Variations on a theme: call-by-value and factorization

Beniamino Accattoli

INRIA & LIX, Ecole Polytechnique

Outline

1. Call-by-value λ-calculus

2. Factorization

Outline

1. Call-by-value λ-calculus
2. Factorization

Call-by-value

- Plotkin's ***call-by-value*** λ-calculus:

$$t ::= V \mid t\ t$$

$$V ::= x \mid \lambda x.t$$

$$\beta_v \ \textbf{\textit{rule}}: (\lambda x.t)\ V \rightarrow_{\beta_v} t\{x/V\}$$

- Most ***functional programming languages*** are ***CBV***.
- Most ***works on*** λ***-calculus*** are ***call-by-name (CBN)***.

Call-by-value

- Plotkin's calculus is ***not satisfactory*** for various reasons.
- Semantic models do not faithful reflect bueibdivergence.
- Let $\Delta = \lambda x.xx$. Now consider:

$$M = (\lambda x.\Delta)\ (y\ z)\ \Delta$$

  Semantically M should be ***divergent***, but it is a β_v***-normal form***!
- Problem studied by Luca ***Paolini*** and Simona ***Ronchi della Rocca*** ("call-by-value solvability").
- Another problem: the ***completeness*** of CPS-translations.

λ-calculus and Linear Logic

- λ-calculus can be represented in various ways inside Linear Logic.
- Two main translations:
  1. ***Call-by-name***: $(A \Rightarrow B)^n := (!A^n) \multimap B^n$.
  2. ***Call-by-value***: $(A \Rightarrow B)^v := !(A^v \multimap B^v)$.
- Both appear in Girard's seminal paper (1987)
- Girard calls the second ***boring***.
- ***Sad consequence***: the CBV-translation is ***less known and understood***.

Call-by-value and Linear Logic

- The translations are typed but both can be extended to ***pure*** CBN and CBV λ-calculus by means of ***recursive types***.
- ***Curious fact***:

$$M = (\lambda x.\Delta)\ (y\ z)\ \Delta$$

  diverges when represented in LL Proof-Nets via the CBV translation (which is good).
- ***Idea***: to extract the calculus corresponding to CBV Proof-Nets.
- Relation with Proof-Nets requires ***explicit substitutions***.
- But here ES are evaluated in just ***one shot***.

The value-substitution calculus $\lambda_{\texttt{vsub}}$

- Let $\mathtt{L}$ be a possibly empty list $[x_1/u_1]\ldots[x_n/u_n]$.
- Define $\lambda_{\texttt{vsub}}$ as:

$$\begin{array}{rcl} t & ::= & V \mid t\ t \mid t[x/u] \\ V & ::= & x \mid \lambda x.t \end{array}$$

- Rules:

$$\begin{array}{lcl} (\lambda x.t)\mathtt{L}\ s & \rightarrow_{\mathtt{dB}} & t[x/s]\mathtt{L} \\ t[x/V\mathtt{L}] & \rightarrow_{\mathtt{sv}} & t\{x/V\}\mathtt{L} \end{array}$$

- ***Note that s needs not*** to be a ***value***.
- ***Note that*** explicit substitutions can be reduced only if the content is a ***value***.
- ***Note*** the use of distance (*i.e.* $\mathtt{L}$).
- $\lambda_{\texttt{vsub}}$ is ***confluent***.

Solvability and explicit substitution

- Re-consider the problematic term:

$$M = (\lambda w.\Delta)\ (y\ z)\ \Delta$$

- Now let's look at it in our new framework:

$$\begin{array}{llll} (\lambda w.\Delta)\ (y\ z)\ \Delta & \rightarrow_{\mathtt{dB}} & \Delta[w/y\ z]\Delta & \rightarrow_{\mathtt{dB}} \\ & & (x\ x)[x/\Delta][w/y\ z] & \rightarrow_{\mathtt{sv}} \\ & & (\Delta\ \Delta)[w/y\ z] & \rightarrow \quad \ldots \end{array}$$

- ***M has no nf!*** (which is good)

Herbelin-Zimmerman's λ_{CBV}

- There is a similar calculus by Herbelin and Zimmerman, but without distance.
- The syntax is the same, but not the rules:

$$\begin{array}{rcl} t & ::= & V \mid t\ t \mid t[x/u] \\ V & ::= & x \mid \lambda x.t \end{array}$$

Operational rules			***Structural rules***		
$(\lambda x.t)\ s$	$\Rightarrow$	$t[x/s]$	$t[x/u[y/w]]$	$\rightarrow_{let_{let}}$	$t[x/u][y/w]$
$t[x/V]$	$\rightarrow_{let_v}$	$t\{x/V\}$	$t[x/u]\ w$	$\rightarrow_{let_{app}}$	$(t\ w)[x/u]$

- ***Note that s needs not*** to be a ***value***, but:
  - $(\lambda x.t)[y/w]\ s$ ***is not*** a $\Rightarrow$ redex.
  - $t[y/V[x/u]]$ ***is not*** a $\rightarrow_{let_v}$ redex.
- The ***structural rules*** become ***identities*** on Proof-Nets.

Simulation

- $\lambda_{\mathtt{vsub}}$ is an ***equational sub-calculus*** of λ_{CBV}:

$$\begin{array}{ccccc} (\lambda x.t)\mathtt{L}\ s & & \rightarrow_{\mathtt{dB}} & & t[x/s]\mathtt{L} \\ (\lambda x.t)\mathtt{L}\ s & \rightarrow^{*}_{let_{app}} & ((\lambda x.t)\ s)\mathtt{L} & \Rightarrow & t[x/s]\mathtt{L} \end{array}$$

$$\begin{array}{ccccc} t[x/V\mathtt{L}] & & \rightarrow_{\mathtt{sv}} & & t\{x/V\}\mathtt{L} \\ t[x/V\mathtt{L}] & \rightarrow^{*}_{let_{let}} & t[x/V]\mathtt{L} & \rightarrow_{let_{v}} & t\{V/x\}\mathtt{L} \end{array}$$

- ***Thus*** $\rightarrow_{\lambda_{\mathtt{vsub}}} \subseteq \rightarrow^{*}_{\lambda_{CBV}}$.

Inverse simulation

- Apparently, $\lambda_{\mathtt{vsub}}$ is ***strictly contained*** in λ_{CBV}.
- These rules ***cannot be simulated***:

$$
\begin{array}{lll}
t[x/u[y/w]] & \rightarrow_{let_{let}} & t[x/u][y/w] \\
t[x/u]\ w & \rightarrow_{let_{app}} & (t\ w)[x/u]
\end{array}
$$

- But this ***is not*** quite true...

Structural congruence

- Let $\equiv_\circ$ be the ***equivalence relation*** generated by:

$$
\begin{array}{llll}
t[x/s][y/u] & \sim_{\circ_1} & t[y/u][x/s] & \text{if } x \notin \mathtt{fv}(u) \& y \notin \mathtt{fv}(s) \\
t\ u[x/s] & \sim_{\circ_2} & (t\ u)[x/s] & \text{if } x \notin \mathtt{fv}(t) \\
t[x/s]\ u & \sim_{\circ_3} & (t\ u)[x/s] & \text{if } x \notin \mathtt{fv}(u) \\
t[x/s[y/u]] & \sim_{\circ_4} & t[x/s][y/u] & \text{if } y \notin \mathtt{fv}(t)
\end{array}
$$

- $\equiv_\circ$ contains λ_{CBV} structural rules:

$$
\begin{array}{lll}
t[x/u[y/w]] & \rightarrow_{let_{let}} & t[x/u][y/w] \\
t[x/u]\ w & \rightarrow_{let_{app}} & (t\ w)[x/u]
\end{array}
$$

- ***Operational*** rules: $t \rightarrow_{\lambda_{CBV}} u$ implies $t \rightarrow_{\lambda_{\mathtt{vsub}}} u$.
- ***Structural*** rules: $t \rightarrow_{\lambda_{CBV}} u$ implies $t \equiv_\circ u$.
- Hence $\rightarrow_{\lambda_{CBV}} \subseteq (\rightarrow_{\lambda_{\mathtt{vsub}}} / \equiv_\circ)$.

Strong bisimulations

- $\equiv_\circ$ is a ***strong bisimulation***, *i.e.*:

$$\begin{array}{ccc} t & & \\ \equiv_\circ & & \\ u & \rightarrow_{\lambda_{\mathtt{vsub}}} & u' \end{array} \quad \Rightarrow \quad \exists\ t' \text{ s.t.} \quad \begin{array}{ccc} t & \rightarrow_{\lambda_{\mathtt{vsub}}} & t' \\ \equiv_\circ & & \equiv_\circ \\ t & \rightarrow_{\lambda_{\mathtt{vsub}}} & u' \end{array}$$

- ***Rewriting modulo*** a ***strong bisimulation*** preserves ***confluence*** and ***strong normalisation***.
- If $t \equiv_\circ u$ then t and u ***map*** to the same ***Proof-Net***.
- Then they can really be considered as ***the same object***.

Concluding

- In $\lambda_{\mathtt{vsub}}$ there is a ***good match*** between semantics and divergence.
- Recent work in collaboration with Luca Paolini (FLOPS 2012).
- This work gives an ***operational characterization*** of CBV-solvablity (a semantic notion).
- The operational characterization uses crucially ***two factorization theorems***.

Outline

1. Call-by-value λ-calculus

2. Factorization

Confluence

- A system S if ***confluent*** when:

$$\begin{array}{ccc} t & \to^* & u_1 \\ \downarrow_* & & \\ u_2 & & \end{array} \quad \text{implies } \exists v \text{ s.t.} \quad \begin{array}{ccc} t & \to^* & u_1 \\ \downarrow_* & & \downarrow_* \\ u_2 & \to^* & v \end{array}$$

- A system S if ***locally confluent*** when:

$$\begin{array}{ccc} t & \to & u_1 \\ \downarrow & & \\ u_2 & & \end{array} \quad \text{implies } \exists v \text{ s.t.} \quad \begin{array}{ccc} t & \to & u_1 \\ \downarrow & & \downarrow_* \\ u_2 & \to^* & v \end{array}$$

- ***Termination*** $\Rightarrow$ Confluence = Local Confluence (Newman's Lemma).

General Idea

- λ-calculus has just one rule:

$$(\lambda x.t)\ u \rightarrow_{\beta} t\{x/u\}$$

  which does not terminate.

- Explicit substitutions, ***abstractly***:

  1. ***Creation of substitutions***: $(\lambda x.t)\mathtt{L}\ u \rightarrow_{\mathtt{dB}} t[x/u]$.
  2. ***Set*** of rules ***executing*** substitutions: $t[x/u] \rightarrow^{*} t\{x/u\}$.

- ***Key property***: each rule of an ES-calculus ***terminates***.

- So ES-calculi are sort of locally terminating systems, which are globally non-terminating.

Local termination and confluence

- New proof technique for ***confluence***.
- Prove ***local*** confluence of ***each rule alone***.
- Termination gives ***confluence*** of each rule.
- Hindley-Rosen Lemma: if two reductions $\to_1$ and $\to_2$ ***commute***:

$$\begin{array}{ccc} t & \to_1^* & u_1 \\ \downarrow_{*2} & & \\ u_2 & & \end{array} \quad \text{implies } \exists v \text{ s.t.} \quad \begin{array}{ccc} t & \to_1^* & u_1 \\ \downarrow_{*2} & & \downarrow_{*2} \\ u_2 & \to_1^* & v \end{array}$$

  and are ***confluent*** then $\to_1 \cup \to_2$ is ***confluent***.
- Prove commutation of each pair of rule.
- Termination often reduces commutation to ***local*** commutation.

Local termination

- So in ES-calculi a ***global*** property as confluence can be reduced to ***local*** confluence and ***local*** commutation.
- Surprising: in λ-calculus confluence do ***not*** reduce to local confluence.
- ES-calculi are ***more complex*** than λ-calculus, but local termination provides ***new proof techniques***.
- Another notion which can be ***localized*** is factorization.

Standardization

- Termination is about the ***existence*** of results.
- Confluence is about the ***unicity*** of results.
- Standardization instead is about ***how to compute***.
- It identifies a specific class of reductions to which any other reduction can be transformed by ***permuting its steps***.
- It has many important corollaries, in particular it gives a ***normalizing strategy*** for evaluation.
- Many applications require a simpler form, called ***factorization***.

Factorization

- Factorization is a simple form of ***standardization***.
- ***Head contexts*** in λ-calculus:

$$H ::= [\cdot] \mid \lambda x.H \mid H\ t$$

- ***Head reduction*** $\rightarrow_h$ in λ-calculus is the closure by head contexts H of:

$$(\lambda x.t)\ u \mapsto_\beta t\{x/u\}$$

- ***Internal*** reduction is the ***complement*** of head reduction, *i.e.*$\rightarrow_i := \rightarrow_\beta \setminus \rightarrow_h$.
- ***Factorization theorem***:

Every reduction $t \rightarrow_\beta^* u$ can be ***re-organized*** as $t \rightarrow_h^* \rightarrow_i^* u$

Factorization theorem in λ-calculus

- At first sight factorization is ***easy***.
- ***Local*** diagram permutation diagram:

$$\begin{tikzcd} t \arrow[r, "i"] \arrow[d, dashed, "h+"'] \arrow[dr, phantom, "\swarrow\!\!\swarrow"] & u \arrow[d, "h"] \\ v \arrow[r, dashed, "*"', "i"] & w \end{tikzcd}$$

- Two ***abstract*** lemmas, similar to Newman's, imply the factorization theorem when:
  1. $\to_h^+$ is composed of ***at most one step***, or
  2. $\to_h$ is ***strongly normalizing***.

Factorization is non-trivial

- Unfortunately, $\to_\beta$ ***lacks*** both properties.
- The sequence $\to_h^+$ can have length > 1:

$$
\begin{tikzcd}
(\lambda x.x\ x)\ (I\ I) \arrow[rr, "\mathtt{i}"] \arrow[d, dashed, "\mathtt{h}"] & \arrow[d, phantom, "\Swarrow"] & (\lambda x.x\ x)\ I \arrow[d, "\mathtt{h}"] \\
(I\ I)\ (I\ I) \arrow[r, dashed, "\mathtt{h}"] & I\ (I\ I) \arrow[r, dashed, "\mathtt{i}"] & I\ I
\end{tikzcd}
$$

- $\to_h$ is ***not*** strongly normalising:

$$(\lambda x.x\ x)\ \lambda x.x\ x \to_h (\lambda x.x\ x)\ \lambda x.x\ x \to_h \ldots$$

Factorization and explicit substitutions

- The basic ES-calculus $\lambda_{\mathtt{sub}}$:

$$\begin{array}{lll} (\lambda x.t)\mathtt{L}\ s & \mapsto_{\mathtt{dB}} & t[x/s]\mathtt{L} \\ t[x/u] & \mapsto_{\mathtt{s}} & t\{x/u\} \end{array}$$

- Define ***head contexts*** as:

$$H ::= [\cdot] \mid \lambda x.H \mid H\ t \mid H[x/t]$$

- We get ***four*** reductions:

	$\rightarrow_i$	$\rightarrow_h$
$\rightarrow_{\mathtt{dB}}$	$\rightarrow_{\mathtt{dB}i}$	$\rightarrow_{\mathtt{dB}h}$
$\rightarrow_{\mathtt{s}}$	$\rightarrow_{\mathtt{s}i}$	$\rightarrow_{\mathtt{s}h}$

- Remember: they all ***terminates***.

Factorization for $\lambda_{\mathtt{sub}}$

- We get four diagrams:

$$\begin{tikzcd} t \arrow[r, "\mathtt{dB}i"] \arrow[d, dashed, "*\mathtt{dB}h"'] \arrow[dr, phantom, "\Swarrow"] & u \arrow[d, "\mathtt{dB}h"] \\ v \arrow[r, dashed, "*"', "\mathtt{dB}i"'] & w \end{tikzcd} \qquad \begin{tikzcd} t \arrow[r, "\mathtt{s}i"] \arrow[d, dashed, "*\mathtt{s}h"'] \arrow[dr, phantom, "\Swarrow"] & u \arrow[d, "\mathtt{s}h"] \\ v \arrow[r, dashed, "*"', "\mathtt{s}i"'] & w \end{tikzcd}$$

$$\begin{tikzcd} t \arrow[r, "\mathtt{s}i"] \arrow[d, dashed, "\mathtt{dB}h"'] \arrow[dr, phantom, "\Swarrow"] & u \arrow[d, "\mathtt{dB}h"] \\ v \arrow[r, dashed, "*"', "\mathtt{s}"'] & w \end{tikzcd} \qquad \begin{tikzcd} t \arrow[r, "\mathtt{dB}i"] \arrow[d, dashed, "\mathtt{s}h"'] \arrow[dr, phantom, "\Swarrow"] & u \arrow[d, "\mathtt{s}h"] \\ v \arrow[r, dashed, "*"', "\mathtt{dB}"'] & w \end{tikzcd}$$

- The abstract lemmas get ***factorization of each single diagram*** (a new abstract lemma is required).

- Glueing the obtained ***local factorizations*** (easy to do) we get the factorization theorem for $\lambda_{\mathtt{sub}}$.

Back to call-by-value

- Call-by-value factors with respect to ***weak reductions***.
- Weak contexts:

$$W ::= [\cdot] \mid W\ t \mid t\ W \mid W[x/t] \mid t[x/W]$$

- Weak reduction $\rightarrow_{\mathtt{w}}$: closure of the rules by weak contexts.
- Same technique gives ***factorization***: if $t \rightarrow^*_{\lambda_{\mathtt{vsub}}} u$ then $t \rightarrow^*_{\mathtt{w}} \rightarrow^*_{\neg\mathtt{w}} u$.
- Factorization also with respect to ***stratified weak reduction***, defined from ***head-weak*** contexts $H[W]$.

Linear substitution calculus

- The linear substitution calculus $\lambda_{\mathtt{ls}}$:

$$
\begin{array}{llll}
(\lambda x.t)L\ u & \to_{\mathtt{dB}} & t[x/u]L & \\
C[x][x/u] & \to_{\mathtt{ls}} & C[u][x/u] & \\
t[x/u] & \to_{\mathtt{w}} & t & x \notin \mathtt{fv}(t)
\end{array}
$$

- Head factorization does ***not*** hold:

$$x[x/y[y/z]][z/u] \to_{\mathtt{ls}i} x[x/z[y/z]][z/u] \to_{\mathtt{ls}h} x[x/u[y/z]][z/u]$$

The two steps cannot be permuted.

Linear substitution calculus

- New notion of head reduction.
- We need to refine the notion of head substitution.
- Set:

$$H[x][x/u] \quad \multimap_{\mathtt{hls}} \quad H[u][x/u]$$

- Then define ***linear head reduction*** as $H[\multimap_{\mathrm{dB}}] \cup H[\multimap_{\mathtt{hls}}]$.
- The linear substitution calculus enjoys factorization with respect to linear head reduction.
- Linear head reduction can be seen as an abstraction of ***Krivine Abstract Machine*** (Danos and Regnier).

Linear Head reduction

- Linear head reduction arises ***naturally*** and ***repeatedly*** in the LL literature.
- First studied in connection with Proof-Nets (Mascari, Pedicini).
- Then in ***semantics***: geometry of interaction and game semantics.
- Then in connection with the π***-calculus*** (Mazza) and ***differential*** λ***-calculus*** (Ehrhard, Regnier).
- Recently it has been shown to induce a measure for ***complexity*** (Accattoli, Dal Lago).

THANKS!