From Proof Terms to Programs

An operational and quantitative study of intuistionistic Curry-Howard languages

Loïc Peyrot

18 novembre 2022

IRIF, Université Paris Cité

Functional vs imperative languages

```
let rec factorial = function
| 1 => 1
| n where n > 1 =>
   n * factorial (n-1)
```

Functional program: “what”

```
int factorial (int n) {
  int factn = 1;
  while (n >= 1) {
    factn = factn * n;
    n--;
  }
  return factn;
}
```

Imperative program: “how”

Functional languages have a solid mathematical underlying theory.

Models of functional languages

At the core of functional programming are abstract models of computation. They:

- Assert fundamental properties of classes of languages.
- Influence implementations.
- Are oblivious to some implementation details.

Our main tools

The theory of λ-calculi and quantitative types.

What is a λ-calculus?

- An elementary syntax of terms (programs).

Example

In the original λ-calculus of Church, terms are built with three constructors:
variables x, abstractions $\lambda x.t$ and applications tu.

What is a λ-calculus?

- An elementary syntax of terms (programs).

Example

In the original λ-calculus of Church, terms are built with three constructors:
variables x, abstractions $\lambda x.t$ and applications tu.

- Reduction rules on terms, that represent computational progress.

Example

In Church's λ-calculus: a unique rule $(\lambda x.t)u \to_\beta t\{x/u\}$.

Semantics of programs

We give a meaning to programs. Two kinds of semantics are relevant for us:

Operational semantics is concerned with reductions on terms generated by the reduction rules.

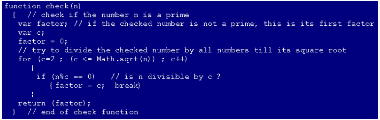
Denotational semantics is concerned with general properties on terms invariant by reduction.

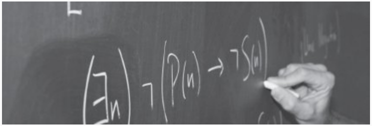
Different λ-calculi give rise to different semantics.

The Curry-Howard correspondence

Logical systems can be seen as models of computations.

Languages	Logic
Types	Propositions
Programs	Proofs
Evaluation	Cut-elimination





The intuitionistic Curry-Howard correspondence

Calculi	Intuitionistic proof systems
λ-calculus	Natural deduction (ND)
...	...
Atomic λ-calculus (Node replication)	Open deduction
Gundersen, Heijltjes & Parigot (2012)	Guglielmi, Gundersen, Parigot (2010)
λ-calculus with gen. applications	ND with gen. elimination
Joachimski & Matthes (2000)	von Plato (2001)

This work

We look into semantical properties of reduction with node replication or generalized applications, both:

Qualitative

a) Does a given term normalize?

b) Given two evaluation strategies, do they both normalize or diverge for a same term?

This work

We look into semantical properties of reduction with node replication or generalized applications, both:

Qualitative 

a) Does a given term normalize?

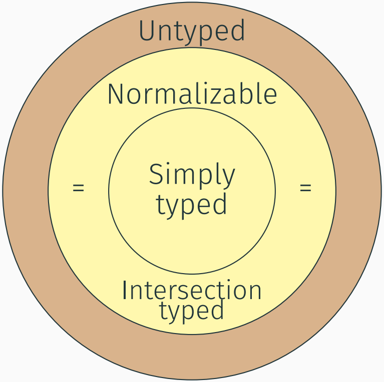
b) Given two evaluation strategies, do they both normalize or diverge for a same term?

Quantitative 

c) What is the reduction length of a given term to normal form?

d) Does an evaluation strategy normalize in less steps than another?

Intersection types capture normalization



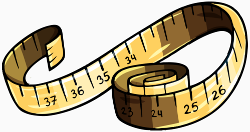
$\lambda x.xx$ is typable with intersection types.

$$\tau \to \sigma \wedge \tau \qquad \tau \to \sigma \quad \tau$$

$$\lambda x.xx$$

Idempotent and non-idempotent intersection types

Idempotent	Non-idempotent
Coppo & Dezani (80's)	Gardner, Kfoury (90's), de Carvalho (2007)
$\tau \wedge \tau = \tau$	$\tau \wedge \tau \neq \tau$
Qualitative analysis	Quantitative analysis



Idempotent:

$\tau \quad \tau \quad \tau$

$$\lambda f.\lambda x.fxx$$

Non-idempotent:

$\tau \wedge \tau \quad \tau \quad \tau$

$$\lambda f.\lambda x.fxx$$

Node Replication

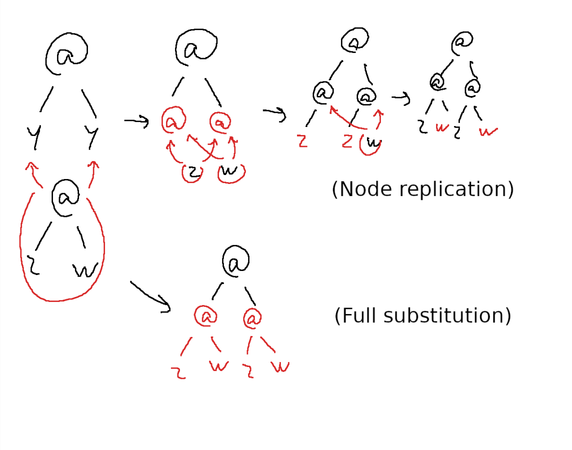
Different Curry-Howard notions of substitution

Substitution kind	Logical framework
Full Substitution	Natural Deduction
Linear Substitution	Linear Logic
Node Replication	Open Deduction

How is normalization affected by node replication (qualitatively and quantitatively)?

Node replication

Duplication of terms constructor by constructor. Enables optimizations by keeping more subterms shared.



Contributions

1. Define a simple calculus with node replication (called λR).
2. Define different evaluation strategies in the calculus.
3. Give a quantitative model for these strategies.
4. Prove observational equivalence between these strategies.

Firing substitution in the λR-calculus

(Terms) $\quad t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u]$

Definition (B-rule)

$(\lambda x.t)u \to_{\mathrm{B}} t[x/u]$

Some reductions are blocked by ES:

$(\lambda x.t)[y/v]u \not\to_{\mathrm{B}}$

Reduction at a distance

Reduction can be recovered by adding structural permutations.

$$(\lambda x.t)[y/v]u \to_\rho ((\lambda x.t)u)[y/v] \to_{\mathrm{B}} t[x/u][y/v]$$

Our approach: distance

B + needed permutations = dB:

$\mathsf{L}\langle \lambda x.t\rangle u \to_{\mathrm{dB}} \mathsf{L}\langle t[x/u]\rangle$, where $\mathsf{L} = \Diamond[x_1/u_1]\dots[x_n/u_n]$.

The λR-calculus

(Terms) $\quad t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u]$

The λR-calculus

(Terms) $\quad t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u]$

$$\mathsf{L}\langle \lambda x.t \rangle u \quad \mapsto_{\mathrm{dB}} \quad \mathsf{L}\langle t[x/u] \rangle$$

} Firing substitution

} Substitution

The λR-calculus

(Terms) $\quad t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u]$

$$\left.\begin{array}{rcl} \mathsf{L}\langle \lambda x.t\rangle u & \mapsto_{\mathrm{dB}} & \mathsf{L}\langle t[x/u]\rangle \end{array}\right\} \text{Firing substitution}$$

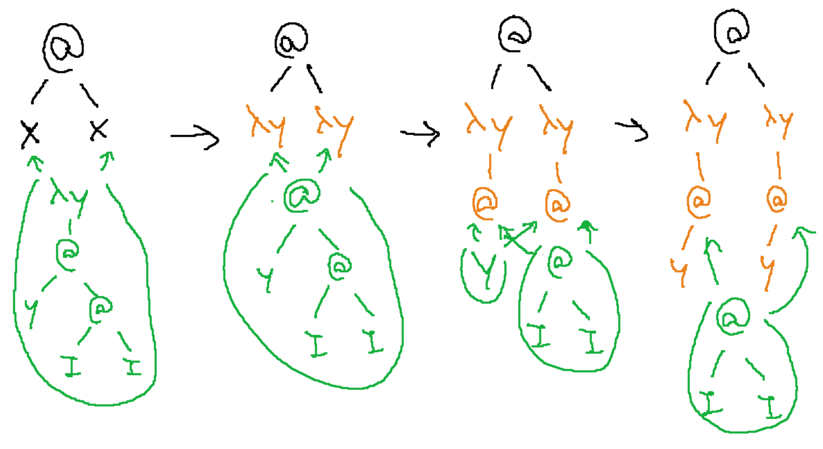
$$\left.\begin{array}{rcl} t[x/\mathsf{L}\langle y\rangle] & \mapsto_{\mathrm{var}} & \mathsf{L}\langle t\{x/y\}\rangle \\ t[x/\mathsf{L}\langle uv\rangle] & \mapsto_{\mathrm{app}} & \mathsf{L}\langle t\{x/yz\}[y/u][z/v]\rangle \\ t[x/\mathsf{L}\langle \lambda y.u\rangle] & \mapsto_{\mathrm{dist}} & \mathsf{L}\langle t[x//\lambda y.z[z/u]]\rangle \\ t[x//\lambda y.u] & \mapsto_{\mathrm{abs}} & \mathsf{L}\langle t\{x/\lambda y.p\}\rangle \end{array}\right\} \text{Substitution}$$

where $u \to_{\rho}^{*} \mathsf{L}\langle p\rangle$ and $y \notin \mathrm{fv}(\mathsf{L})$.

Full laziness

- An optimization of call-by-need (CbNeed).
- Can be implemented by node replication.
- Only duplicates the skeleton of abstractions.
- The skeleton is the path from the topmost abstraction λy to the occurrences of y.
- The complement of the skeleton stays shared.
- This avoids some duplication of computations.

Example of graphical fully lazy duplication



$$\lambda y.y(II) = \underbrace{\lambda y.yz}_{\text{skeleton}} + \underbrace{[z/II]}_{\text{sharing}}$$

Example of fully lazy duplication in λR

Full laziness can be implemented in λR.

$$
\begin{aligned}
(\lambda x.xx)(\lambda y.y(II)) &\rightarrow_{\mathrm{dB}} (xx)[x/\lambda y.y(II)] \\
&\rightarrow_{\mathrm{dist}} (xx)[x /\!/ \lambda y.z[z/y(II)]] \\
&\rightarrow_{\mathrm{app}} (xx)[x /\!/ \lambda y.(z_1 z_2)[z_1/y][z_2/II]] \\
&\rightarrow_{\mathrm{var}} (xx)[x /\!/ \lambda y.(y z_2)[z_2/II]] \\
&\rightarrow_{\mathrm{abs}} ((\lambda y.y z_2)(\lambda y.y z_2))[z_2/II]
\end{aligned}
$$

Comparison with the atomic λ-calculus

λR-calculus with Delia Kesner, Daniel Ventura (FoSSaCS 2021)	Atomic λ-calculus Gundersen, Heijltjes & Parigot (2012)
Non-linear variables	Linear variables
Distance	Independent permutations
Focuses on programming languages	Focuses on logical systems

Two strategies with node replication

We define (weak-head) call-by-name (CbN) and CBNeed strategies of λR.

- Our CbN simulates full substitution in the λ-calculus.
- Our CbNeed is fully lazy:
    - memoization,
    - need contexts, and
    - skeleton extraction.

Two different semantic for splitting

Big-step (one of the rules)

$$\frac{t \Downarrow^{\theta \cup \{x\}} \mathsf{L}\langle s \rangle}{\lambda x.t \Downarrow^{\theta} \mathsf{L}\langle \lambda x.s \rangle}$$

Small-step (one of the rules)

$t[x/\lambda z.u] \mapsto^{y}_{\mathrm{dist}} t[x//\lambda z.x'[x'/u]]$ where $y \in \mathrm{fv}(u)$

Theorem

The two semantics are equivalent, and give the correct splitting.

Small-steps skeleton extraction is more flexible

When considering skeletons of terms with ES, the big-steps semantics may cause inefficency.

Example

Let $\lambda x.t = \lambda x.(\lambda y.y[x'/x])z$.

- $t \Downarrow^{\{x\}} ((\lambda y.y)z')[z'/z]$, but:
- $w[w//\lambda x.t] \to^* (\lambda x.w)[w/(\lambda y.y)z]$ (in two steps)

The quantitative type system $\cap R$

Some of the typing rules:

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\textsc{ax}) \qquad\qquad \frac{}{\emptyset \vdash \lambda x.t : \mathsf{a}}\ (\textsc{ans})$$

$$\frac{\Gamma; x : [\tau_i]_{0 \leq i \leq n} \vdash t : \sigma \qquad \Delta_1 \vdash u : \tau_1 \qquad \ldots \qquad \Delta_n \vdash u : \tau_n}{\Gamma \uplus \Delta_1 \uplus \ldots \uplus \Delta_n \vdash t[x/u] : \sigma}\ (\textsc{es})$$

Characterization of normalization by typability

Contribution

call-by-name (NR)

fully lazy call-by-need (NR)

quantitative type system $\cap R$

Full laziness is an operational feature, not a semantical one.

NR = Node replication

FS = Full substitution

Characterization of normalization by typability

Contribution

call-by-name (NR)   fully lazy call-by-need (NR)

quantitative type system $\cap R$

call-by-name (FS)   call-by-need (FS)

NR = Node replication   FS = Full substitution

An upper bound for fully lazy reduction

- Usually, in intuitionistic calculi, the size of the non-idempotent type derivation decreases at each step.
- In λR, rules app and dist adds fresh variables that makes the size of the derivation grow.

$$t[x/u_1u_2] \rightarrow_{\mathrm{app}} t\{x/x_1x_2\}[x_1/u_1][x_2/u_2]$$

- We define a decreasing measure on type derivations, enabling a combinatorial proof of normalization.

The quantitative model: permutations vs distance

At every step of reduction, the measure on type derivations decreases.

With permutations: not every step consumes resources.

With distance: every step consumes resources.

Back to the questions (I)

Qualitative questions

a) Does a given term normalize?

b) Given two evaluation strategies, do they both normalize or diverge for a same term?

Answers:

a) If and only if it is typable in system $\cap R$.

b) CbN and CbNeed, with full substitution or node replication all normalize on the same terms.

Back to the questions (II)

Quantitative questions

c) What is the reduction length of a given term to normal form?

d) Does an evaluation strategy normalize in less steps than another?

Answers:

c) The measure gives an upper bound on the number of reduction steps.

d) Full laziness reduces the length of reduction w.r.t. full substitution.

Generalized Applications

Call-by-name and call-by-value generalized applications

Generalized applications (GA) are a Curry-Howard interpretation of natural deduction with generalized elimination rules.

	Original calculi	Distant variants
CbN	ΛJ (Joachimski & Matthes, 2000)	λJ_n (new)
CbV	ΛJ_v (Espírito Santo, 2020)	λJ_v (new)

CbN: call-by-name

CbV: call-by-value

The syntax of terms for generalized applications

$$\begin{array}{lrl}
\text{(Values)} & v & ::= x \mid \lambda x.t \\
\text{(Terms)} & t, u, r & ::= v \mid t(u, x.r)
\end{array}$$

$$\frac{}{\Gamma, x : A \vdash x : A}\ \text{AX} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \to_i$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A \qquad \Gamma, x : B \vdash r : C}{\Gamma \vdash t(u, x.r) : C}\ \to_e$$

All and only applications are shared

Shared?	Variables	Abstractions	Applications
ES	$r[x/y]$	$r[x/\lambda y.t]$	$r[x/tu]$
GA	no	no	$t(u, x.r)$

First intuition (not completely right)

$t(u, x.r) \approx$ let x = tu in $r \approx r[x/tu]$

Rewriting in the original CbN calculus with GA

A β-rule with meta-level substitutions

$(\lambda x.t)(u, y.r) \to_\beta r\{y/t\{x/u\}\}$

- Generalizes β-reduction in the λ-calculus.

A commutative conversion π

$t(u, x.r)(u', y.r') \to_\pi t(u, x.r(u', y.r'))$

- Moves the leftmost redex on top of the term.

Failure of CbN subject reduction for π

We give a quantitative type system $\cap J$ for CbN reduction of GA.

Subject reduction/expansion in a quantitative type system

- Weighted subject reduction: $\underbrace{t_1 \to t_2 \to \ldots \to t_n}_{\Gamma \vdash t:\tau}$
- Subject expansion: $\underbrace{t_1 \leftarrow \cdots \leftarrow t_n}_{\Gamma \vdash t:\tau}$

But with π: subject reduction in the quantitative system fails.

Question

e) Can we define a CbN calculus with generalized applications compatible with a quantitative model?

Joint work with Delia Kesner and José Espírito Santo, FoSSaCS 2022.

Permutations are necessary

We cannot remove π-permutations without changing normalization, because π-permutations are useful to unblock beta-reduction.

$$z(u_1, y_1.\lambda x.x)(u_2, y_2.y_2) \not\to_\beta$$

Permutations are necessary

We cannot remove π-permutations without changing normalization, because π-permutations are useful to unblock beta-reduction.

$$
\begin{aligned}
z(u_1, y_1.\lambda x.x)(u_2, y_2.y_2) &\to_\pi z(u_1, y_1.(\lambda x.x)(u_2, y_2.y_2)) \\
&\to_\beta z(u_1, y_1.u_2)
\end{aligned}
$$

A new CbN calculus

- We consider instead the permutation rule $\mathrm{p2}$:

$$t(u, y.\lambda x.r) \to_{\mathrm{p2}} \lambda x.t(u, y.r)$$

- We define a distant calculus λJ_n based on $\mathrm{p2}$ and using a single distant rule $\mathrm{d}\beta$.
- Unlike ΛJ, this calculus is compatible with the quantitative type system:
  - **a)** Typability characterizes strong normalization.
  - **c)** The size of type derivations gives an upper bound on the length of reduction and size of normal forms.

The qualitative semantics is preserved

Comparison of the semantics of the CbN calculi

b) Given two evaluation strategies, do they both normalize or diverge for a same term?
d) Does an evaluation strategy normalize in less steps than another?

Answers:

- **b)** Strong normalization of λJ_n and ΛJ correspond.
- **d)** The quantitivities are incomparable.

A different duplication behavior in the new CbN calculus

Definition (Distant contexts)

$\mathsf{D} ::= \Diamond \mid t_1(t_2, x.\mathsf{D})$

What makes λJ_n and ΛJ different? Compare:

$$(\lambda J_n) \qquad \mathsf{D}\langle \lambda x.t\rangle(u, y.r) \to_{\mathrm{d}\beta} r\{y/\mathsf{D}\langle t\{x/u\}\rangle\}$$

Duplication or erasure of D

$$(\Lambda J) \qquad \mathsf{D}\langle \lambda x.t\rangle(u, y.r) \to_{\pi}^{*}\to_{\beta} \mathsf{D}\langle r\{y/t\{x/u\}\}\rangle$$

Sharing of D

Towards a faithful translation to explicit substitutions

We want to relate strong normalization in GA and the λ-calculus (with explicit substitutions).

Reminder: initial (wrong) intuition

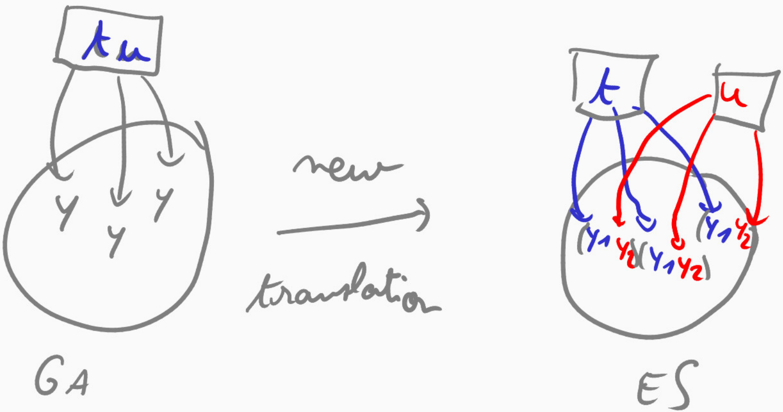
$t(u, x.r) \approx r[x/tu]$

But the semantics differs.

Example

Let $\delta = \lambda x.xx$ and $\delta_j = \lambda x.x(x, z.z)$.
The terms $\delta_j(\delta_j, x.\lambda y.y)$ and $(\lambda y.y)[x/\delta\delta]$ seem to correspond.
But in CbN, only the first one is strongly normalizing.

The new translation



Theorem

Translations preserve strong normalization both way.

Refined operational study of generalized applications

f) Does the operational semantics of generalized applications enable to capture semantical properties?

We look at:

- A perpetual strategy.
- A normalizing strategy.
- Solvability (FSCD 2022).

The Cbn and CbV original calculi

Call-by-name (Joachimski & Matthes):

$$\left.\begin{array}{rcl} (\lambda x.t)(u,y.r) & \to_\beta & r\{y/t\{x/u\}\} \\ t(u,x.r)(u',y.r') & \to_\pi & t(u,x.r(u',y.r')) \end{array}\right\}\Lambda J$$

Call-by-value (Espírito Santo):

$$\left.\begin{array}{rcl} (\lambda x.t)(u,y.r) & \to_{\beta\mathrm{v}} & r\{y\backslash\!\backslash t\{x\backslash\!\backslash u\}\} \\ t(u,x.r)(u',y.r') & \to_\pi & t(u,x.r(u',y.r')) \end{array}\right\}\Lambda J_v$$

Definition (CbV substitution)

$t\{x\backslash\!\backslash \mathsf{D}\langle v\rangle\} = \mathsf{D}\langle t\{x/v\}\rangle$

The new CbN and CbV distant calculi

$$\left.\mathtt{D}\langle\lambda x.t\rangle(u, y.r) \quad \to_{\mathrm{d}\beta} \quad r\{y/\mathtt{D}\langle t\{x/u\}\rangle\}\right\} \lambda J_n$$

CbN, based on $\mathrm{p}2$

$$\left.\mathtt{D}\langle\lambda x.t\rangle(u, y.r) \quad \to_{\mathrm{d}\beta_{\mathrm{v}}} \quad \mathtt{D}\langle r\{y\backslash\backslash t\{x\backslash\backslash u\}\}\rangle\right\} \lambda J_v$$

CbV, based on π

A benefit of using generalized applications

- In the CbN λ-calculus, the leftmost-outermost reduction is normalizing.
- Giving a normalizing strategy for a CbV calculus is in general much more difficult (Leberle, 2021).

In the framework of generalized applications:

- We give a simple normalizing strategy for CbV.
- This strategy reduces redexes in the leftmost-outermost order.

A normalizing leftmost-outermost CbV strategy (for ΛJ_v)

Normal forms

$\mathrm{NF} ::= x \mid \lambda x.\,\mathrm{NF} \mid x(\mathrm{NF}, y.\,\mathrm{NF})$

Base rules

$\beta\mathrm{v} \quad + \quad \pi$

Contextual rules

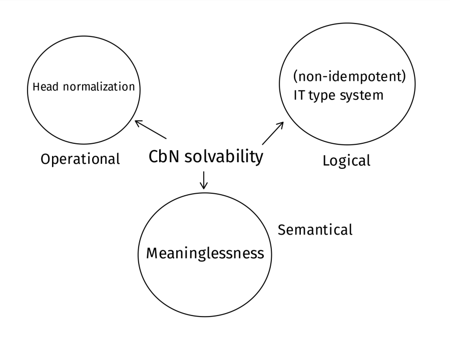
$$\frac{t \to t'}{\lambda x.t \to \lambda x.t'} \qquad \frac{u \to u'}{x(u, y.r) \to x(u', y.r)} \qquad \frac{r \to r'}{x(u, y.r) \to x(u, y.r')}$$

Taking the same normal forms and inductive rules, we can obtain a CbN normalizing strategy.

Call-by-name solvability for generalized applications

Contribution

Characterizations of CbN solvability in λJ_n and ΛJ.



Definition

t is CbN solvable:

$\exists$H, D such that

H$\langle t \rangle \rightarrow_{\lambda J_n}$ D$\langle \lambda x.x \rangle$.

Theorem

Translations to and from the λ-calculus preserve solvability.

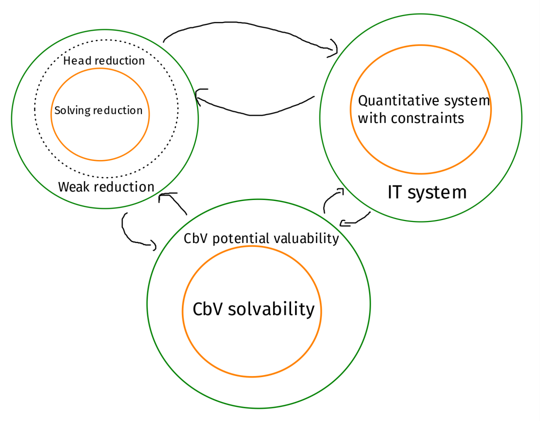
What about CbV solvability?

- Normalizable terms should should all be meaningful.
- But Plotkin's CbV calculus is defective:
- The term $(\lambda x.\delta)(yy)\delta$ (for instance) has no denotation but is in normal form.

CbV solvability for generalized applications

Contribution

Characterizations of CbV solvability in λJ_v and ΛJ_v.



Definition

t is CbV solvable:

$\exists$H such that

H$\langle t \rangle \to_{\lambda J_v} \lambda x.x$.

Alternative CbV operational characterizations

Operational characterizations of CbV solvability were already given for two other calculi.

λ_{vsub} (Accattoli & Paolini, 2012)

Uses explicit substitutions and distance:

$(\lambda x.\delta)(yy)\delta \to_{\mathrm{d}\beta\mathrm{v}} (\delta\delta)[x/yy] \circlearrowleft^{2}_{\mathrm{d}\beta\mathrm{v}}$

$\lambda^{\sigma}_{\mathsf{v}}$ (Carraro & Guerrieri, 2014)

Adds permutations to Plotkin's calculus:

$(\lambda x.\delta)(yy)\delta \to_{\sigma_1} (\lambda x.\delta\delta)(yy) \circlearrowleft_{\beta\mathrm{v}}$

Comparing CbV solvability in different frameworks

Theorem

Translations between GA and the λ-calculus (with explicit substitutions) preserve CbV solvability both ways.

Comparing CbV solvability in different frameworks

Theorem

Translations between GA and the λ-calculus (with explicit substitutions) preserve CbV solvability both ways.

We can compare the solving evaluation strategies:

	GA	$\lambda_{\mathsf{v}}^{\sigma}$	λ_{vsub}
Simple normal forms	Yes	No	Yes
Meta-level substitutions	Yes	Yes	No
Stuck reductions	No	$(\lambda x.x)(yy) \not\to$	$x[x/yy] \not\to$
Moggi's identity	Yes	No	No

Definition (Moggi's identity)

For any term u, $(\lambda x.x)u \to u$.

Conclusion

Contributions

- A new calculus for node replication.
- CbN and fully lazy CbNeed strategies based on node replication.
- Quantitative models for these strategies.
- CbN and CbV distant calculi with GA.
- Operational characterizations of solvability and weak normalization in GA.
- Quantitative models for CbN and CbV generalized applications.

Further works

Short term:

- Exact bounds with tight type systems.
- Abstract machines for full laziness.

Long term:

- Understand the correct notion of meaningless term in CbV equipped with a genericity lemma.
- Classical calculi to capture control operators in generalized applications and node replication.
- Fully abstract CbV models.

Thank you for your attention!