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

1. What is Call-by-Value?
2. What is Wrong with Plotkin's Call-by-Value?
3. A Linear Logic Perspective to Call-by-Value
4. Restoring Call-by-Value thanks to Linear Logic
5. Conclusions

Outline

1. What is Call-by-Value?
2. What is Wrong with Plotkin's Call-by-Value?
3. A Linear Logic Perspective to Call-by-Value
4. Restoring Call-by-Value thanks to Linear Logic
5. Conclusions

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

The -calculus is the model of computation underlying

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

Actually, there are many -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 -calculus (mainly).

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

The -calculus is the model of computation underlying

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

Actually, there are many -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 -calculus (mainly).

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

The -calculus is the model of computation underlying

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

Actually, there are many -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).

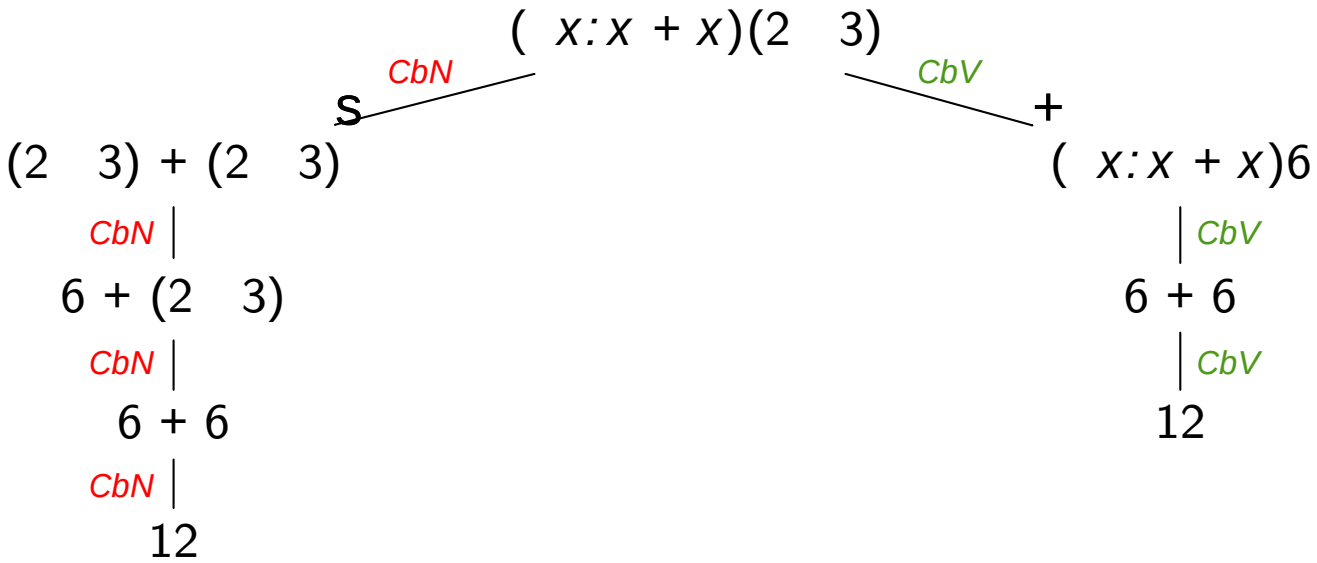
In this talk: pure untyped call-by-value -calculus (mainly).

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.

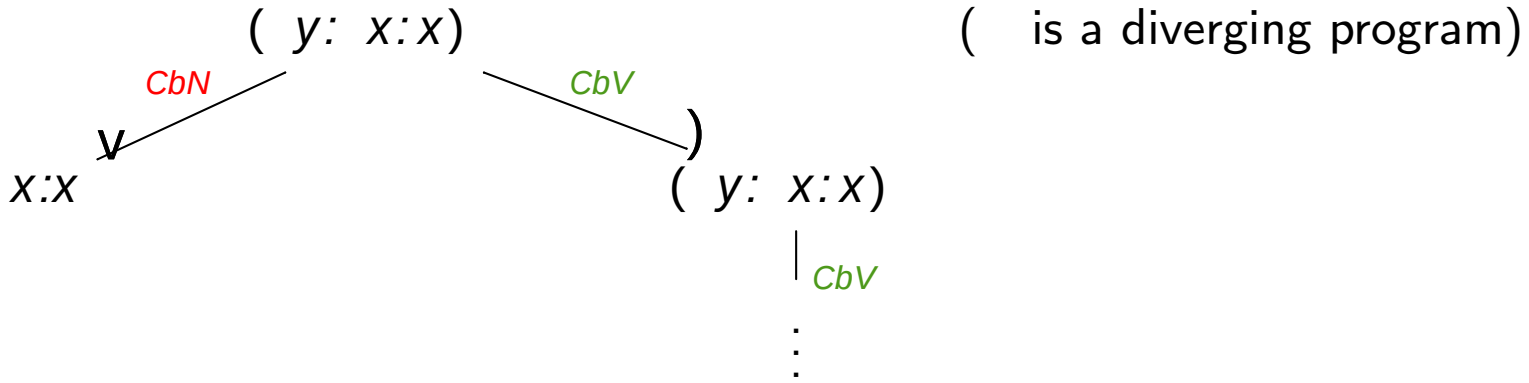
Call-by-Name vs. Call-by-Value (for dummies)

- 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.



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.



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

CbV reduction: $(\lambda x.t)v \to_v t\{v/x\}$ (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
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 $\to_v$-diverges

$$(\lambda y.I)(\Delta\Delta) \to_\beta I \qquad (\lambda y.I)(\Delta\Delta) \to_v (\lambda y.I)(\Delta\Delta) \to_v \cdots$$

2. $(\lambda x.\Omega)(xx)$ is $\to_v$-normal but β-divergent: $(\lambda x.\Omega)(xx) \to_\beta \Omega \to_\beta \Omega \to_\beta \cdots$

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

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

CbV reduction: $(\lambda x.t)v \to_v t\{v/x\}$ (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
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 $\to_v$-diverges

$$(\lambda y.I)(\Delta\Delta) \to_\beta I \qquad (\lambda y.I)(\Delta\Delta) \to_v (\lambda y.I)(\Delta\Delta) \to_v \cdots$$

2. $(\lambda x.\Omega)(xx)$ is $\to_v$-normal but β-divergent: $(\lambda x.\Omega)(xx) \to_\beta \Omega \to_\beta \Omega \to_\beta \cdots$

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

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

CbV reduction: $(\lambda x.t)v \to_v t\{v/x\}$ (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
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 $\to_v$-diverges

$$(\lambda y.I)(\Delta\Delta) \to_\beta I \qquad (\lambda y.I)(\Delta\Delta) \to_v (\lambda y.I)(\Delta\Delta) \to_v \cdots$$

2. $(\lambda x.\Omega)(xx)$ is $\to_v$-normal but β-divergent: $(\lambda x.\Omega)(xx) \to_\beta \Omega \to_\beta \Omega \to_\beta \cdots$

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

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

CbV reduction: $(\lambda x.t)v \to_v t\{v/x\}$ (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
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 $\to_v$-diverges

$$(\lambda y.I)(\Delta\Delta) \to_\beta I \qquad (\lambda y.I)(\Delta\Delta) \to_v (\lambda y.I)(\Delta\Delta) \to_v \cdots$$

2. $(\lambda x.\Omega)(xx)$ is $\to_v$-normal but β-divergent: $(\lambda x.\Omega)(xx) \to_\beta \Omega \to_\beta \Omega \to_\beta \cdots$

Outline

1. What is Call-by-Value?
2. What is Wrong with Plotkin's Call-by-Value?
3. A Linear Logic Perspective to Call-by-Value
4. Restoring Call-by-Value thanks to Linear Logic
5. Conclusions

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

Def. Terms $t; t^0$ are contextually equivalent if they are observably indistinguishable, i.e., for every context C, $C\langle t\rangle \to_v v$ (for some value v) iff $C\langle t^0\rangle \to_v v^0$ (for some value v^0)

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

$$\Omega_1 := (\lambda x.\Delta)(\Delta\Delta) \qquad \Omega_3 := \lambda z((\lambda x.\Delta)(\Delta\Delta))$$

Ω_1 and Ω_3 are $\to_v$-normal but contextually equivalent to Ω (which is $\to_v$-divergent)!

The "energy" (i.e. divergence) in Ω_1 and Ω_3 is only potential, in Ω is kinetic!

Why are Ω_1 and Ω_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 $t; t^0$ are contextually equivalent if they are observably indistinguishable, i.e., for every context C, $C\langle t\rangle \to_v v$ (for some value v) iff $C\langle t^0\rangle \to_v v^0$ (for some value v^0)

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

$$\Omega_1 := (\lambda x.\Delta)(\Delta) \qquad \Omega_3 := (\lambda x.\Delta)(\Delta)$$

Ω_1 and Ω_3 are $\to_v$-normal but contextually equivalent to Ω (which is $\to_v$-divergent)!

The "energy" (i.e. divergence) in Ω_1 and Ω_3 is only potential, in Ω is kinetic!

Why are Ω_1 and Ω_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 $t; t^0$ are contextually equivalent if they are observably indistinguishable, i.e., for every context C, $C\langle t\rangle \to_v v$ (for some value v) iff $C\langle t^0\rangle \to_v v^0$ (for some value v^0)

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

$$\Omega_1 := (\lambda x.\Delta)(\Delta\Delta) \qquad \Omega_3 := \lambda z((\lambda x.\Delta)(\Delta\Delta))$$

Ω_1 and Ω_3 are $\to_v$-normal but contextually equivalent to Ω (which is $\to_v$-divergent)!

The "energy" (i.e. divergence) in Ω_1 and Ω_3 is only potential, in Ω is kinetic!

Why are Ω_1 and Ω_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 H such that $H\langle t\rangle \to_{\beta} I$.

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.

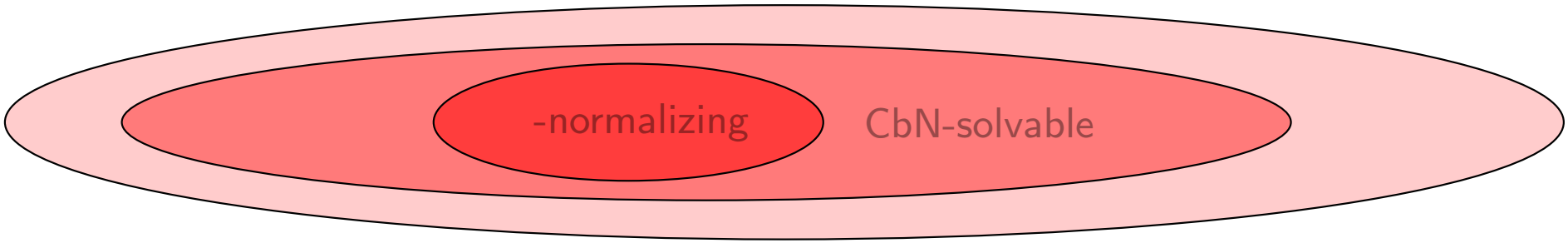
Moral: CbN-solvable terms are all and only the meaningful terms in CbN.
CbN-unsolvable terms are meaningless, and "heavily" divergent.

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 H such that $H\langle t\rangle \to_{\beta} I$.

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 β-theory that equates all CbN-unsolvable terms is consistent.
4. CbN-unsolvable terms represent undefined partial recursive functions.



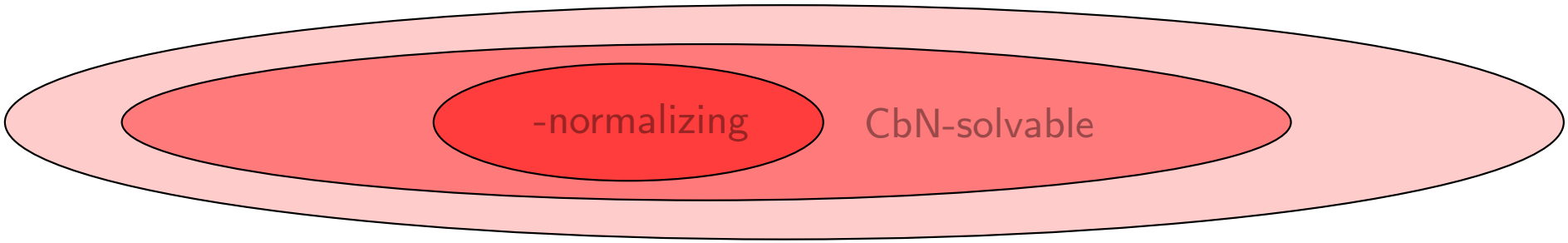
Moral: CbN-solvable terms are all and only the meaningful terms in CbN.
CbN-unsolvable terms are meaningless, and "heavily" divergent.

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 H such that $H\langle t\rangle \to_X^* I$.

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 λ-normalizing.
2. Every λ-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.
CbN-unsolvable terms are meaningless, and "heavily" divergent.

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 $_v$-normalizing ones.

$$_1 := (\ x.\ )(xx) \qquad _3 := ((\ x.\ )(xx))$$

$_1$ and $_3$ are $_v$-normal but CbV-unsolvable (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).

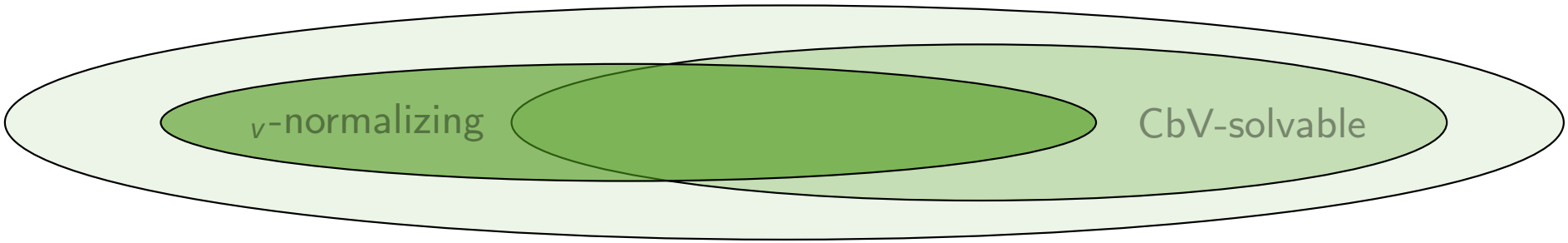
A second symptom that Plotkin's CbV is sick: Solvability (2 of 2)

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 $_v$-normalizing ones.

$$_1 := (\ x.\ )(xx) \qquad _3 := ((\ x.\ )(xx))$$

$_1$ and $_3$ are $_v$-normal but CbV-unsolvable (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).

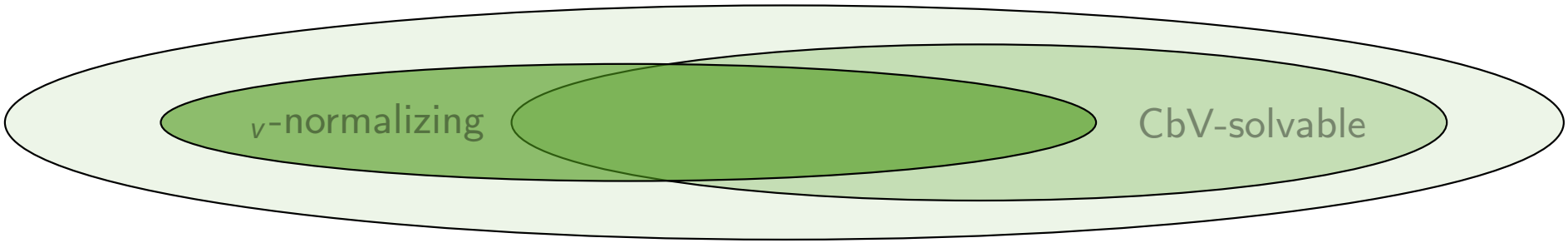
A second symptom that Plotkin's CbV is sick: Solvability (2 of 2)

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 $_v$-normalizing ones.

$$_1 := (\ x.\ )(xx) \qquad _3 := ((\ x.\ )(xx))$$

$_1$ and $_3$ are $_v$-normal but CbV-unsolvable (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 third symptom that Plotkin's CbV is sick: denotational semantics. (1 of 2)

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

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

Idea: $[L, L', L'] \simeq L \wedge L' \wedge L' \not\simeq L \wedge L'$ (commutative, associative, non-idempotent $\wedge$).

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

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. (1 of 2)

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

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

Idea: $[L, L', L'] \simeq L \wedge L' \wedge L' \not\simeq L \wedge L'$ (commutative, associative, non-idempotent $\wedge$).

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

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. (2 of 2)

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

$$\llbracket t \rrbracket_{\mathcal{R}} = \{(\Gamma; M) \mid \vdash t : M \text{ is derivable}\}$$

Theorem (Invariance, [Ehr12])

If $t \to_v u$ then $\llbracket t \rrbracket_{\mathcal{R}} = \llbracket u \rrbracket_{\mathcal{R}}$.

Theorem (Correctness, [Ehr12])

If $\llbracket t \rrbracket_{\mathcal{R}} \neq \emptyset$ then t is normalizing for "weak" $\to_v$-reduction (not reducing under λ's).

The converse (completeness) fail!

$$\llbracket \Delta_1 \rrbracket = \emptyset = \llbracket \Delta_3 \rrbracket \quad (\text{and } \llbracket \Omega \rrbracket = \emptyset \text{ too!})$$

but Δ_1 and Δ_3 are $\to_v$-normal, while Ω is $\to_v$-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

$$\llbracket t \rrbracket_{\mathcal{R}} = \{ (\Gamma ; M) \mid \Gamma \vdash t : M \text{ is derivable} \}$$

Theorem (Invariance, [Ehr12])

If $t \to_v u$ then $\llbracket t \rrbracket_{\mathcal{R}} = \llbracket u \rrbracket_{\mathcal{R}}$.

Theorem (Correctness, [Ehr12])

If $\llbracket t \rrbracket_{\mathcal{R}} \neq \emptyset$ then t is normalizing for "weak" v-reduction (not reducing under λ's).

The converse (completeness) fail!

$$\llbracket \Omega_1 \rrbracket = \emptyset = \llbracket \Omega_3 \rrbracket \quad \text{(and } \llbracket \Omega \rrbracket = \emptyset \text{ too!)}$$

but Ω_1 and Ω_3 are v-normal, while Ω is 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

$$\llbracket t \rrbracket_x = \{(\Gamma ; M) \mid \Gamma \vdash t : M \text{ is derivable}\}$$

Theorem (Invariance [Ehr12])

If $t \to_v u$ then $\llbracket t \rrbracket_x = \llbracket u \rrbracket_x$.

Theorem (Separation [Ehr12])

If $\llbracket t \rrbracket_x \neq \emptyset$ then t is normalizing for weak v-reduction (not reducing under λ's).

The converse (completeness) fail!

$$\llbracket \Omega_1 \rrbracket = \emptyset = \llbracket \Omega_3 \rrbracket \quad (\text{and } \llbracket \Omega \rrbracket = \emptyset \text{ too!})$$

but Ω_1 and Ω_3 are v-normal, while Ω is 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 -calculus there is a mismatch between syntax and semantics.

There are terms, such as

$$_1 := (\ \ x:\ )(xx) \qquad _3 := \ \ ((\ \ x:\ )(xx))$$

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

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

Somehow, in Plotkin's CbV -calculus, $_v$-reduction is not enough .

Can we extend $_v$ so that $_1$ and $_3$ are divergent?

But we want to keep a CbV discipline:

(x:I)( ) is $_v$-divergent (but -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 -calculus there is a mismatch between syntax and semantics.

There are terms, such as

$$_1 := (\ x:\ )(xx) \qquad _3 := ((\ x:\ )(xx))$$

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

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

Somehow, in Plotkin's CbV -calculus, $_v$-reduction is not enough .

Can we extend $_v$ so that $_1$ and $_3$ are divergent?

But we want to keep a CbV discipline:

$$(\ x{:}I)(\quad) \text{ is } _v\text{-divergent (but } \text{-normalizing)}$$

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 -calculus there is a mismatch between syntax and semantics.

There are terms, such as

$$_1 := (\ x:\ )(xx) \qquad _3 := ((\ x:\ )(xx))$$

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

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

Somehow, in Plotkin's CbV -calculus, $_v$-reduction is not enough .

Can we extend $_v$ so that $_1$ and $_3$ are divergent?

But we want to keep a CbV discipline:

$$(\ x{:}I)(\ \ ) \text{ is } _v\text{-divergent (but } \text{-normalizing)}$$

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 ght 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

1. What is Call-by-Value?
2. What is Wrong with Plotkin's Call-by-Value?
3. A Linear Logic Perspective to Call-by-Value
4. Restoring Call-by-Value thanks to Linear Logic
5. Conclusions

The role of linear logic with respect to λ-calculi

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

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

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:

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

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:

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

...

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:

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

...

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:

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

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	!	type
proof	!	program
cut-elimination	!	evaluation
coherence	!	termination
different encodings of intuitionistic arrow in LL	!	different evaluation mechanisms

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

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

The Curry-Howard-Girard correspondence

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

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

call-by-name evaluation via Girard's translation $(\ )^N$,

call-by-value evaluation via Girard's translation $(\ )^V$.

The Curry-Howard-Girard correspondence

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

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

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

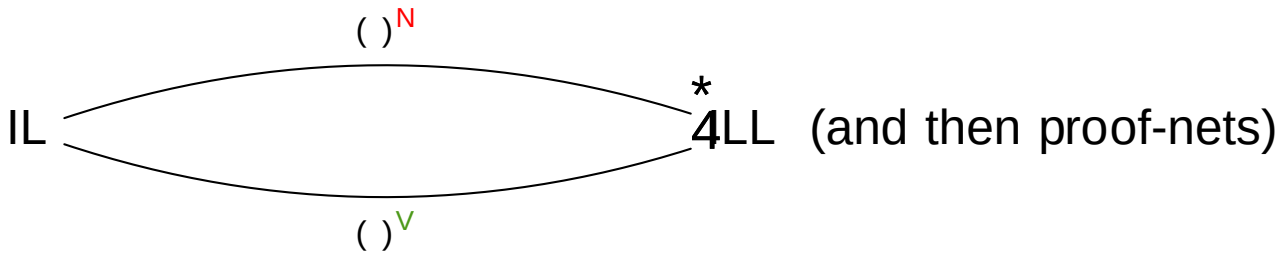
The two Girard's translations of IL into ILL (1987)

well-known translation $(\ )^N$

$$X^N = X$$
$$(A \Rightarrow B)^N = !A^N \multimap B^N$$
$$(\Gamma \vdash A)^N = !\Gamma^N \vdash A^N$$

boring translation $(\ )^V$

$$X^V = X$$
$$(A \Rightarrow B)^V = !A^V \multimap !B^V$$
$$(\Gamma \vdash A)^V = !\Gamma^V \vdash !A^V$$



(and then proof-nets)

simply typed = IL (via Curry-Howard)

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

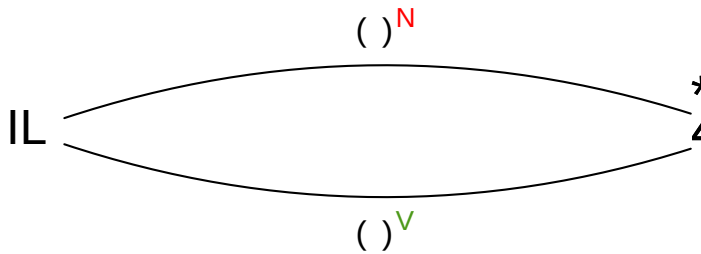
The two Girard's translations of IL into ILL (1987)

well-known translation $(\ )^N$

$$X^N = X$$
$$(A \Rightarrow B)^N = !A^N \multimap B^N$$
$$(\Gamma \vdash A)^N = !\Gamma^N \vdash A^N$$

boring translation $(\ )^V$

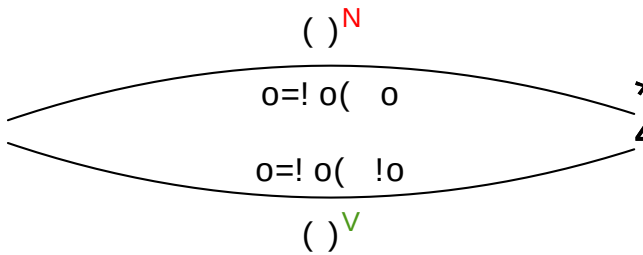
$$X^V = X$$
$$(A \Rightarrow B)^V = !A^V \multimap !B^V$$
$$(\Gamma \vdash A)^V = !\Gamma^V \vdash !A^V$$



(and then proof-nets)

simply typed = IL (via Curry-Howard)

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



(and then proof-nets)

Girard's first translation: $(\ )^N$

$$X^N = X$$
$$(A \multimap B)^N = \,!A^N \multimap B^N$$
$$(\Gamma \vdash A)^N = \,!\Gamma^N \vdash A^N$$

(natural deduction for IL)

$$\frac{}{x:A \vdash x:A}\ \text{ax}$$

$$\frac{;x:A \vdash M:B}{\Gamma \vdash x\,M:A \multimap B}\ \multimap_i$$

$$\frac{\Gamma \vdash M:A \multimap B \qquad \Gamma \vdash N:A}{;\Gamma \vdash MN:B}\ \multimap_e$$

(sequent calculus for ILL)

$$\frac{}{A^N \vdash A^N}\ \text{ax}$$
$$\frac{A^N \vdash A^N}{!A^N \vdash A^N}\ \text{der}$$

$$\frac{!\Gamma^N;!A^N \vdash B^N}{!\Gamma^N \vdash !A^N \multimap B^N}$$

$$\frac{\dfrac{!\Gamma^N \vdash A^N}{!\Gamma^N \vdash !A^N}\,! \qquad \dfrac{}{B^N \vdash B^N}\ \text{ax}}{!\Gamma^N \vdash !A^N \multimap B^N \qquad !\Gamma^N;!A^N \multimap B^N \vdash B^N}\ \text{cut}$$
$$!\Gamma^N;!\Gamma^N \vdash B^N$$

The translation $(\ )^N$ puts a $!$ in front of every formula on the left-hand side of ⋆ the translation of the structural rules is obvious.

Girard's first translation: $(\ )^N$

$$X^N = X$$
$$(A \multimap B)^N = \,!A^N \multimap B^N$$
$$(\Gamma \vdash A)^N = \,!\Gamma^N \vdash A^N$$

(natural deduction for IL)

$$\frac{}{x:A \vdash x:A}\ \text{ax}$$

$$\frac{;x:A \vdash M:B}{\Gamma \vdash x\,M:A \multimap B}\ \multimap_i$$

$$\frac{\Gamma \vdash M:A \multimap B \qquad \Delta \vdash N:A}{;\Gamma;\Delta \vdash MN:B}\ \multimap_e$$

(sequent calculus for ILL)

$$\frac{}{A^N \vdash A^N}\ \text{ax}$$
$$\frac{A^N \vdash A^N}{!A^N \vdash A^N}\ \text{der}$$

$$\frac{!\Gamma^N;!A^N \vdash B^N}{!\Gamma^N \vdash !A^N \multimap B^N}$$

$$\frac{\dfrac{!\Gamma^N \vdash A^N}{!\Gamma^N \vdash !A^N}\,! \qquad \dfrac{}{B^N \vdash B^N}\ \text{ax}}{!\Gamma^N \vdash !A^N \multimap B^N \qquad !\Delta^N;!A^N \multimap B^N \vdash B^N}\ \text{cut}$$
$$!\Gamma^N;!\Delta^N \vdash B^N$$

The translation $(\ )^N$ puts a $!$ in front of every formula on the left-hand side of ⋆ the translation of the structural rules is obvious.

Girard's second (boring) translation $(\ )^{V}$

$$X^{V} = X$$

$$(A \,!\, B)^{V} = \,!\, A^{V} \,(\, !B^{V}$$

$$(\ \ `\ A)^{V} = !\ \ ^{V} `\ !A^{V}$$

(natural deduction for IL)

$$\frac{}{x : A \ ` \ x : A}\ ax$$

$$\frac{;\, x : A \ ` \ M : B}{`\ \ x\, M : A \,!\, B}\ !_{i}$$

$$\frac{`\ M : A \,!\, B \qquad `\ N : A}{;\ \ `\ MN : B}\ !_{e}$$

(sequent calculus for ILL)

$$\frac{}{!A^{V} \ ` \ !A^{V}}\ ax$$

$$\frac{!\ ^{V}; !A^{V} \ ` \ !B^{V}}{!\ ^{V} `\ !A^{V} \,(\, !B^{V}}$$

$$\frac{!\ ^{V} `\ !(!\, A^{V} \,(\, !B^{V}) \qquad !\ ^{V}; !(!\, A^{V} \,(\, !B^{V}) \ ` \ !B^{V}}{!\ ^{V}; !\ \ ^{V} `\ !B^{V}}\ cut$$

$$\frac{!\ ^{V} `\ !A^{V} \qquad \dfrac{}{!B^{V} \ ` \ !B^{V}}\ ax}{!\ \ ^{V}; !A^{V} \,(\, !B^{V} \ ` \ !B^{V}}\ der$$

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

Girard's second (boring) translation $(\ )^{V}$

$$X^{V} = X$$

$$(A \,!\, B)^{V} = \,!\, A^{V} \,(\, !B^{V}$$

$$(\ \ `\ A)^{V} = !\ \ ^{V} `\ !A^{V}$$

(natural deduction for IL)

$$\frac{}{x : A \ ` \ x : A}\ ax$$

$$\frac{;\, x : A \ ` \ M : B}{`\ \ x\, M : A \,!\, B}\ !_{i}$$

$$\frac{`\ M : A \,!\, B \qquad `\ N : A}{;\ \ `\ MN : B}\ !_{e}$$

(sequent calculus for ILL)

$$\frac{}{!A^{V} \ ` \ !A^{V}}\ ax$$

$$\frac{!\ ^{V}; !A^{V} \ ` \ !B^{V}}{!\ ^{V} `\ !A^{V} \,(\, !B^{V}}\ !$$

$$\frac{!\ ^{V} `\ !(!\, A^{V} \,(\, !B^{V}) \qquad !\ ^{V}; !(!\, A^{V} \,(\, !B^{V}) \ ` \ !B^{V}}{!\ ^{V}; !\ \ ^{V} `\ !B^{V}}\ cut$$

$$\frac{!\ ^{V} `\ !A^{V} \qquad \dfrac{}{!B^{V} \ ` \ !B^{V}}\ ax}{!\ \ ^{V}; !A^{V} \,(\, !B^{V} \ ` \ !B^{V}}\ der$$

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

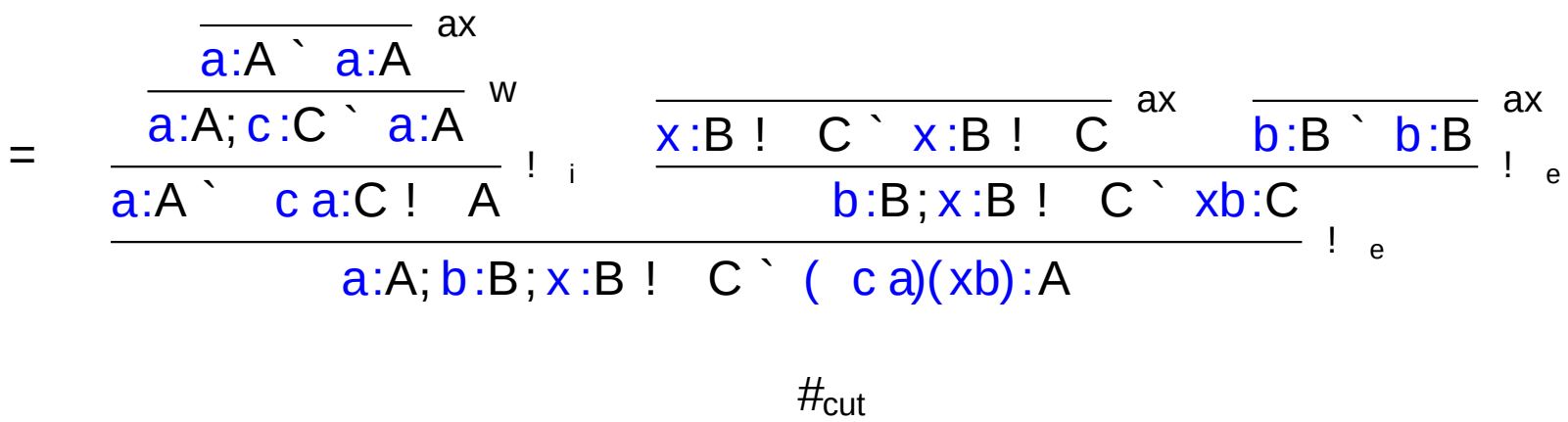
An example: from IL (natural deduction) . . .

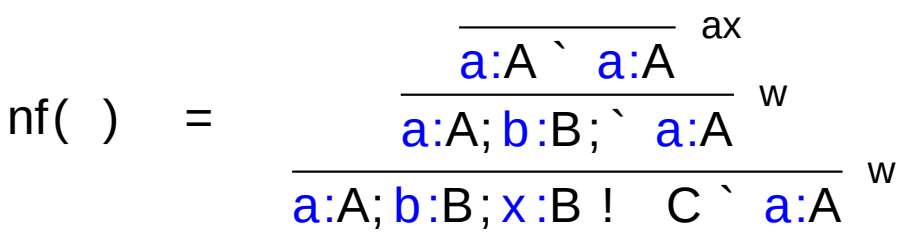
$$= \dfrac{\dfrac{\dfrac{}{a:A \vdash a:A}\,ax}{a:A, c:C \vdash a:A}\,w}{a:A \vdash c\, a:C \multimap A}\,\multimap_i \quad \dfrac{\dfrac{}{x:B \multimap C \vdash x:B \multimap C}\,ax \qquad \dfrac{}{b:B \vdash b:B}\,ax}{b:B, x:B \multimap C \vdash xb:C}\,\multimap_e}{a:A, b:B, x:B \multimap C \vdash (c\, a)(xb):A}\,\multimap_e$$

$$\#_{\mathrm{cut}}$$

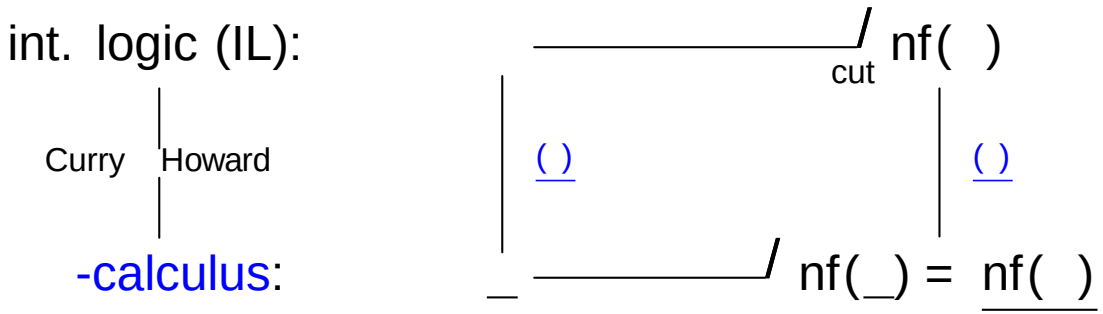
$$\mathrm{nf}(\ ) \quad = \quad \dfrac{\dfrac{\dfrac{}{a:A \vdash a:A}\,ax}{a:A, b:B, \vdash a:A}\,w}{a:A, b:B, x:B \multimap C \vdash a:A}\,w$$

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





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



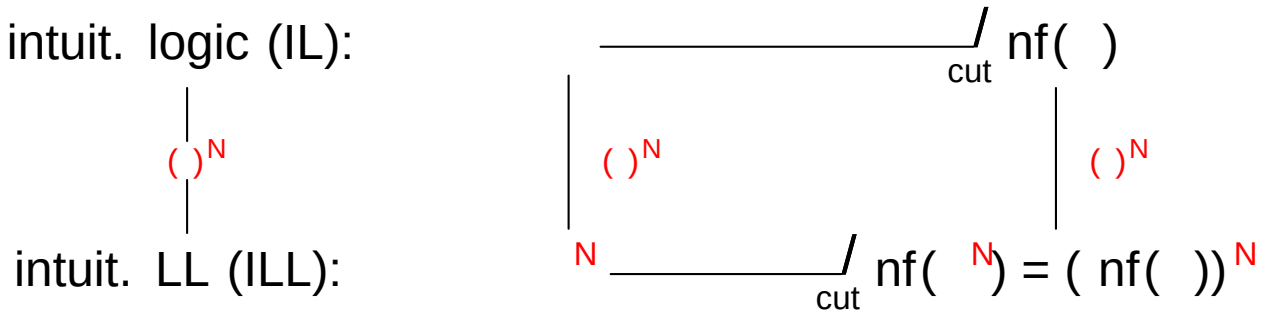
An example: . . . to ILL via $(\ )^N$

$$
{}^N = \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\mathrm{ax}}{!A \vdash A}\,\mathrm{der}}{!A;!C \vdash A}\,\mathrm{w}}{!A \vdash !C \multimap A} \qquad \dfrac{\dfrac{\dfrac{}{!B \multimap C \vdash !B \multimap C}\,\mathrm{ax}}{!(!B \multimap C) \vdash !B \multimap C}\,\mathrm{der} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B}\,\mathrm{ax}}{!B \vdash B}\,\mathrm{der}}{!B \vdash !B}\,! \qquad \dfrac{}{C \vdash C}\,\mathrm{ax}}{!B;!B \multimap C \vdash C}}{!B;!(!B \multimap C) \vdash C}\,\mathrm{cut}}{!B;!(!B \multimap C) \vdash !C}\,! \qquad \dfrac{}{A \vdash A}\,\mathrm{ax}}{!C \multimap A;!B;!(!B \multimap C) \vdash A}\,\mathrm{cut}}{a:!A;b:!B;x:!(!B \multimap C) \vdash (c\,a)(x\,b):A}
$$

$$
\mathrm{cut}\ \#_+
$$

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

An example: . . . to ILL via $(\ )^N$



An example: ...to ILL via $(\ )^N$

$$
{}^N = \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\mathrm{ax}}{!A \vdash A}\,\mathrm{der}}{!A;!C \vdash A}\,\mathrm{w}}{!A \vdash !C \multimap A} \qquad \dfrac{\dfrac{\dfrac{}{!B \multimap C \vdash !B \multimap C}\,\mathrm{ax}}{!(!B \multimap C) \vdash !B \multimap C}\,\mathrm{der} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B}\,\mathrm{ax}}{!B \vdash B}\,\mathrm{der}}{!B \vdash !B}\,! \qquad \dfrac{}{C \vdash C}\,\mathrm{ax}}{!B;!B \multimap C \vdash C}}{!B;!(!B \multimap C) \vdash C}\,\mathrm{cut} \qquad \dfrac{}{A \vdash A}\,\mathrm{ax}}{!B;!(!B \multimap C) \vdash !C}\,! \qquad \dfrac{}{A \vdash A}\,\mathrm{ax}}{!C \multimap A;!B;!(!B \multimap C) \vdash A}\,\mathrm{cut}
$$

$$
a{:}!A;\ b{:}!B;\ x{:}!(!B \multimap C) \vdash (c\,a)(x\,b){:}A
$$

$$
\xrightarrow[\mathrm{cut}]{}\ \#_+
$$

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

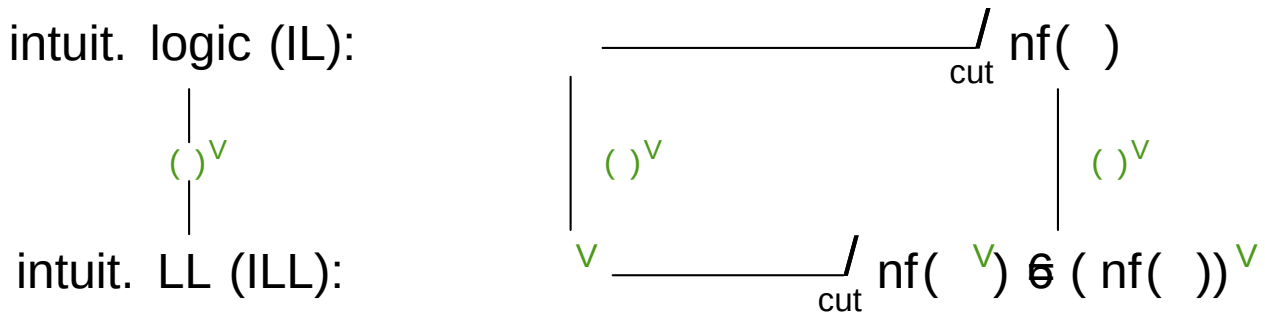
$$
(c\,a)(x\,b) \quad ! \quad a
$$

An example: . . . to ILL via $(\ )^{\vee}$

$$(\ )^{\vee} = \dfrac{\dfrac{\dfrac{\dfrac{}{!A \vdash !A}\,ax}{!A;!C \vdash !A}\,w}{!A \vdash !C \parr !A}\,\parr}{!A \vdash !(!C \parr !A)}\,! \qquad \dfrac{\dfrac{}{!(!B \parr !C) \vdash !(!B \parr !C)}\,ax \qquad \dfrac{\dfrac{\dfrac{}{!B \vdash !B}\,ax \quad \dfrac{}{!C \vdash !C}\,ax}{!B;!B \parr !C \vdash !C}\,der}{!B;!(!B \parr !C) \vdash C}\,der}{!B;!(!B \parr !C) \vdash !C}\,cut \qquad \dfrac{}{A \vdash A}\,ax}{!C \parr A;!B;!(!B \parr !C) \vdash A}\,der}{!(!C \parr A);!B;!(!B \parr !C) \vdash A}\,der}{a:!A;\ b:!B;\ x:!(!B \parr !C) \vdash (c\ a)(x\,b):A}\,cut$$

$$\mathrm{nf}(\ ^{\vee}) = \dfrac{\dfrac{\dfrac{}{!B \vdash !B}\,ax \qquad \dfrac{\dfrac{}{!A \vdash !A}\,ax}{!C;!A \vdash !A}\,w}{!A;!B;!B \parr !C \vdash !A}\,der}{a:!A;\ b:!B;\ x:!(!B \parr !C) \vdash a[x\,b=c]:!A}\ \underset{\mathrm{cut}}{\#_{+}}\ \dfrac{\dfrac{\dfrac{}{!A \vdash !A}\,ax}{!B;!A \vdash !A}\,w}{a:!A;\ b:!B;\ x:!(!B \parr !C) \vdash a:!A}\,w = (\mathrm{nf}(\ ))^{\vee}$$

An example: . . . to ILL via $(\ )^{\vee}$



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

$$
{}^{v} = \dfrac{\dfrac{\dfrac{}{!A \vdash !A}\,ax}{!A; !C \vdash !A}\,w}{\dfrac{!A \vdash !C \parr !A}{!A \vdash !(!C \parr !A)}\,!} \quad \dfrac{\dfrac{}{!(!B \parr !C) \vdash !(!B \parr !C)}\,ax \qquad \dfrac{\dfrac{\dfrac{}{!B \vdash !B}\,ax \quad \dfrac{}{!C \vdash !C}\,ax}{!B; !B \parr !C \vdash !C}\,der}{!B; !(!B \parr !C) \vdash !C}\,cut \qquad \dfrac{}{A \vdash A}\,ax}{\dfrac{!C \parr A; !B; !(!B \parr !C) \vdash A}{!(!C \parr A); !B; !(!B \parr !C) \vdash A}\,der}\,cut}{a{:}!A; b{:}!B; x{:}!(!B \parr !C) \vdash (c\ a)(x\,b) : A}
$$

$$
\mathrm{nf}(\ {}^{v}) = \dfrac{\dfrac{\dfrac{}{!B \vdash !B}\,ax \quad \dfrac{\dfrac{}{!A \vdash !A}\,ax}{!C; !A \vdash !A}\,w}{!A; !B; !B \parr !C \vdash !A}}{a{:}!A; b{:}!B; x{:}!(!B \parr !C) \vdash a[x\,b{=}c] : !A}\,der \quad \overset{\text{cut}\ \#_{+}}{\leadsto} \quad \dfrac{\dfrac{}{!A \vdash !A}\,ax}{\dfrac{!A \vdash !A}{!B; !A \vdash !A}\,w}\,w \quad = (\ \mathrm{nf}(\ ))^{v}
$$

$(c\ a)(x\,b)\ !\quad {}_{v}\ a[x\,b{=}c]$ (i.e. let $c := x\,b$ in a) $\quad (c\ a)(x\,b) :::$ boring (according to Girard).

But $a[x\,b{=}c]$ is ${}_{v}$-normal!

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

In the -calculus there are two evaluation mechanisms:

call-by-name (CbN, -reduction): no restriction in firing a -redex;

call-by-value (CbV, $_v$-reduction): a -redex (x t) s can be fired only if s is a value.

ILL (and proof-nets) cut-elimination simulates (-reduction via the translation $(\ )^N$ $_v$-reduction via the translation $(\ )^V$

via $(\ )^N$ every argument is translated by a box
every argument can be duplicated or discarded (CbN discipline);

via $(\ )^V$ every (and only) abstraction or variable is translated by a box
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 -calculus there are two evaluation mechanisms:

call-by-name (CbN, -reduction): no restriction in ring a -redex;

call-by-value (CbV, $_v$-reduction): a -redex (x t) s can be red only if s is a value.

ILL (and proof-nets) cut-elimination simulates (-reduction via the translation $( )^N$ / $_v$-reduction via the translation $( )^V$

via $( )^N$ every argument is translated by a box
every argument can be duplicated or discarded (CbN discipline);

via $( )^V$ every (and only) abstraction or variable is translated by a box
only abstraction or variable can be duplicated or discarded (CbV discipline).

The two Girard's logical translations can explain the two di erent 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 calculJohn 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 -calculus there are two evaluation mechanisms:

call-by-name (CbN, -reduction): no restriction in ring a -redex;

call-by-value (CbV, $_v$-reduction): a -redex (x t) s can be red only if s is a value.

ILL (and proof-nets) cut-elimination simulates (-reduction via the translation $(\ )^N$ / $_v$-reduction via the translation $(\ )^V$

via $(\ )^N$ every argument is translated by a box
every argument can be duplicated or discarded (CbN discipline);

via $(\ )^V$ every (and only) abstraction or variable is translated by a box
only abstraction or variable can be duplicated or discarded (CbV discipline).

The two Girard's logical translations can explain the two di erent 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

1 What is Call-by-Value?

2 What is Wrong with Plotkin's Call-by-Value?

3 A Linear Logic Perspective to Call-by-Value

4 Restoring Call-by-Value thanks to Linear Logic

5 Conclusions

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

The terms

$$_1 := (\ \ x:\ )(xx) \qquad _3 := \ \ ((\ \ x:\ )(xx))$$

are $_v$-normal because the -redex (but not $_v$-redex) $(\ \ x:::)(xx)$ is stuck.
The -redex prevents the two 's from interacting!

But if we translate $_1$ and $_3$ into ILL proof-nets, the two 's can interact.
The translations of $_1$ and $_3$ into ILL proof-nets are diverging!

ILL is suggesting a way to extend $_v$-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

$$_1 := (\ \ x:\ )(xx) \qquad _3 := \ \ ((\ \ x:\ )(xx))$$

are $_v$-normal because the -redex (but not $_v$-redex) $(\ \ x:::)(xx)$ is stuck.
The -redex prevents the two 's from interacting!

But if we translate $_1$ and $_3$ into ILL proof-nets, the two 's can interact.
The translations of $_1$ and $_3$ into ILL proof-nets are diverging!

ILL is suggesting a way to extend $_v$-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

$$_1 := (\ \ x:\ )(xx) \qquad _3 := \ \ ((\ \ x:\ )(xx))$$

are $_v$-normal because the -redex (but not $_v$-redex) $(\ \ x:::)(xx)$ is stuck.
The -redex prevents the two 's from interacting!

But if we translate $_1$ and $_3$ into ILL proof-nets, the two 's can interact.
The translations of $_1$ and $_3$ into ILL proof-nets are diverging!

ILL is suggesting a way to extend $_v$-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: $(\lambda x.t)Ls \to_m t[s/x]L$ $t[vL/x] \to_e t\{v/x\}L$

$\to_v$-reduction can be simulated into VSC.

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

VSC extends $\to_v$-reduction:

$\Delta_1 = (\lambda x.\ )(xx) \to_m [xx/x] \to_m (zz)[\ /z][xx/x] \to_e [xx/x] \to$

$\Delta_3 = ((\lambda x.\ )(xx)) \to_m (\ [xx/x]) \to_m (zz)[\ [xx/x]/z] \to_e [xx/x] \to$

In VSC, Δ_1 and Δ_3 are divergent as Ω.

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: $L ::= [t_1/x_1] \cdots [t_n/x_n]$

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

1. $\to_v$-reduction can be simulated into VSC.

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

2. VSC extends $\to_v$-reduction:

$$\Omega_1 = (\lambda x.\_)(xx) \to_m [xx/x] \to_m (\lambda z.z)[\_/z][xx/x] \to_e [xx/x] \to$$
$$\Omega_3 = ((\lambda x.\_)(xx)) \to_m (\_)[xx/x] \to_m (\lambda z.z)[\_[xx/x]/z] \to_e [xx/x] \to$$

In VSC, Ω_1 and Ω_3 are divergent as $\to$!

Solution 2: Shuffling Calculus [CarrGue14]

Terms: $s; t; u ::= v \mid tu$ Values: $v ::= x \mid \lambda x{:}t$

$_v$-reduction: $(\lambda x{:}t)v \to_v t\{v/x\}$

Shuffling reductions: $(\lambda x{:}t)su \to_1 (\lambda x{:}tu)s$ $v((\lambda x{:}t)s) \to_3 (\lambda x{:}vt)s$

The shuffling calculus extends $_v$-reduction:

$$\Omega_1 = (\lambda x{:}\_)(xx)\ (\lambda x{:}\_)(xx) \to_1 (\lambda x{:}\_)(xx)\ (\lambda x{:}\_)(xx) \to_v \cdots$$
$$\Omega_3 = ((\lambda x{:}\_)(xx))\ (\lambda x{:}\_)(xx) \to_3 (\lambda x{:}\_)(xx)\ (\lambda x{:}\_)(xx) \to_v \cdots$$

In the shuffling calculus, Ω_1 and Ω_3 are divergent as $\to$

Solution 2: Shuffling Calculus [CarrGue14]

Terms: $s; t; u ::= v \mid tu$ Values: $v ::= x \mid x:t$

$_v$-reduction: $(x:t)v \to_v t\{v/x\}$

Shuffling reductions: $(x:t)su \to_1 (x:tu)s$ $v((x:t)s) \to_3 (x:vt)s$

1. The shuffling calculus extends $_v$-reduction:

$$\Omega_1 = (x:)(xx) \to_1 (x:)(xx) \to_v (x:)(xx) \to_v$$
$$\Omega_3 = ((x:)(xx)) \to_3 (x:)(xx) \to_v (x:)(xx) \to_v$$

In the shuffling calculus, Ω_1 and Ω_3 are divergent as Ω!

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 ~-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 ~-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 $_1$ and $_3$ are CbV-divergent as .
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 i 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])

$\mathrm{J}t\mathrm{K}_x$; iff t is normalizing for "weak" CbV-reduction (not reducing under '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 $_1$ and $_3$ are CbV-divergent as .
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 i 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])

$Jt K_x$; iff t is normalizing for "weak" CbV-reduction (not reducing under '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 $_1$ and $_3$ are CbV-divergent as .
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 i 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])

$\llbracket t \rrbracket_x \neq 0$; iff t is normalizing for "weak" CbV-reduction (not reducing under 's).

Outline

1. What is Call-by-Value?
2. What is Wrong with Plotkin's Call-by-Value?
3. A Linear Logic Perspective to Call-by-Value
4. Restoring Call-by-Value thanks to Linear Logic
5. Conclusions

Summing up

1. Plotkin's CbV -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 -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 -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 -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: $\mathit{CbN} : \mathit{CCC} = \mathit{CbV} : X$. 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 -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 -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 -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 -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 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 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 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?