$-calculus (Curien & Herbelin, 2000);
- ...

(Introduced with different motivations: implementative, semantic, proof-theoretic, etc.)

Just different syntactic incarnations of the "same" CbV calculus (extending Plotkin's one).

Restoring what was wrong in Plotkin's CbV!

In these CbV extensions we restore the good properties missing in Plotkin's CbV calculus.

1. Contextual equivalence in VSC and Shuf is the same as in Plotkin's calculus, but now δ_1 and δ_3 are CbV-divergent as $\delta\delta$.

2. Solvability in VSC and Shuf is the same as in Plotkin's calculus, but now we have an internal operational characterization of CbV solvability

Theorem [AccPao12, CarrGue14]

1. t is CbV-solvable iff t is normalizing for weak CbV-reduction.
2. Every CbV-normalizing term is CbV-solvable, but the converse fails (e.g. Y_v).

3. Denotational semantics in the VSC and Shuf is the same as in Plotkin's calculus, but now we also have completeness

Theorem (Correctness and Completeness [CarrGue12])

$[\![t]\!]_{\vec{x}} \neq \emptyset$ iff t is normalizing for "weak" CbV-reduction (not reducing under λ's).

Restoring what was wrong in Plotkin's CbV!

In these CbV extensions we restore the good properties missing in Plotkin's CbV calculus.

1. Contextual equivalence in VSC and Shuf is the same as in Plotkin's calculus, but now δ_1 and δ_3 are CbV-divergent as $\delta\delta$.

2. Solvability in VSC and Shuf is the same as in Plotkin's calculus, but now we have an internal operational characterization of CbV solvability

Theorem [AccPao12, CarrGue14]

1. t is CbV-solvable iff t is normalizing for weak CbV-reduction.
2. Every CbV-normalizing term is CbV-solvable, but the converse fails (e.g. Y_v).

3. Denotational semantics in the VSC and Shuf is the same as in Plotkin's calculus, but now we also have completeness

Theorem (Correctness and Completeness [CarrGue12])

$[\![t]\!]_{\vec{x}} \neq \emptyset$ iff t is normalizing for "weak" CbV-reduction (not reducing under λ's).

Restoring what was wrong in Plotkin's CbV!

In these CbV extensions we restore the good properties missing in Plotkin's CbV calculus.

1. Contextual equivalence in VSC and Shuf is the same as in Plotkin's calculus, but now δ_1 and δ_3 are CbV-divergent as $\delta\delta$.

2. Solvability in VSC and Shuf is the same as in Plotkin's calculus, but now we have an internal operational characterization of CbV solvability

Theorem [AccPao12, CarrGue14]

1. t is CbV-solvable iff t is normalizing for weak CbV-reduction.
2. Every CbV-normalizing term is CbV-solvable, but the converse fails (e.g. Y_v).

3. Denotational semantics in the VSC and Shuf is the same as in Plotkin's calculus, but now we also have completeness

Theorem (Correctness and Completeness [CarrGue12])

$[\![t]\!]_{\vec{x}} \neq \emptyset$ iff t is normalizing for "weak" CbV-reduction (not reducing under λ's).

Outline

Summing up

1. Plotkin's CbV λ-calculus can be extended by taking inspiration from LL.
2. The extensions are "conservative": they do not change CbV semantic notions.
3. Many issues in Plotkin's calculus are solved in these extended CbV settings.
4. We have all the ingredients to develop a theory for CbV as elegant as for CbN.

Summing up

1. Plotkin's CbV λ-calculus can be extended by taking inspiration from LL.
2. The extensions are "conservative": they do not change CbV semantic notions.
3. Many issues in Plotkin's calculus are solved in these extended CbV settings.
4. We have all the ingredients to develop a theory for CbV as elegant as for CbN.

Summing up

1. Plotkin's CbV λ-calculus can be extended by taking inspiration from LL.
2. The extensions are "conservative": they do not change CbV semantic notions.
3. Many issues in Plotkin's calculus are solved in these extended CbV settings.
4. We have all the ingredients to develop a theory for CbV as elegant as for CbN.

Summing up

1. Plotkin's CbV λ-calculus can be extended by taking inspiration from LL.
2. The extensions are "conservative": they do not change CbV semantic notions.
3. Many issues in Plotkin's calculus are solved in these extended CbV settings.
4. We have all the ingredients to develop a theory for CbV as elegant as for CbN.

Open Question 0: Categorical Semantics for CbV

Question: $CbN : CCC = CbV : X$. What is X?

Partial answer: [Ehrh12] shows how to build a model for CbV from a model of LL.

Open Question 1: A more general framework!

The existence of two separate paradigms (CbN and CbV λ-calculi) is troubling:

- it makes each language appear arbitrary (a unified language is more canonical);
- each time we create a new style of semantics (e.g. operational semantics, continuations, Scott semantics, game semantics, etc.) we always need to do it twice.

Question: Is there a general calculus containing both CbN and CbV?

In this setting we compare CbN and CbV λ-calculi

- in the same rewriting system, and
- with the same denotational semantics,
- obtaining CbN and CbV as fragments of this setting via translations.

Answer: [EhrGue16], [GueMan18], [BKVV20], [FagGue20], . . .

Open Question 1: A more general framework!

The existence of two separate paradigms (CbN and CbV λ-calculi) is troubling:

- it makes each language appear arbitrary (a unified language is more canonical);
- each time we create a new style of semantics (e.g. operational semantics, continuations, Scott semantics, game semantics, etc.) we always need to do it twice.

Question: Is there a general calculus containing both CbN and CbV?

In this setting we compare CbN and CbV λ-calculi

- in the same rewriting system, and
- with the same denotational semantics,
- obtaining CbN and CbV as fragments of this setting via translations.

Answer: [EhrGue16], [GueMan18], [BKVV20], [FagGue20], ...

Open Question 2: Inhabitation

In the non-idempotent intersection type system for CbV, typability is undecidable.

Question: Is the inabitation problem decidable in CbV?

Given an typing context Γ and a multi type M, is there a term t such that

$$\Gamma \vdash t : M \quad \text{is derivable?}$$

Question bis: Same question, but in a more general framework subsuming CbV and CbN.

Answer: Yes, it is decidable an we can find all the inhabitants! [ArrKesGue23]

Open Question 2: Inhabitation

In the non-idempotent intersection type system for CbV, typability is undecidable.

Question: Is the inabitation problem decidable in CbV?

Given an typing context Γ and a multi type M, is there a term t such that

$$\Gamma \vdash t : M \quad \text{is derivable?}$$

Question bis: Same question, but in a more general framework subsuming CbV and CbN.

Answer: Yes, it is decidable an we can find all the inhabitants! [ArrKesGue23]

Open Question 2: Inhabitation

In the non-idempotent intersection type system for CbV, typability is undecidable.

Question: Is the inabitation problem decidable in CbV?

Given an typing context Γ and a multi type M, is there a term t such that

$$\Gamma \vdash t : M \quad \text{is derivable?}$$

Question bis: Same question, but in a more general framework subsuming CbV and CbN.

Answer: Yes, it is decidable an we can find all the inhabitants! [ArrKesGue23]

Open question 3: Call by Need

Question: What about Call-by-Need? Can we use LL to understand Call-by-Need?

Question bis: Is there a general framework subsuming CbV, CbN and CbNeed?

Idea: We should split the ! comonad into two:

- one for duplication;
- one for erasure.

Open question 3: Call by Need

Question: What about Call-by-Need? Can we use LL to understand Call-by-Need?

Question bis: Is there a general framework subsuming CbV, CbN and CbNeed?

Idea: We should split the ! comonad into two:

- one for duplication;
- one for erasure.

Open question 3: Call by Need

Question: What about Call-by-Need? Can we use LL to understand Call-by-Need?

Question bis: Is there a general framework subsuming CbV, CbN and CbNeed?

Idea: We should split the ! comonad into two:

- one for duplication;
- one for erasure.

Thank you!

Questions?