The Permutative λ-calculus

Beniamino Accattoli[1] Delia Kesner[2]

INRIA and LIX (École Polytechnique)

PPS (CNRS and Université Paris-Diderot)

Outline

Outline

λ-calculus

- ***Syntax***:

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

- ***Evaluation***: β-reduction

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

- ***Expressiveness***: Turing-complete and ***machine independent***.
- ***Mathematics***: Intuitionistic ***Logic***, Cartesian Closed ***Categories***.
- ***Applications***:
  - Functional Languages,
  - Theorem Provers,
  - Linguistics,
  - Polymorphism,
  - MapReduce.

Rewriting

- Some ***examples*** of β-reductions:
  - ***Duplication***: $(\lambda x.x\ x)\ t \to_\beta t\ t$.
  - ***Erasure***: $(\lambda x.y)\ t \to_\beta y$.
  - ***Linear replacement***: $(\lambda x.u\ x)\ t \to_\beta u\ t$.
- β-reduction is ***non-deterministic***, but ***well-behaved***, *i.e.*:

$$\begin{array}{ccccc} (\lambda x.x\ x)\ (\ (\lambda y.z)\ u\ ) & & \to & & (\lambda x.x\ x)\ z \\ \downarrow & & & & \downarrow \\ (\ (\lambda y.z)\ u\ )\ (\ (\lambda y.z)\ u\ ) & \to & (\ (\lambda y.z)\ u\ )\ z & \to & z\ z \end{array}$$

- But β-reduction ***may not terminate***:

$$(\lambda x.x\ x)\ \lambda y.y\ y \to_\beta (\lambda y.y\ y)\ \lambda y.y\ y \to_\beta (\lambda y.y\ y)\ \lambda y.y\ y \ldots$$

Outline

Permutations

- λ-calculus can be extended with ***permutations of constructors***.
- β-reduction alone ***does not allow to postpone erasing steps***:

$$\underbrace{(\lambda x.\lambda y.y)\ t\ v \ \rightarrow_\beta\ (\lambda y.y)\ v}_{\text{erasing step}} \ \rightarrow_\beta\ v$$

- ***One solution***: a ***rule*** permuting the two λs (De Groote, 1993):

$$(\lambda x.\lambda y.t)\ u \ \rightarrow_p\ \lambda y.((\lambda x.t)\ u) \quad \text{if } y \notin \mathtt{fv}(u)$$

- So that:

$$(\lambda x.\lambda y.y)\ t\ v \ \rightarrow_p\ (\lambda y.((\lambda x.y)\ t))\ v \ \rightarrow_\beta\ \underbrace{(\lambda x.v)\ t \ \rightarrow_\beta\ v}_{\text{erasing step}}$$

Further examples

Some other cases where λ-calculus is extended with some ***permutation rules***:

1. Studying ***generalized notions of β-reduction*** (Kamareddine, 2000).
2. Relating λ-calculus and ***Linear Logic Proof-Nets*** (Regnier 1992).
3. Completeness of ***CPS-translation*** for the call-by-value λ-calculus (Sabry & Felleisen, 1992).
4. Mapping ***Moggi's monadic metalanguage*** on λ-calculus (Espirito Santo, Matthes & Pinto, 2009).

This work

Every extension of λ-calculus should enjoy:

1. ***Confluence***:

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

2. ***Preservation of β-strong normalization*** (PSN), *i.e.* no diverging behaviour is introduced by the extension:

   If t terminates with β then t ***terminates with the extension***.

The ***aim of this work***:

Unify and ***generalize*** all the extensions in the literature

The generalization

All the mentioned examples use ***rewriting rules*** as:

$$
\begin{array}{lcll}
(\lambda x.\lambda y.t)\ u & \to & \lambda y.((\lambda x.t)\ u) & \text{if } y \notin \mathtt{fv}(u) \\
(\lambda x.t\ v)\ u & \to & (\lambda x.t)\ u\ v & \text{if } x \notin \mathtt{fv}(v) \\
(\lambda x.t\ v)\ u & \to & t\ ((\lambda x.v)\ u) & \text{if } x \notin \mathtt{fv}(t)
\end{array}
$$

Our generalization consist in taking them as ***equations***:

$$
\begin{array}{lcll}
(\lambda x.\lambda y.t)\ u & \equiv_{\mathtt{P}} & \lambda y.((\lambda x.t)\ u) & \text{if } y \notin \mathtt{fv}(u) \\
(\lambda x.t\ v)\ u & \equiv_{\mathtt{P}} & (\lambda x.t)\ u\ v & \text{if } x \notin \mathtt{fv}(v) \\
(\lambda x.t\ v)\ u & \equiv_{\mathtt{P}} & t\ ((\lambda x.v)\ u) & \text{if } x \notin \mathtt{fv}(t)
\end{array}
$$

1. Prove ***confluence*** and ***PSN*** of β ***modulo*** $\equiv_{\mathtt{P}}$,
2. All previous results become ***instances*** of our result.

The permutative λ-calculus

The permutative λ-calculus $\Lambda_{\mathbb{P}}$ is given by:

- ***Syntax***:

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

- ***Evaluation***: β-reduction

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

Modulo:

$$\begin{array}{llll} (\lambda x.\lambda y.t)\ u & \equiv_{\mathbb{P}} & \lambda y.((\lambda x.t)\ u) & \text{if } y \notin \mathtt{fv}(u) \\ (\lambda x.t\ v)\ u & \equiv_{\mathbb{P}} & (\lambda x.t)\ u\ v & \text{if } x \notin \mathtt{fv}(v) \\ (\lambda x.t\ v)\ u & \equiv_{\mathbb{P}} & t\ ((\lambda x.v)\ u) & \text{if } x \notin \mathtt{fv}(t) \end{array}$$

- $\equiv_{\mathbb{P}}$ has a natural justification in terms of ***Linear Logic***.

The permutative λ-calculus

- λ-calculus corresponds to Intuitionistic Logic.
- Variations on λ-calculus usually correspond to ***different*** logics (modal, classical, linear).
- The permutative λ-calculus $\Lambda_{\mathtt{P}}$ extends λ-calculus ***within*** Intuitionistic Logic.
- ***Interest*** of $\Lambda_{\mathtt{P}}$:
  - unifies and generalizes many ***ad-hoc*** extensions.
  - $\Lambda_{\mathtt{P}}$ uses ***rewriting modulo***.
  - Confluence and PSN for $\Lambda_{\mathtt{P}}$ are ***challenging rewriting problems***.

Outline

1. λ-calculus
2. Permutative extension
3. Confluence
4. Explicit substitutions
5. Termination
6. Conclusions

Confluence

- The paper focus on proving ***confluence of* β *modulo*** $\equiv_{\mathtt{P}}$.
- λ-calculus does not terminate, so ***confluence does not reduce to local confluence*** (*i.e.* it is non-trivial).
- Standard proof-techniques:
  1. ***Parallel reduction*** (Tait-Martin Löf).
  2. ***Finite (super)developments***.
- Unfortunately, these techniques do ***not work*** for β modulo $\equiv_{\mathtt{P}}$.

Some words about developments

- An ***abstract development*** is a function $(\cdot)^\circ$ from terms to terms:
    1) $t \to_\beta u$ implies $u \to_\beta^* t^\circ$.
    2) $t \to_\beta u$ implies $t^\circ \to_\beta^* u^\circ$.
- If a system admits an abstract development than it is ***confluent*** (Van Oostrom).
- For ***confluence modulo*** one needs also a third property:
    3) $t \equiv_{\mathrm{P}} u$ implies $t^\circ = u^\circ$.

  For known notions of development property 3 ***does not hold***.

Outline

Proof Technique 1

- We introduce a ***new notion of development*** verifying properties 1-2-3.
- This notion is defined via a simple calculus of explicit substitutions (or `let` expressions) refining β-reduction.
- λ-calculus syntax + ***explicit substitutions***:

$$t, u, v ::= x \mid \lambda x.t \mid t\ u \mid t[x/u]$$

Meta-notation: $\mathtt{L} := [x_1/u_1]\ldots[x_k/u_k]$ with $k \geq 0$.

- Refined evaluation:

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

Main property

- The refinement simulates β-reduction:

$$(\lambda x.t)\ u \rightarrow_{\mathtt{dB}} t[x/u] \rightarrow_{\mathtt{sub}} t\{x/u\}$$

  and so it does not terminate.

- But each rule of the refinement is ***terminating*** and ***confluent*** when considered alone.

- The refinement is a ***non-terminating*** system which is ***locally terminating***.

- ***Main rewriting idea*** of the paper:

  To use ***local termination*** to define a ***new notion of development***.

Proof Technique 1

- For every term t there exist ***normal forms*** $\mathtt{sub}(t)$ and $\mathtt{dB}(t)$.
- The term $t^{\circ\circ} := \mathtt{sub}(\mathtt{dB}(t))$ is an ***abstract development***.
- The ***simple and elegant*** proof is based on ***local* confluence** and ***local* commutation** of $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{sub}}$.
- Moreover, $t^{\circ\circ}$ verifies property 3:

$$t \equiv_{\mathtt{P}} u \text{ implies } t^{\circ\circ} = u^{\circ\circ}$$

- So the permutative λ-calculus is ***confluent modulo*** $\equiv_{\mathtt{P}}$.

Proof Technique 2

- The proof of ***property 3*** ($t \equiv_{\mathrm{P}} u$ implies $t^{\circ\circ} = u^{\circ\circ}$) is also based on a ***local principle***.
- We define an equivalence $\equiv_{\Pi}$ on terms with explicit substitutions:
  1. $\equiv_{\mathrm{P}}$ is ***transported*** on $\equiv_{\Pi}$:

$$\begin{array}{ccc} t & \rightarrow & t' \\ \equiv_{\mathrm{P}} & & \\ u & & \end{array} \quad \text{implies } \exists u' \text{ s.t.} \quad \begin{array}{ccc} t & \rightarrow & t' \\ \equiv_{\mathrm{P}} & & \equiv_{\Pi} \\ u & \rightarrow & u' \end{array}$$

  2. $\equiv_{\Pi}$ is ***continuos*** with respect to reduction $\rightarrow$:

$$\begin{array}{ccc} t & \rightarrow & t' \\ \equiv_{\Pi} & & \\ u & & \end{array} \quad \text{implies } \exists u' \text{ s.t.} \quad \begin{array}{ccc} t & \rightarrow & t' \\ \equiv_{\Pi} & & \equiv_{\Pi} \\ u & \rightarrow & u' \end{array}$$

The equivalence $\equiv_{\Pi}$

- The equivalence $\equiv_{\Pi} := \equiv_{\Pi_\lambda} \cup \equiv_{\Pi_{[\cdot]}}$ is given by :

$$
\begin{array}{llll}
(\lambda x.\lambda y.t)\ u & \equiv_{\Pi_\lambda} & \lambda y.((\lambda x.t)\ u) & \text{if } y \notin \mathtt{fv}(u) \\
(\lambda x.t\ v)\ u & \equiv_{\Pi_\lambda} & (\lambda x.t)\ u\ v & \text{if } x \notin \mathtt{fv}(v) \\
(\lambda x.t\ v)\ u & \equiv_{\Pi_\lambda} & t\ ((\lambda x.v)\ u) & \text{if } x \notin \mathtt{fv}(t) \\
\\
t[x/s][y/v] & \equiv_{\Pi_{[\cdot]}} & t[y/v][x/s] & \text{if } x \notin \mathtt{fv}(v)\ \&\ y \notin \mathtt{fv}(s) \\
\lambda y.(t[x/s]) & \equiv_{\Pi_{[\cdot]}} & (\lambda y.t)[x/s] & \text{if } y \notin \mathtt{fv}(s) \\
t[x/s]\ v & \equiv_{\Pi_{[\cdot]}} & (t\ v)[x/s] & \text{if } x \notin \mathtt{fv}(v) \\
\\
t\ v[x/u] & \equiv_{\Pi_{[\cdot]}} & (t\ v)[x/u] & \text{if } x \notin \mathtt{fv}(t) \\
t[y/v][x/u] & \equiv_{\Pi_{[\cdot]}} & t[y/v[x/u]] & \text{if } x \notin \mathtt{fv}(t)
\end{array}
$$

- $\equiv_{\Pi_{[\cdot]}}$ is obtained by ***elimination of*** $\mathtt{dB}$***-redexes*** from $\equiv_{\mathtt{P}} = \equiv_{\Pi_\lambda}$.

Explaining the equivalence

- The second equation:

$$\begin{array}{ccc} (\lambda x.\lambda y.t)\ u & \equiv_{\mathtt{P}} & \lambda y.((\lambda x.t)\ u) \\ \downarrow_{\mathtt{dB}} & & \downarrow_{\mathtt{dB}} \\ (\lambda y.t)[x/u] & \equiv_{\Pi_{[\cdot]}} & \lambda y.(t[x/u]) \end{array}$$

- The third equation:

$$\begin{array}{ccc} ((\lambda x.t)\ u)v & \equiv_{\mathtt{P}} & (\lambda x.(t\ v))\ u \\ \downarrow_{\mathtt{dB}} & & \downarrow_{\mathtt{dB}} \\ t[x/u]\ v & \equiv_{\Pi_{[\cdot]}} & (t\ v)[x/u] \end{array}$$

- The first equation is obtained combining the previous two cases.

Explaining the equivalence

- The fourth equation:

$$\begin{array}{ccc} (\lambda x.t\ v)\ u & \equiv_{\mathtt{P}} & t\ ((\lambda x.v)\ u) \\ \downarrow_{\mathtt{dB}} & & \downarrow_{\mathtt{dB}} \\ (t\ v)[x/u] & \equiv_{\Pi_{[\cdot]}} & t\ v[x/u] \end{array}$$

- The fifth equation:

$$\begin{array}{ccc} ((\lambda y.t)\ v)[x/u] & \equiv_{\mathtt{P}} & (\lambda y.t)\ v[x/u] \\ \downarrow_{\mathtt{dB}} & & \downarrow_{\mathtt{dB}} \\ t[y/v][x/u] & \equiv_{\Pi_{[\cdot]}} & t[y/v[x/u]] \end{array}$$

Idea of the proof

- Taking the $\mathtt{dB}$-***normal form*** maps $\equiv_\sigma$ on $\equiv_{\Pi_{[\cdot]}}$.
- Taking the $\mathtt{sub}$-***normal form*** $\equiv_{\Pi_{[\cdot]}}$ ***disappear***.
- For instance:

$$\begin{array}{ccc} t[y/v][x/u] & \equiv_{\Pi_{[\cdot]}} & t[y/v[x/u]] \\ \downarrow_{\mathtt{sub}} & & \downarrow_{\mathtt{sub}} \\ \downarrow_{\mathtt{sub}} & & \downarrow_{\mathtt{sub}} \\ t\{y/v\}\{x/u\} & = & t\{y/v\{x/u\}\} \end{array}$$

- Thus $t \equiv_{\mathtt{P}} u$ implies $\mathtt{dB}(\mathtt{sub}(t)) = \mathtt{dB}(\mathtt{sub}(u))$.
- The technique also proves ***confluence of ES modulo*** $\equiv_\Pi$.
- Actually, in both cases proves ***Church-Rosser modulo*** (stronger).

Outline

PSN

- PSN is a ***conditional*** termination property:

  if t is β-strongly normalizing ***then*** t is strongly normalizing modulo $\equiv_{\mathtt{P}}$.

- Usually, PSN is ***difficult*** to prove.
- Our proof technique:
  - ***Reduce*** PSN for the permutative λ-calculus to ***ES modulo*** $\equiv_{\Pi_{[\cdot]}}$.
  - Done ***via a*** $\mathtt{dB}$***-projection***, showed to preserve SN.
  - ***PSN for ES modulo*** $\equiv_{\Pi_{[\cdot]}}$ is a recent, non-trivial result of ours (LMCS).

Difficulty 1

- These two equations are ***problematic***:

$$t\ v[x/u] \equiv_{\Pi_{[.]}} (t\ v)[x/u] \quad \text{if } x \notin \mathtt{fv}(t)$$

$$t[y/v][x/u] \equiv_{\Pi_{[.]}} t[y/v[x/u]] \quad \text{if } x \notin \mathtt{fv}(t)$$

- They are ***not*** a ***strong bisimulation***:

$$\begin{array}{ccccc} (y\ y)[y/x][x/z] & \rightarrow_{\mathtt{sub}} & (x\ x)[x/z] & \rightarrow_{\mathtt{sub}} & z\ z \\ \equiv_{\Pi_{[.]}} & & \not\equiv_{\Pi_{[.]}} & & = \\ (y\ y)[y/x[x/z]] & \rightarrow_{\mathtt{sub}} & x[x/z]\ x[x/z] & \rightarrow_{\mathtt{sub}}\rightarrow_{\mathtt{sub}} & z\ z \end{array}$$

Difficulty 2

- We cheated a bit, ***PSN does not hold***.
- Let $u = (zz)[z/y]$, then:

$$\begin{array}{rcllll} t & = & u[x/u] = (zz)[z/y][x/u] & \equiv_{\Pi_{[\cdot]}} & (zz)[z/y[x/u]] & \to_{\mathtt{c}} \\ & & (z_1 z_2)[z_1/y[x/u]][z_2/y[x/u]] & \to^{+}_{\mathtt{d}} & y[x/u](y[x/u]) & \equiv_{\Pi_{[\cdot]}} \\ & & (yy)[x_1/u][x/u] & \equiv_{\Pi_{[\cdot]}} & (yy)[x_1/u[x/u]] & \end{array}$$

- The term t reduces to a term ***containing*** t.
- ***Loop*** of the form $t \to^{+} C_0[t] \to^{+} C_0[C_1[t]] \to^{+} \ldots$.
- $t_0 = (\lambda x.((\lambda z.z\ z)\ y))\ ((\lambda z.z\ z)\ y)$ is SN in λ-calculus but it reduces to $t \notin \mathtt{SN}_{\lambda\mathtt{j}}$.

Refining the equations

- So the equations have to be refined:

$$t\ v[x/u] \equiv_{\Pi_{[\cdot]}} (t\ v)[x/u] \quad \text{if } x \notin \mathtt{fv}(t)\ \&\ x \in \mathtt{fv}(v)$$

$$t[y/v][x/u] \equiv_{\Pi_{[\cdot]}} t[y/v[x/u]] \quad \text{if } x \notin \mathtt{fv}(t)\ \&\ x \in \mathtt{fv}(v)$$

- For this system ***PSN holds***.
- The following rule can be added ***without breaking PSN***:

$$t[y/v[x/u]] \rightarrow t[y/v][x/u] \quad \text{if } x \notin \mathtt{fv}(t)$$

- For the other direction ***we do not know***.

Refining the permutative λ-calculus

- The permutative λ-calculus ***suffers*** from the same problem.
- The calculus actually is:

$$
\begin{array}{llll}
(\lambda x.t)\ u & \rightarrow_{\beta} & t\{x/u\} & \\
 & & & \\
(\lambda x.\lambda y.t)\ u & \equiv_{\mathrm{P}} & \lambda y.((\lambda x.t)\ u) & \text{if } y \notin \mathtt{fv}(u) \\
(\lambda x.t\ v)\ u & \equiv_{\mathrm{P}} & (\lambda x.t)\ u\ v & \text{if } x \notin \mathtt{fv}(v) \\
 & & & \\
t\ ((\lambda x.v)\ u) & \equiv_{\mathrm{P}} & (\lambda x.t\ v)\ u & \text{if } x \notin \mathtt{fv}(t)\ \&\ x \in \mathtt{fv}(v) \\
 & & & \\
t\ ((\lambda x.v)\ u) & \rightarrow_{\mathrm{u}} & (\lambda x.t\ v)\ u & \text{if } x \notin \mathtt{fv}(t)
\end{array}
$$

- The confluence proof ***still works***.

Additional comments

- The PSN result for the ***structural*** λ***-calculus*** is ***hard***.
- For the ***linear substitutions calculus***, thanks to better diagrams (*i.e.* residual property) it becomes ***much easier***.
- Some equivalences may be ***oriented***, and the results ***still holds***.
- The proof of confluence is essentially ***unchanged***.
- There is a ***core*** at a ***distance*** sub-calculus computing normal forms.
- So the equations can be oriented form ***left to right or right to left***, indifferently.

Outline

Conclusions

- An extension of λ-calculus with ***equations*** permuting constructors, generalizing all previous calculi in the literature.
- Generality obtained via ***rewriting modulo***.
- ***Difficult confluence problem*** solved in a simple way using an elementary calculus with ***explicit substitutions***.
- The refinement:

λ-calculus	$\Rightarrow$	***explicit substitutions***
non-termination	$\Rightarrow$	***local termination***

- Then ***confluence modulo*** reduces to ***local properties***.

THANKS!