Types of Fireballs

The noble art of non-idempotent intersection types in Call-by-Value

Giulio Guerrieri[1] Beniamino Accattoli[2]

[1]DISI, Università di Bologna

[2]INRIA, Saclay

Crecogi/Elica/GDRI-LL
Paris, October 10, 2018

Outline

Outline

What is a good λ-calculus?

There are many variants of λ-calculus:

- Strong or weak (evaluation can fire under λ's or not);
- Term may be open or only closed;
- Evaluation strategies: call-by-name (CbN), call-by-value (CbV), call-by-need;
- With or without explicit substitutions, with or without linear substitutions;
- ...

What is a good λ-calculus? Some (non-exhaustive) criteria:

1. Rewriting;
2. Logic;
3. Implementation;
4. Cost model;
5. Denotations;
6. Equality.

Meta-principle: The more principles are connected, the better.

What is a good λ-calculus?

There are many variants of λ-calculus:

- Strong or weak (evaluation can fire under λ's or not);
- Term may be open or only closed;
- Evaluation strategies: call-by-name (CbN), call-by-value (CbV), call-by-need;
- With or without explicit substitutions, with or without linear substitutions;
- ...

What is a good λ-calculus? Some (non-exhaustive) criteria:

1. Rewriting;
2. Logic;
3. Implementation;
4. Cost model;
5. Denotations;
6. Equality.

Meta-principle: The more principles are connected, the better.

Open Call-by-Value

The setting we will study via non-idempotent intersection types: Open CbV, i.e.

- evaluation is weak (does not reduce under λ's),
- terms are possibly open.

$\rightsquigarrow$ intermediate setting between Strong CbV (which evaluates under λ's) and Closed CbV (terms are closed and evaluation is weak).

Motivation: Closed CbV is enough for modeling programming languages, not proof-assistants. Strong CbV is a very tricky setting.

In the literature, there are many equivalent presentations of Open CbV.
The fireball calculus λ_{fire} is one of them, with many good properties...

- elegant rewrite theory, clear logical understanding;
- simple implementation in abstract machines, with a reasonable cost model;

...many inventors with different motivations: Ronchi Della Rocca & Paolini (1999, 2004), Grégoire & Leroy (2002), Accattoli & Sacerdoti Coen (2014);

...and a big problem: there is no adequate denotational model for it!

Open Call-by-Value

The setting we will study via non-idempotent intersection types: Open CbV, i.e.

- evaluation is weak (does not reduce under λ's),
- terms are possibly open.

$\rightsquigarrow$ intermediate setting between Strong CbV (which evaluates under λ's) and Closed CbV (terms are closed and evaluation is weak).

Motivation: Closed CbV is enough for modeling programming languages, not proof-assistants. Strong CbV is a very tricky setting.

In the literature, there are many equivalent presentations of Open CbV.
The fireball calculus λ_{fire} is one of them, with many good properties...

- elegant rewrite theory, clear logical understanding;
- simple implementation in abstract machines, with a reasonable cost model;

...many inventors with different motivations: Ronchi Della Rocca & Paolini (1999, 2004), Grégoire & Leroy (2002), Accattoli & Sacerdoti Coen (2014);

...and a big problem: there is no adequate denotational model for it!

Non-idempotent intersection types

Non-idempotent intersections types (aka multi-types) have many features:

- qualitative: they characterize normalization for (some strategy of) λ-calculus;
- quantitative: they provide bounds on the execution time (i.e. the number of β-steps) to reach the normal form;
- denotational semantics: they can be seen as a syntactic presentation of *relational semantics*, a denotational model of λ-calculus;
- linear logic interpretation: they are deeply linked to linear logic.

De Carvalho's System R (2009) shows all these features for CbN λ-calculus.

Our goal: to show these features in a CbV setting.

Rmk (for LL friends only): The multi-type system used in the λ-calculus depends on the evaluation mechanism, according the two Girard's translations.

CbN	CbV ("boring")
$(A \to B)^n = !A^n \multimap B^n$	$(A \to B)^v = !A^v \multimap !B^v$

Non-idempotent intersection types

Non-idempotent intersections types (aka multi-types) have many features:

- qualitative: they characterize normalization for (some strategy of) λ-calculus;
- quantitative: they provide bounds on the execution time (i.e. the number of β-steps) to reach the normal form;
- denotational semantics: they can be seen as a syntactic presentation of *relational semantics*, a denotational model of λ-calculus;
- linear logic interpretation: they are deeply linked to linear logic.

De Carvalho's System R (2009) shows all these features for CbN λ-calculus.

Our goal: to show these features in a CbV setting.

Rmk (for LL friends only): The multi-type system used in the λ-calculus depends on the evaluation mechanism, according the two Girard's translations.

CbN	CbV ("boring")
$(A \to B)^n = !A^n \multimap B^n$	$(A \to B)^v = !A^v \multimap !B^v$

Non-idempotent intersection types

Non-idempotent intersections types (aka multi-types) have many features:

- qualitative: they characterize normalization for (some strategy of) λ-calculus;
- quantitative: they provide bounds on the execution time (i.e. the number of β-steps) to reach the normal form;
- denotational semantics: they can be seen as a syntactic presentation of *relational semantics*, a denotational model of λ-calculus;
- linear logic interpretation: they are deeply linked to linear logic.

De Carvalho's System R (2009) shows all these features for CbN λ-calculus.

Our goal: to show these features in a CbV setting.

Rmk (for LL friends only): The multi-type system used in the λ-calculus depends on the evaluation mechanism, according the two Girard's translations.

CbN	CbV ("boring")
$(A \to B)^n = {!A^n} \multimap B^n$	$(A \to B)^v = {!A^v} \multimap {!B^v}$

Outline

1. Introduction
2. The rise and the fall of the fireball calculus λ_{fire}
3. The split fireball calculus $\lambda_{\mathsf{fire}}^{\mathsf{split}}$ and multi types
4. Tight type derivations and exact bounds

Plotkin's CbV λ-calculus

$$\begin{array}{rrcl} \text{Terms} & t,u,s & ::= & x \mid \lambda x.t \mid tu \\ \text{Values} & v,v' & ::= & x \mid \lambda x.t \\ \text{Left evaluation contexts} & C & ::= & \langle\cdot\rangle \mid vC \mid Ct \end{array}$$

Rule at top level	Contextual closure
$(\lambda x.t)v \quad \mapsto_{\beta_v} \quad t\{v/x\}$	$C\langle t\rangle \to_{\beta_v} C\langle u\rangle \quad \text{if } t \mapsto_{\beta_v} u$

Plotkin's CbV is well-behaved when evaluation is weak and terms are only closed.

Consider the case of weak evaluation and possibly open terms (Open CbV).

$$t \coloneqq (\lambda z.\delta)(xx)\delta \qquad \text{where } \delta \coloneqq \lambda y.yy$$

t is β_v-normal but it "morally" shouldn't! It should behave like the diverging $\delta\delta$.

- In any denotational model for CbV, t has the same semantics as $\delta\delta$;
- for any sensitive notion of observational equivalence, t and $\delta\delta$ are equivalent.

The fireball calculus λ_{fire}

Restore the good properties of Plotkin's CbV for Open CbV.

$$\begin{array}{rrcl}
\text{Terms} & t,u,s,r & ::= & x \mid \lambda x.t \mid tu \\
\text{Values} & v,v',v'' & ::= & x \mid \lambda x.t \\
\text{Fireballs} & f,f',f'' & ::= & v \mid i \\
\text{Inert terms} & i,i',i'' & ::= & xf_1 \ldots f_n \qquad n > 0 \\
\text{Right evaluation contexts} & C & ::= & \langle\cdot\rangle \mid tC \mid Cf
\end{array}$$

Rule at top level	Contextual closure
$(\lambda x.t)v \mapsto_{\beta_v} t\{v/x\}$	$C\langle t\rangle \to_{\beta_v} C\langle u\rangle$ if $t \mapsto_{\beta_v} u$
$(\lambda x.t)i \mapsto_{\beta_i} t\{i/x\}$	$C\langle t\rangle \to_{\beta_i} C\langle u\rangle$ if $t \mapsto_{\beta_i} u$
Reduction	$\to_{\beta_f} := \to_{\beta_v} \cup \to_{\beta_i}$

Example: $(\lambda z.\delta)(xx)\delta \to_{\beta_i} \delta\delta \to_{\beta_v} \delta\delta \to_{\beta_v} \ldots$ (where $\delta := \lambda y.yy$)

Proposition (Distinctive properties of λ_{fire})

1. *Open harmony*: t is β_f-normal if and only if t is a fireball.
2. *Conservative open extension*: $t \to_{\beta_f} u$ if and only if $t \to_{\beta_v} u$, when t is closed.
3. *Evaluation and inert substitutions commute* $t \to_{\beta_f} u$ iff $t\{i/x\} \to_{\beta_f} u\{i/x\}$.

Non-idempotent intersection types for CbV (Ehrhard, 2012)

Multi types and linear types are defined by mutual induction:

$$\textit{linear types} \quad L, L' ::= P \multimap Q \qquad \textit{multi types} \quad P, Q ::= \overbrace{[L_1, \ldots, L_n]}^{\text{multiset}} \; (n \geq 0)$$

- there are no base types: their role is played by the empty multiset $[\,]$ $(n = 0)$;
- A multi type $[L_1, \ldots, L_n]$ has to be intended as a conjunction $L_1 \wedge \cdots \wedge L_n$;
- this conjunction $\wedge$ is commutative and associative, non-idempotent;
- $t : [L_1, L_2, L_2]$ intuitively means that t can be used once as data of type L_1 and twice as data of type L_2.

$$\frac{}{x : P \vdash x : P}\,\mathsf{ax} \qquad \frac{\Gamma \vdash t : [P \multimap Q] \qquad \Delta \vdash u : P}{\Gamma \uplus \Delta \vdash tu : Q}\,@$$

$$\frac{\Gamma_1, x : P_1 \vdash t : Q_1 \qquad \overset{n \in \mathbb{N}}{\ldots} \qquad \Gamma_n, x : P_n \vdash t : Q_n}{\Gamma_1 \uplus \cdots \uplus \Gamma_n \vdash \lambda x.t : [P_1 \multimap Q_1, \ldots, P_n \multimap Q_n]}\,\lambda$$

where $\Gamma \uplus \Delta$ is the pointwise multiset sum between type contexts.

This nothing but the CbV counterpart of De Carvalho's System R for CbN.

A concrete model for Plotkin's CbV: relational semantics

The semantics of a term is a the set of its types, together with their type contexts.

If $x_1, \ldots, x_n$ are pairwise distinct variables, and $\mathsf{fv}(t) \subseteq \{x_1, \ldots, x_n\}$, we say that $\vec{x} = (x_1, \ldots, x_n)$ is suitable for t. Given $\vec{x}$ suitable for t, its semantics is:

$$[\![t]\!]_{\vec{x}} := \{((P_1, \ldots, P_n), Q) \mid \exists\, \pi \rhd x_1 \colon P_1, \ldots, x_n \colon P_n \vdash t \colon Q\}\,.$$

Notation: $\pi \rhd \Gamma \vdash t \colon P$ means "π is a type derivation with conclusion $\Gamma \vdash t \colon P$".

Theorem (invariance under $\to_{\beta_v}$, Ehrhard 2012)

Let $\vec{x}$ be a suitable list of variables for t and u. If $t \to_{\beta_v} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.

Terminology: A denotational model for a λ-calculus is adequate if it characterizes all and only the normalisable terms.

$\rightsquigarrow$ In relational semantics: $[\![t]\!] = \emptyset$ if and only if t is not normalisable.
$\rightsquigarrow$ CbV relational semantics is not adequate for Plotkin's CbV:

$[\![(\lambda z.\delta)(xx)\delta]\!]_x = \emptyset$ but $(\lambda z.\delta)(xx)\delta$ is β_v-normal.

A concrete model for Plotkin's CbV: relational semantics

The semantics of a term is a the set of its types, together with their type contexts.

If $x_1, \ldots, x_n$ are pairwise distinct variables, and $\mathsf{fv}(t) \subseteq \{x_1, \ldots, x_n\}$, we say that $\vec{x} = (x_1, \ldots, x_n)$ is suitable for t. Given $\vec{x}$ suitable for t, its semantics is:

$$[\![t]\!]_{\vec{x}} := \{((P_1, \ldots, P_n), Q) \mid \exists\, \pi \rhd x_1 : P_1, \ldots, x_n : P_n \vdash t : Q\}\,.$$

Notation: $\pi \rhd \Gamma \vdash t : P$ means "π is a type derivation with conclusion $\Gamma \vdash t : P$".

Theorem (invariance under $\to_{\beta_v}$, Ehrhard 2012)

Let $\vec{x}$ be a suitable list of variables for t and u. If $t \to_{\beta_v} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.

Terminology: A denotational model for a λ-calculus is adequate if it characterizes all and only the normalisable terms.

$\rightsquigarrow$ In relational semantics: $[\![t]\!] = \emptyset$ if and only if t is not normalisable.
$\rightsquigarrow$ CbV relational semantics is not adequate for Plotkin's CbV:

$$[\![(\lambda z.\delta)(xx)\delta]\!]_x = \emptyset \qquad \text{but} \qquad (\lambda z.\delta)(xx)\delta \text{ is } \beta_v\text{-normal.}$$

But relational semantics is not a denotational model for λ_{fire}!

Relational semantics is not invariant under $\to_{\beta_f}$!

$$(\lambda z.y)(xx) \to_{\beta_f} y \qquad \text{or} \qquad (\lambda z.zz)(xx) \to_{\beta_f} (xx)(xx)$$

$$[\![(\lambda z.y)(xx)]\!]_{x,y} \neq [\![y]\!]_x \qquad\qquad [\![(\lambda z.z)(xx)]\!]_{x,y} \neq [\![(xx)(xx)]\!]_x$$

All counterexamples are due to $\to_{\beta_i}$, when an inert term is erased or duplicated.

- Relational semantics is not adequate for Plotkin's CbV;
- Relational semantics is not a denotational model for λ_{fire}.

Goal: Forcing relational semantics to be an adequate denotational model for λ_{fire}.

Rmk: Any denotational model for the CbN λ-calculus is invariant under $\to_{\beta_f}$ (since $\to_{\beta_f} \subseteq \to_\beta$), but there is no hope that it could be adequate for λ_{fire}:

in CbN $[\![(\lambda x.y)(\delta\delta)]\!]_y = [\![y]\!]_y$ but y is β_f-normal, $(\lambda x.y)(\delta\delta)$ is β_f-diverging.

Ehrhard's methodological law (what I learned in my PhD)

If there is a mismatch between syntax and semantics, the problem is in the syntax.
$\rightsquigarrow$ Change the syntax!

But relational semantics is not a denotational model for λ_{fire}!

Relational semantics is not invariant under $\to_{\beta_f}$!

$$(\lambda z.y)(xx) \to_{\beta_f} y \qquad \text{or} \qquad (\lambda z.zz)(xx) \to_{\beta_f} (xx)(xx)$$

$$[\![(\lambda z.y)(xx)]\!]_{x,y} \neq [\![y]\!]_x \qquad\qquad [\![(\lambda z.z)(xx)]\!]_{x,y} \neq [\![(xx)(xx)]\!]_x$$

All counterexamples are due to $\to_{\beta_i}$, when an inert term is erased or duplicated.

- Relational semantics is not adequate for Plotkin's CbV;
- Relational semantics is not a denotational model for λ_{fire}.

Goal: Forcing relational semantics to be an adequate denotational model for λ_{fire}.

Rmk: Any denotational model for the CbN λ-calculus is invariant under $\to_{\beta_f}$ (since $\to_{\beta_f} \subseteq \to_{\beta}$), but there is no hope that it could be adequate for λ_{fire}:

in CbN $[\![(\lambda x.y)(\delta\delta)]\!]_y = [\![y]\!]_y$ but y is β_f-normal, $(\lambda x.y)(\delta\delta)$ is β_f-diverging.

Ehrhard's methodological law (what I learned in my PhD)

If there is a mismatch between syntax and semantics, the problem is in the syntax.
$\rightsquigarrow$ Change the syntax!

But relational semantics is not a denotational model for λ_{fire}!

Relational semantics is not invariant under $\to_{\beta_f}$!

$$(\lambda z.y)(xx) \to_{\beta_f} y \qquad \text{or} \qquad (\lambda z.zz)(xx) \to_{\beta_f} (xx)(xx)$$

$$[\![(\lambda z.y)(xx)]\!]_{x,y} \neq [\![y]\!]_x \qquad\qquad [\![(\lambda z.z)(xx)]\!]_{x,y} \neq [\![(xx)(xx)]\!]_x$$

All counterexamples are due to $\to_{\beta_i}$, when an inert term is erased or duplicated.

- Relational semantics is not adequate for Plotkin's CbV;
- Relational semantics is not a denotational model for λ_{fire}.

Goal: Forcing relational semantics to be an adequate denotational model for λ_{fire}.

Rmk: Any denotational model for the CbN λ-calculus is invariant under $\to_{\beta_f}$ (since $\to_{\beta_f} \subseteq \to_{\beta}$), but there is no hope that it could be adequate for λ_{fire}:

in CbN $[\![(\lambda x.y)(\delta\delta)]\!]_y = [\![y]\!]_y$ but y is β_f-normal, $(\lambda x.y)(\delta\delta)$ is β_f-diverging.

Ehrhard's methodological law (what I learned in my PhD)

If there is a mismatch between syntax and semantics, the problem is in the syntax.
$\rightsquigarrow$ Change the syntax!

Outline

The split fireball calculus $\lambda_{\mathsf{fire}}^{\mathsf{split}}$

Goal: A different presentation of λ_{fire} that has an adequate denotational model.

Terms, Values, Fireballs, Inert terms, Right ev. contexts — as for the fireball calculus λ_{fire}

$$
\begin{array}{rrcl}
\text{Environments} & E & ::= & \epsilon \mid [i/x]:E \\
\text{Programs} & p & ::= & (t, E) \\
\text{Rules} & (C\langle(\lambda x.t)v\rangle,\ E) & \to_{\beta_v} & (C\langle t\{v/x\}\rangle,\ E) \\
 & (C\langle(\lambda x.t)i\rangle,\ E) & \to_{\beta_i} & (C\langle t)\rangle,\ [i/x]:E) \\
\text{Reduction} & \to_{\beta_f} & := & \to_{\beta_v} \cup \to_{\beta_i}
\end{array}
$$

Proposition (Harmony for $\lambda_{\mathsf{fire}}^{\mathsf{split}}$)

A program p is normal iff $p = (f, E)$.

Unfolding (from $\lambda_{\mathsf{fire}}^{\mathsf{split}}$ to λ_{fire}): $\quad (t, \epsilon)\!\downarrow := t \quad (t, E@[i/x])\!\downarrow := (t, E)\!\downarrow\!\{i/x\}$.

Proposition (Strong bisimulation)

1. *Split to plain*: if $p \to_{\beta_f} q$ then $p\!\downarrow \to_{\beta_f} q\!\downarrow$.
2. *Plain to split*: if $p\!\downarrow \to_{\beta_f} u$ then there exists q such that $p \to_{\beta_f} q$ and $q\!\downarrow = u$.

The multi type system for $\lambda_{\mathsf{fire}}^{\mathsf{split}}$

$$\frac{}{x\colon P \vdash x\colon P}\ \mathsf{ax} \qquad \frac{\Gamma \vdash t\colon [P \multimap Q] \qquad \Delta \vdash u\colon P}{\Gamma \uplus \Delta \vdash tu\colon Q}\ @$$

$$\frac{\Gamma_1, x\colon P_1 \vdash t\colon Q_1 \qquad \overset{n\in\mathbb{N}}{\cdots} \qquad \Gamma_n, x\colon P_n \vdash t\colon Q_n}{\Gamma_1 \uplus \cdots \uplus \Gamma_n \vdash \lambda x.t\colon [P_1 \multimap Q_1, \ldots, P_n \multimap Q_n]}\ \lambda$$

$$\frac{\Gamma \vdash t\colon P}{\Gamma \vdash (t, \epsilon)\colon P}\ \mathsf{es}_\epsilon \qquad \frac{\Gamma, x\colon P \vdash (t, E)\colon Q \qquad \Delta \vdash i\colon P}{\Gamma \uplus \Delta \vdash (t, E@[i/x])\colon Q}\ \mathsf{es}_@$$

1. Size $|t|$ of a term t: the number of its applications not in the scope of λ's.
2. Size $|p|$ of a program (t, E): $|t|$ plus the sizes of the inert terms in E.
3. Size $|\pi|$ of a type derivation π: the number of its @ rules.

Examples and intuitions about multi types for CbV

Example:

$$\pi_{II} = \cfrac{\cfrac{\cfrac{}{x\colon[\,]\vdash x\colon[\,]}\ \mathsf{ax}}{\vdash I\colon[[\,]\multimap[\,]]}\ \lambda \qquad \cfrac{}{\vdash I\colon[\,]}\ \lambda}{\cfrac{\vdash II\colon[\,]}{\vdash (II,\epsilon)\colon[\,]}\ \mathsf{es}_\epsilon}\ @ \qquad \pi_I = \cfrac{\cfrac{}{\vdash I\colon[\,]}\ \lambda}{\vdash (I,\epsilon)\colon[\,]}\ \mathsf{es}_\epsilon \quad \text{where } I := \lambda x.x$$

Note that $(II,\epsilon) \to_{\beta_f} (I,\epsilon)$ (with 1 β_f-step) and $|\pi_{II}| = 1 = |\pi_I| + 1$.

Idea: [] is the type of terms that can be erased.

- In multi types for CbN, every term, even a diverging one, is typable with [].
  $\rightsquigarrow$ in CbN every term can be erased, even the diverging ones;
  $\rightsquigarrow$ in CbN adequacy is formulated with respect to non-empty types.
- In multi-types for CbV, terms have to be evaluated before being erased.
  $\rightsquigarrow$ in CbV, normalisable terms and erasable terms coincide;
  $\rightsquigarrow$ in CbV, a term is typable if and only if it is typable with [].

Examples and intuitions about multi types for CbV

Example:

$$\pi_{II} = \cfrac{\cfrac{\cfrac{}{x\colon[\,] \vdash x\colon[\,]}\,\mathsf{ax}}{\vdash I\colon[[\,]\multimap[\,]]}\,\lambda \qquad \cfrac{}{\vdash I\colon[\,]}\,\lambda}{\cfrac{\vdash II\colon[\,]}{\vdash (II,\epsilon)\colon[\,]}\,\mathsf{es}_\epsilon}\,@ \qquad \pi_I = \cfrac{\cfrac{}{\vdash I\colon[\,]}\,\lambda}{\vdash (I,\epsilon)\colon[\,]}\,\mathsf{es}_\epsilon \qquad \text{where } I := \lambda x.x$$

Note that $(II,\epsilon) \to_{\beta_f} (I,\epsilon)$ (with 1 β_f-step) and $|\pi_{II}| = 1 = |\pi_I| + 1$.

Idea: $[\,]$ is the type of terms that can be erased.

- In multi types for CbN, every term, even a diverging one, is typable with $[\,]$.
  $\rightsquigarrow$ in CbN every term can be erased, even the diverging ones;
  $\rightsquigarrow$ in CbN adequacy is formulated with respect to non-empty types.
- In multi-types for CbV, terms have to be evaluated before being erased.
  $\rightsquigarrow$ in CbV, normalisable terms and erasable terms coincide;
  $\rightsquigarrow$ in CbV, a term is typable if and only if it is typable with $[\,]$.

Examples and intuitions about multi types for CbV

Example:

$$\pi_{II} = \dfrac{\dfrac{\dfrac{\dfrac{}{x:[\,] \vdash x:[\,]}\,\mathsf{ax}}{\vdash I:[[\,] \multimap [\,]]}\,\lambda \qquad \dfrac{}{\vdash I:[\,]}\,\lambda}{\vdash II:[\,]}\,@}{\vdash (II,\epsilon):[\,]}\,\mathsf{es}_\epsilon \qquad \pi_I = \dfrac{\dfrac{}{\vdash I:[\,]}\,\lambda}{\vdash (I,\epsilon):[\,]}\,\mathsf{es}_\epsilon \quad \text{where } I := \lambda x.x$$

Note that $(II,\epsilon) \to_{\beta_f} (I,\epsilon)$ (with 1 β_f-step) and $|\pi_{II}| = 1 = |\pi_I| + 1$.

Idea: $[\,]$ is the type of terms that can be erased.

- In multi types for CbN, every term, even a diverging one, is typable with $[\,]$.
  $\rightsquigarrow$ in CbN every term can be erased, even the diverging ones;
  $\rightsquigarrow$ in CbN adequacy is formulated with respect to non-empty types.
- In multi-types for CbV, terms have to be evaluated before being erased.
  $\rightsquigarrow$ in CbV, normalisable terms and erasable terms coincide;
  $\rightsquigarrow$ in CbV, a term is typable if and only if it is typable with $[\,]$.

Examples and intuitions about multi types for CbV

Example:

$$\pi_{II} = \cfrac{\cfrac{\cfrac{}{x\colon[\,]\vdash x\colon[\,]}\ \text{ax}}{\vdash I\colon[[\,]\multimap[\,]]}\ \lambda \qquad \cfrac{}{\vdash I\colon[\,]}\ \lambda}{\cfrac{\vdash II\colon[\,]}{\vdash (II,\epsilon)\colon[\,]}\ \text{es}_\epsilon}\ @ \qquad \pi_I = \cfrac{\cfrac{}{\vdash I\colon[\,]}\ \lambda}{\vdash (I,\epsilon)\colon[\,]}\ \text{es}_\epsilon \quad \text{where } I := \lambda x.x$$

Note that $(II,\epsilon) \to_{\beta_f} (I,\epsilon)$ (with 1 β_f-step) and $|\pi_{II}| = 1 = |\pi_I| + 1$.

Idea: $[\,]$ is the type of terms that can be erased.

- In multi types for CbN, every term, even a diverging one, is typable with $[\,]$.
  $\rightsquigarrow$ in CbN every term can be erased, even the diverging ones;
  $\rightsquigarrow$ in CbN adequacy is formulated with respect to non-empty types.
- In multi-types for CbV, terms have to be evaluated before being erased.
  $\rightsquigarrow$ in CbV, normalisable terms and erasable terms coincide;
  $\rightsquigarrow$ in CbV, a term is typable if and only if it is typable with $[\,]$.

Properties of multi types for CbV: correctness

Proposition (Type derivations bound the size of normal forms)

Let $\pi \rhd \Gamma \vdash p : P$ be a type derivation for a normal program p. Then $|p| \leq |\pi|$.

Proposition (quantitative subject reduction)

Let p and q be programs and $\pi \rhd \Gamma \vdash p : P$ be a type derivation. If $p \to_{\beta_f} q$ then there exists a type derivation $\sigma \rhd \Gamma \vdash q : P$ such that $|\pi| = |\sigma| + 1$.

From propositions above, it follows that typability implies termination:

Theorem (correctness)

Let $\pi \rhd \Gamma \vdash p : P$ be a type derivation. Then there is a normalising evaluation $d : p \to^*_{\beta_f} q$ such that $|d| + |q| \leq |\pi|$.

Properties of multi types for CbV: correctness

Proposition (Type derivations bound the size of normal forms)

Let $\pi \rhd \Gamma \vdash p : P$ be a type derivation for a normal program p. Then $|p| \leq |\pi|$.

Proposition (quantitative subject reduction)

Let p and q be programs and $\pi \rhd \Gamma \vdash p : P$ be a type derivation. If $p \to_{\beta_f} q$ then there exists a type derivation $\sigma \rhd \Gamma \vdash q : P$ such that $|\pi| = |\sigma| + 1$.

From propositions above, it follows that typability implies termination:

Theorem (correctness)

Let $\pi \rhd \Gamma \vdash p : P$ be a type derivation. Then there is a normalising evaluation $d : p \to_{\beta_f}^{*} q$ such that $|d| + |q| \leq |\pi|$.

Properties of multi types for CbV: completeness

Proposition (normal forms are typable)

1. *Normal program:* For any normal program, there exists a type derivation $\pi \rhd \Gamma \vdash p : P$ for some type context Γ and some multi type P.
2. *Inert term:* For any multi type Q and any inert term i, there exists a type derivation $\sigma \rhd \sigma \vdash i : Q$ for some type context Δ.

Proposition (quantitative subject expansion)

Let p and q be programs and $\sigma \rhd \Gamma \vdash q : P$ be a type derivation. If $p \to_{\beta_f} q$ then there exists a type derivation $\pi \rhd \Gamma \vdash p : P$ such that $|\pi| = |\sigma| + 1$.

From propositions above, it follows that termination implies typability:

Theorem (completeness)

Let $d : p \to_{\beta_f}^{*} q$ be a normalising evaluation. Then there is a type derivation $\pi \rhd \Gamma \vdash p : P$, and it satisfies $|d| + |q| \leq |\pi|$.

Properties of multi types for CbV: completeness

Proposition (normal forms are typable)

1. *Normal program:* For any normal program, there exists a type derivation $\pi \rhd \Gamma \vdash p \colon P$ for some type context Γ and some multi type P.
2. *Inert term:* For any multi type Q and any inert term i, there exists a type derivation $\sigma \rhd \sigma \vdash i \colon Q$ for some type context Δ.

Proposition (quantitative subject expansion)

Let p and q be programs and $\sigma \rhd \Gamma \vdash q \colon P$ be a type derivation. If $p \to_{\beta_f} q$ then there exists a type derivation $\pi \rhd \Gamma \vdash p \colon P$ such that $|\pi| = |\sigma| + 1$.

From propositions above, it follows that termination implies typability:

Theorem (completeness)

Let $d \colon p \to_{\beta_f}^{*} q$ be a normalising evaluation. Then there is a type derivation $\pi \rhd \Gamma \vdash p \colon P$, and it satisfies $|d| + |q| \leq |\pi|$.

Relational semantics is an adequate model for $\lambda_{\mathsf{fire}}^{\mathsf{split}}$

Corollary (invariance)

Let $\vec{x}$ be a suitable list of variables for p and q. If $p \to_{\beta_f} q$ then $\llbracket p \rrbracket_{\vec{x}} = \llbracket q \rrbracket_{\vec{x}}$.

Corollary (adequacy)

Let $\vec{x}$ be a suitable list of variables for p. The following are equivalent:

1. *Termination:* p is β_f-normalizable;
2. *Typability:* there is a type derivation $\pi \triangleright \Gamma \vdash p\colon P$ for some Γ and P;
3. *Non-empty denotation:* $\llbracket p \rrbracket_{\vec{x}} \neq \emptyset$.

Rmk: $\lambda_{\mathsf{fire}}^{\mathsf{split}}$ and λ_{fire} and Plotkin's CbV are the same calculus when restricted to closed terms.

$\rightsquigarrow$ Adequacy theorem above shows also that relational semantics is adequate for closed Plotkin's CbV.

Relational semantics is an adequate model for $\lambda_{\text{fire}}^{\text{split}}$

Corollary (invariance)

Let $\vec{x}$ be a suitable list of variables for p and q. If $p \to_{\beta_f} q$ then $[\![p]\!]_{\vec{x}} = [\![q]\!]_{\vec{x}}$.

Corollary (adequacy)

Let $\vec{x}$ be a suitable list of variables for p. The following are equivalent:

1. *Termination:* p is β_f-normalizable;
2. *Typability:* there is a type derivation $\pi \triangleright \Gamma \vdash p : P$ for some Γ and P;
3. *Non-empty denotation:* $[\![p]\!]_{\vec{x}} \neq \emptyset$.

Rmk: $\lambda_{\text{fire}}^{\text{split}}$ and λ_{fire} and Plotkin's CbV are the same calculus when restricted to closed terms.
$\rightsquigarrow$ Adequacy theorem above shows also that relational semantics is adequate for closed Plotkin's CbV.

Outline

Tight type derivations

In CbN, multi-types can provide exact bounds for the *evaluation length* and for the *size of normal forms*.

$\rightsquigarrow$ Can we extract this quantitative information in Open CbV as well? Yes!

We define two subsets of linear types and multi types:

inert linear types $L^i ::= [\,] \multimap P^i$ inert multi types $P^i ::= [L^i_1, \ldots, L^i_n]$ $(n \geq 0)$

A type context Γ is inert if it assigns only inert multi types to variables.
A type derivation $\pi \triangleright \Gamma \vdash p : P$ is

- inert if Γ is a inert type context;
- tight if π is inert and $P = [\,]$.

Intuition: Any β_f-normalisable (i.e. any typable) program is typable with type $[\,]$.
More precisely, with a tight type derivation.

Tight type derivations

In CbN, multi-types can provide exact bounds for the *evaluation length* and for the *size of normal forms*.
$\rightsquigarrow$ Can we extract this quantitative information in Open CbV as well? Yes!

We define two subsets of linear types and multi types:

inert linear types $L^i ::= [\,] \multimap P^i$ inert multi types $P^i ::= [L^i_1, \dots, L^i_n]$ $(n \geq 0)$

A type context Γ is inert if it assigns only inert multi types to variables.
A type derivation $\pi \rhd \Gamma \vdash p \colon P$ is

- inert if Γ is a inert type context;
- tight if π is inert and $P = [\,]$.

Intuition: Any β_f-normalisable (i.e. any typable) program is typable with type $[\,]$. More precisely, with a tight type derivation.

Exact bounds for Open CbV

Tight type derivations have a nice property with respect to normal forms:

Lemma (tight type derivations are minimal)

If $\pi \triangleright \Gamma \vdash p : [\,]$ is a tight type derivation and p is β_f-normal, then $|p| = |\pi|$ and $|\pi|$ is minimal among the type derivations of p.

We can refine correctness and completeness with exact quantitative information:

Theorem (tight correctness)

If $\pi \triangleright \Gamma \vdash p : [\,]$ is a tight type derivation, then there is a normalising evaluation $d : p \to^*_{\beta_f} q$ such that $|d| + |q| = |\pi|$. And if q is a value, then $|d| = |\pi|$.

Theorem (tight completeness)

If $d : p \to^*_{\beta_f} q$ is a normalizing evaluation, then there is a tight type derivation $\pi \triangleright \Gamma \vdash p : [\,]$, and it satisfies $|d| + |q| = |\pi|$. And if q is a value, then $|d| = |\pi|$.

Exact bounds for Open CbV

Tight type derivations have a nice property with respect to normal forms:

Lemma (tight type derivations are minimal)

If $\pi \rhd \Gamma \vdash p : [\,]$ is a tight type derivation and p is β_f-normal, then $|p| = |\pi|$ and $|\pi|$ is minimal among the type derivations of p.

We can refine correctness and completeness with exact quantitative information:

Theorem (tight correctness)

If $\pi \rhd \Gamma \vdash p : [\,]$ is a tight type derivation, then there is a normalising evaluation $d : p \to^*_{\beta_f} q$ such that $|d| + |q| = |\pi|$. And if q is a value, then $|d| = |\pi|$.

Theorem (tight completeness)

If $d : p \to^*_{\beta_f} q$ is a normalizing evaluation, then there is a tight type derivation $\pi \rhd \Gamma \vdash p : [\,]$, and it satisfies $|d| + |q| = |\pi|$. And if q is a value, then $|d| = |\pi|$.

What are we counting?

$\lambda_{\text{fire}}^{\text{split}}$ mimics LL proof-nets behavior via "boring" translation $A \Rightarrow B \overset{(\cdot)^v}{\rightsquigarrow} !A^v \multimap !B^v$.

$$(C\langle(\lambda x.t)v\rangle, E) \rightarrow_{\beta_v} (C\langle t\{v/x\}\rangle, E) \quad (C\langle(\lambda x.t)i\rangle, E) \rightarrow_{\beta_i} (C\langle t\rangle, [i/x] : E)$$

- a β_v-step corresponds to a multiplicative followed by an exponential step;
- a β_i-step corresponds to a multiplicative step.

$\rightsquigarrow$ The number of β_f-steps is the number of multiplicative steps in LL proof-nets.
$\rightsquigarrow$ Tight derivations count the number of multiplicative steps to reach a normal form of a LL proof-net.

Rmk: The number of β_f-steps is a reasonable cost model ($\lambda_{\text{fire}}^{\text{split}}$ can be implemented on a RAM with an overhead that is linear in the number of β_f-step).

Summing up: At least in the CbV fragment of LL:

- the multiplicative step is the computationally meaningful cut-elimination step;
- the exponential step just allows evaluation to go on.

What are we counting?

$\lambda_{\text{fire}}^{\text{split}}$ mimics LL proof-nets behavior via "boring" translation $A \Rightarrow B \overset{(\cdot)^v}{\rightsquigarrow} !A^v \multimap !B^v$.

$$(C\langle(\lambda x.t)v\rangle, E) \to_{\beta_v} (C\langle t\{v/x\}\rangle, E) \quad (C\langle(\lambda x.t)i\rangle, E) \to_{\beta_i} (C\langle t\rangle, [i/x] : E)$$

- a β_v-step corresponds to a multiplicative followed by an exponential step;
- a β_i-step corresponds to a multiplicative step.

$\rightsquigarrow$ The number of β_f-steps is the number of multiplicative steps in LL proof-nets.
$\rightsquigarrow$ Tight derivations count the number of multiplicative steps to reach a normal form of a LL proof-net.

Rmk: The number of β_f-steps is a reasonable cost model ($\lambda_{\text{fire}}^{\text{split}}$ can be implemented on a RAM with an overhead that is linear in the number of β_f-step).

Summing up: At least in the CbV fragment of LL:

- the multiplicative step is the computationally meaningful cut-elimination step;
- the exponential step just allows evaluation to go on.