The rewriting theory of explicit substitution at a distance

Beniamino Accattoli

Ecole Polytechnique, LIX

Outline

Outline

Discriminating ES calculi

- ***Many*** calculi of explicit substitutions (ES).
- How to ***discriminate***?
- Denotational and categorical semantics describe ***normal forms***.
- Explicit substitutions can always be ***executed***, getting a λ-term.
- Normal forms thus are λ-terms ***without ES***.
- Denotational and categorical semantics ***cannot help***.

Discriminating ES calculi

- Explicit substitutions are a ***purely operational*** topic.
- Our discrimination criterion: ***logic background*** and ***quality of the rewriting theory***.
- ***Logic background***: ***Linear Logic Proof-Nets*** (previous talk).
- ***Quality of the rewriting theory***: ***properties***, ***insights*** and ***compactness*** of the proofs.
- ***Challenge***: match the ***beauty*** of λ-calculus rewriting theory.
- ***Faith***: beauty will induce a ***powerful theory***.

Outline

1. Introduction
2. Confluence
3. Refining the calculus
4. Other properties
5. Developments

Definition

- 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).
- λ-calculus and calculi with ES do ***not terminate***.

Parallel reductions

- Confluence for non-terminating calculi often obtained via ***parallel reduction*** (Tait-Martin-Löf).
- ***Idea***: find a new reduction $\Rightarrow$ s.t.:
  - ***Extends*** $\rightarrow$: $\rightarrow \subseteq \Rightarrow \subseteq \rightarrow^*$.
  - ***Is parallel*** (=diamond property=strong confluence):

$$\begin{array}{ccc} t & \Rightarrow & u_1 \\ \Downarrow & & \\ u_2 & & \end{array} \quad \text{implies } \exists v \text{ s.t.} \quad \begin{array}{ccc} t & \Rightarrow & u_1 \\ \Downarrow & & \Downarrow \\ u_2 & \Rightarrow & v \end{array}$$

- Parallelism implies $\Rightarrow$ and $\Rightarrow^*$ are confluent.
- By 1) $\Rightarrow^* = \rightarrow^*$
- So $\rightarrow$ is confluent.

Residuals

Residuals are a sort of ***refinement*** of parallel reduction.

The refinement consist in:

1. Adding a ***tracing system*** for redexes.
2. Asking that the redexes reduced to close the diagram can be traced back to the ***starting term***:

$$\begin{array}{ccc} t & \Rightarrow_R & u_1 \\ \Downarrow_S & & \\ u_2 & & \end{array} \quad \text{implies } \exists v, R/S, S/R \text{ s.t.} \quad \begin{array}{ccc} t & \Rightarrow_R & u_1 \\ \Downarrow_S & & \Downarrow_{R/S} \\ u_2 & \Rightarrow_{S/R} & v \end{array}$$

S/R is the ***set*** of redexes which are residuals of S ***after*** R.

Examples in λ-calculus

- Singleton set:

$$\begin{array}{ccc} (I\,I)\,(I\,I) & \Rightarrow_R & (I\,I)\,I \\ \Downarrow_S & & \Downarrow_{R/S} \\ I\,(I\,I) & \Rightarrow_{R/S} & I\,I \end{array}$$

- Set:

$$\begin{array}{ccc} (\lambda x.xx)\,(I\,I) & \Rightarrow_R & (\lambda x.xx)\,I \\ \Downarrow_S & & \Downarrow_{R/S} \\ (I\,I)\,(I\,I) & \Rightarrow_{R/S} & I\,I \end{array}$$

- Empty Set:

$$\begin{array}{ccc} (\lambda x.y)\,(I\,I) & \Rightarrow_R & (\lambda x.y)\,I \\ \Downarrow_S & & \Downarrow_{R/S} \\ y & \Rightarrow_{R/S} & y \end{array}$$

Residuals

- The residual property implies ***confluence*** (it induces a parallel reduction).
- The ***advanced rewriting theory*** of λ-calculus (standardization, families, optimality) is based on residuals.
- Residuals are the right ***semantic*** abstraction of being ***orthogonal***.
- ***Traditionally***: a system is orthogonal if it is ***left-linear*** and it has ***no critical pair***.
- This is a ***syntactic*** definition.
- ***Orthogonality*** $\Rightarrow$ ***residual property***, which is why orthogonal systems are interesting.
- But there are systems with residuals which are ***not orthogonal***.

The structural λ-calculus λj

- Rules:

$$
\begin{array}{llll}
(\lambda x.t)L\, u & \rightarrow_{\texttt{B-distance}} & t[x/u]L & \\
t[x/u] & \rightarrow_{\texttt{weakening}} & t & |t|_x = 0 \\
t[x/u] & \rightarrow_{\texttt{dereliction}} & t\{x/u\} & |t|_x = 1 \\
t[x/u] & \rightarrow_{\texttt{contraction}} & t_{[y]_x}[x/u][y/u] & |t|_x > 1 \text{ \& } y \text{ fresh}
\end{array}
$$

- λj ***does not enjoy the residual property***.

No residuals for $\lambda\mathtt{j}$ 1

- Consider:

$$\begin{array}{ccccc} x[y/w] & {}_{\mathtt{w}}\leftarrow & x[z/y\ y][y/w] & \rightarrow_{\mathtt{c}} & x[z/y_1\ y_2][y_1/w][y_2/w] \\ \downarrow_{\mathtt{w}} & & & & \downarrow_{\mathtt{w}} \\ x & & & & x[y_1/w][y_2/w] \end{array}$$

- The diagram ***can be closed***:

$$\begin{array}{ccccc} x[y/w] & {}_{\mathtt{w}}\leftarrow & x[z/y\ y][y/w] & \rightarrow_{\mathtt{c}} & x[z/y_1\ y_2][y_1/w][y_2/w] \\ \downarrow_{\mathtt{w}} & & & & \downarrow_{\mathtt{w}} \\ x & {}_{\mathtt{w}}\leftarrow & x[y_1/w] & {}_{\mathtt{w}}\leftarrow & x[y_1/w][y_2/w] \end{array}$$

But the two further steps reduce ***created redexes***.

No residuals for λj 2

- Consider:

$$\begin{array}{ccccc} (xx)[x/z] & {}_{\mathtt{d}}\leftarrow & (xx)[x/y][y/z] & \rightarrow_{\mathtt{c}} & (x_1x_2)[x_1/y][x_2/y][y/z] \\ \downarrow_{\mathtt{c}} & & & & \downarrow_{\mathtt{c}} \\ (x_1x_2)[x_1/z][x_2/z] & & & & (x_1x_2)[x_1/y_1][x_2/y_2][y_1/z][y_2/z] \end{array}$$

- The diagram ***can be closed***:

$$\begin{array}{ccccc} (xx)[x/z] & {}_{\mathtt{d}}\leftarrow & (xx)[x/y][y/z] & \rightarrow_{\mathtt{c}} & (x_1x_2)[x_1/y][x_2/y][y/z] \\ \downarrow_{\mathtt{c}} & & & & \downarrow_{\mathtt{c}} \\ (x_1x_2)[x_1/z][x_2/z] & {}_{\mathtt{d}}\leftarrow & (x_1x_2)[x_1/y_1][x_2/z][y_1/z] & {}_{\mathtt{d}}\leftarrow & (x_1x_2)[x_1/y_1][x_2/y_2][y_1/z][y_2 \end{array}$$

But the two further steps reduce ***created redexes***.

Outline

Milner's calculus

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

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

Is a ***mix*** of $\lambda\mathtt{j}$ and ***Milner's calculus***.

- ***It enjoys residuals***.

Residuals for $\lambda_{\mathtt{ls}}$

- The first critical pair:

$$\begin{array}{ccc} x[z/y\ y][y/w] & \rightarrow_{\mathtt{ls}} & x[z/w\ y][y/w] \\ \downarrow_{\mathtt{w}} & & \downarrow_{\mathtt{w}} \\ x[y/w] & = & x[y/w] \end{array}$$

- The second one:

$$\begin{array}{ccccc} (xx)[x/y][y/z] & & \rightarrow_{\mathtt{ls}} & & (xx)[x/z][y/z] \\ \downarrow_{\mathtt{ls}} & & & & \downarrow_{\mathtt{ls}} \\ (yx)[x/y][y/z] & \rightarrow_{\mathtt{ls}} & (zx)[x/z][y/z] & \rightarrow_{\mathtt{ls}} & (zx)[x/z][y/z] \end{array}$$

Outline

Postponement of erasing steps

- In λ-calculus it ***is not possible to postpone erasing steps***:

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

- In $\lambda_{\mathtt{ls}}$ instead the postponement ***holds***.
- ***w-postponement***: $t \rightarrow^* u$ then $t \rightarrow^*_{\neg w}\rightarrow^*_{w} u$.
- $\lambda_{\mathtt{ls}}$ generalizes Klop's memory calculus.

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_\beta$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \to^*_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda \mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_{\beta}$, then $t \in SN_{\lambda \mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^{*}_{\lambda \mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda\mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_\beta$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^*_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda\mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_{\beta}$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^{*}_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda \mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_{\beta}$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^{*}_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda\mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_\beta$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^*_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda\mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_\beta$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^*_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda \mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_\beta$, then $t \in SN_{\lambda \mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^*_{\lambda \mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

Properties of $\lambda\mathtt{j}$

- ***Simulation of one-step β-reduction***.
- ***Strong Normalisation in the typed case***.
- ***Preservation of β-strong normalisation (PSN)***:
  if $t \in SN_\beta$, then $t \in SN_{\lambda\mathtt{j}}$.
  Melliès counter-example out.
  Short proof!
- ***Full Composition***:
  $t[x/u] \rightarrow^*_{\lambda\mathtt{j}} t\{x/u\}$.
  Without equations!
- ***Confluence***.
- ***Meta-Confluence*** (Fabien Renaud, Kesner's student).

$\equiv_{\circ}$-equivalence

- The translation on graphs induces a quotient:

$$
\begin{array}{rcll}
(\lambda y.t)[u/x] & \equiv & \lambda y.(t[u/x]) & \text{if } y \notin \mathtt{fv}(u) \\
(t[u/x])\ v & \equiv & (t\ v)[u/x] & \text{if } x \notin \mathtt{fv}(v) \\
t[x/u][y/v] & \equiv & t[y/v][x/u] & \text{if } y \notin \mathtt{fv}(u)\ \&\ x \notin \mathtt{fv}(v)
\end{array}
$$

- Which is a strong bisimulation by construction:

$$
\begin{array}{ccc}
t & \to & t' \\
\downarrow^{\cdot} & & \downarrow^{\cdot} \\
G & \to & G' \\
\uparrow^{\cdot} & & \uparrow^{\cdot} \\
s & \to & s'
\end{array}
$$

$\equiv_\circ$-equivalence

- The translation on graphs induces a quotient:

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

- Which is a strong bisimulation by construction:

$$\begin{array}{ccc}
t & \rightarrow & t' \\
\downarrow\cdot^{-} & & \downarrow\cdot^{-} \\
G & \rightarrow & G' \\
\uparrow\cdot^{-} & & \uparrow\cdot^{-} \\
s & \rightarrow & s'
\end{array}$$

$\equiv_{\circ}$-equivalence

- The translation on graphs induces a quotient:

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

- Which is a strong bisimulation by construction:

$$
\begin{array}{ccc}
t & \rightarrow & t' \\
\downarrow_{\bar{\cdot}} & & \downarrow_{\bar{\cdot}} \\
G & \rightarrow & G' \\
\uparrow_{\bar{\cdot}} & & \uparrow_{\bar{\cdot}} \\
s & \rightarrow & s'
\end{array}
$$

$\equiv_\circ$-equivalence

- $\equiv_\circ$ is a reformulation of ***Regnier's σ-equivalence***.
- But $\equiv_\circ$ is a strong bisimulation whether σ ***is not***.
- Strong bisimulations ***preserve reduction lengths***.
- $\Rightarrow \lambda\mathtt{j}$ and $\lambda_{\mathtt{ls}}$ modulo $\equiv_\circ$ enjoy ***PSN***.
- ***Church-Rosser modulo*** also follows.

$\equiv_\circ$-equivalence

- $\equiv_\circ$ is a reformulation of ***Regnier's σ-equivalence***.
- But $\equiv_\circ$ is a strong bisimulation whether σ ***is not***.
- Strong bisimulations ***preserve reduction lengths***.
- $\Rightarrow \lambda \mathtt{j}$ and $\lambda_{\mathtt{ls}}$ modulo $\equiv_\circ$ enjoy ***PSN***.
- ***Church-Rosser modulo*** also follows.

$\equiv_\circ$-equivalence

- $\equiv_\circ$ is a reformulation of ***Regnier's σ-equivalence***.
- But $\equiv_\circ$ is a strong bisimulation whether σ ***is not***.
- Strong bisimulations ***preserve reduction lengths***.
- $\Rightarrow \lambda\mathtt{j}$ and $\lambda_{\mathtt{ls}}$ modulo $\equiv_\circ$ enjoy ***PSN***.
- ***Church-Rosser modulo*** also follows.

$\equiv_\circ$-equivalence

- $\equiv_\circ$ is a reformulation of ***Regnier's σ-equivalence***.
- But $\equiv_\circ$ is a strong bisimulation whether σ ***is not***.
- Strong bisimulations ***preserve reduction lengths***.
- $\Rightarrow \lambda\mathtt{j}$ and $\lambda_{\mathtt{ls}}$ modulo $\equiv_\circ$ enjoy ***PSN***.
- ***Church-Rosser modulo*** also follows.

$\equiv_{\circ}$-equivalence

- $\equiv_{\circ}$ is a reformulation of ***Regnier's σ-equivalence***.
- But $\equiv_{\circ}$ is a strong bisimulation whether σ ***is not***.
- Strong bisimulations ***preserve reduction lengths***.
- $\Rightarrow \lambda\mathtt{j}$ and $\lambda_{\mathtt{ls}}$ modulo $\equiv_{\circ}$ enjoy ***PSN***.
- ***Church-Rosser modulo*** also follows.

Composition

- In $\lambda\mathtt{j}$ there is ***no rule*** for ***composing substitutions***:

$$t\,[y/v]\,[x/u] \not\to_{comp} t\,[x/u]\,[y/v[x/u]]$$

- There is a notion of ***implicit*** composition:

$$t\,[y/v\{x/u\}][x/u]$$

Which can be computed, ***at a distance***, in $\lambda\mathtt{j}$.

- For instance:

$$\begin{array}{lll} (x\ y)[y/x][x/u] & \to_{\mathrm{c}} & (x_1\ y)[y/x_2][x_1/u][x_2/u] \quad \to_{\mathrm{d}} \\ & & (x_1\ y)[y/u][x_1/u] \quad =_{\alpha} \\ & & (x\ y)[y/u][x/u] \end{array}$$

Composition

- In $\lambda\mathtt{j}$ there is ***no rule*** for ***composing substitutions***:

$$t\,[y/v]\,[x/u] \not\to_{comp} t\,[x/u]\,[y/v[x/u]]$$

- There is a notion of ***implicit*** composition:

$$t\,[y/v\{x/u\}][x/u]$$

  Which can be computed, ***at a distance***, in $\lambda\mathtt{j}$.

- For instance:

$$(x\ y)[y/x][x/u] \quad \to_{\mathrm{c}} \quad (x_1\ y)[y/x_2][x_1/u][x_2/u] \quad \to_{\mathrm{d}}$$
$$(x_1\ y)[y/u][x_1/u] \quad =_{\alpha}$$
$$(x\ y)[y/u][x/u]$$

Composition

- In $\lambda \mathtt{j}$ there is ***no rule*** for ***composing substitutions***:

$$t\,[y/v]\,[x/u] \not\to_{comp} t\,[x/u]\,[y/v[x/u]]$$

- There is a notion of ***implicit*** composition:

$$t\,[y/v\{x/u\}][x/u]$$

Which can be computed, ***at a distance***, in $\lambda \mathtt{j}$.

- For instance:

$$\begin{array}{lll} (x\ y)[y/x][x/u] & \to_{\mathrm{c}} & (x_1\ y)[y/x_2][x_1/u][x_2/u] & \to_{\mathrm{d}} \\ & & (x_1\ y)[y/u][x_1/u] & =_{\alpha} \\ & & (x\ y)[y/u][x/u] & \end{array}$$

Composition

- In $\lambda\mathtt{j}$ there is ***no rule*** for ***composing substitutions***:

$$t\,[y/v]\,[x/u] \not\to_{comp} t\,[x/u]\,[y/v[x/u]]$$

- There is a notion of ***implicit*** composition:

$$t\,[y/v\{x/u\}][x/u]$$

Which can be computed, ***at a distance***, in $\lambda\mathtt{j}$.

- For instance:

$$\begin{array}{lll}(x\ y)[y/x][x/u] & \to_{\mathrm{c}} & (x_1\ y)[y/x_2][x_1/u][x_2/u] & \to_{\mathrm{d}} \\ & & (x_1\ y)[y/u][x_1/u] & =_\alpha \\ & & (x\ y)[y/u][x/u] & \end{array}$$

Composition

- In $\lambda\mathtt{j}$ there is ***no rule*** for ***composing substitutions***:

$$t\,[y/v]\,[x/u] \not\to_{comp} t\,[x/u]\,[y/v[x/u]]$$

- There is a notion of ***implicit*** composition:

$$t\,[y/v\{x/u\}][x/u]$$

  Which can be computed, ***at a distance***, in $\lambda\mathtt{j}$.

- For instance:

$$\begin{aligned}(x\ y)[y/x][x/u] &\to_{\mathrm{c}} (x_1\ y)[y/x_2][x_1/u][x_2/u] &\to_{\mathrm{d}}\\ &(x_1\ y)[y/u][x_1/u] &=_{\alpha}\\ &(x\ y)[y/u][x/u]\end{aligned}$$

Composition

- In $\lambda \mathtt{j}$ there is ***no rule*** for ***composing substitutions***:

$$t\,[y/v]\,[x/u] \not\to_{comp} t\,[x/u]\,[y/v[x/u]]$$

- There is a notion of ***implicit*** composition:

$$t\,[y/v\{x/u\}][x/u]$$

  Which can be computed, ***at a distance***, in $\lambda \mathtt{j}$.

- For instance:

$$\begin{array}{lll} (x\ y)[y/x][x/u] & \to_{\mathtt{c}} & (x_1\ y)[y/x_2][x_1/u][x_2/u] & \to_{\mathtt{d}} \\ & & (x_1\ y)[y/u][x_1/u] & =_{\alpha} \\ & & (x\ y)[y/u][x/u] & \end{array}$$

Outline

Complete Developments

- A ***complete development*** from a term t is a reduction sequence in which all and only residuals of redexes that already exist in t are contracted.
- Complete developments are terminating (and confluent).
- The result of complete developments can be defined by induction on the term:

$$
\begin{array}{lcll}
\texttt{Dev}(x) & := & x & \\
\texttt{Dev}(\lambda x.t) & := & \lambda x.\texttt{Dev}(t) & \\
\texttt{Dev}((\lambda x.t)\ u) & := & \texttt{Dev}(t)\{x/\texttt{Dev}(u)\} & \\
\texttt{Dev}(t\ u) & := & \texttt{Dev}(t)\ \texttt{Dev}(u) & \text{if } t \neq \lambda
\end{array}
$$

Extending complete developments

- Creation of redexes in λ-calculus (Levy):

$$1)\quad ((\lambda x.\lambda y.t)u)\ v \quad \to_\beta \quad (\lambda y.t\{x/u\})\ v$$

$$2)\quad (\lambda x.x)(\lambda y.t)\ u \quad \to_\beta \quad (\lambda y.t)\ u$$

$$3)\quad (\lambda x.C[x\ v])\ (\lambda y.u) \quad \to_\beta \quad C\{x/\lambda y.u\}[(\lambda y.u)\ v]$$

- 1) Creates a redex that was hidden by a λ.
- 2) The redex was hidden by an identity redex.
- 3) It is the dangerous kind of creation: the one leading to divergence.
- δ δ creates only redexes of the third kind.

Superdevelopments

- There exists an extension of complete developments which reduces redexes of type 1 and 2:

$$1)\quad ((\lambda x.\lambda y.t)u)\ v \quad \to_\beta \quad (\lambda y.t\{x/u\})\ v$$

$$2)\quad (\lambda x.x)(\lambda y.t)\ u \quad \to_\beta \quad (\lambda y.t)\ u$$

- These superdevelopments are convergent and can be defined by induction, too:

$$\begin{array}{lcll} x^{\circ\circ} & := & x & \\ (\lambda x.t)^{\circ\circ} & := & \lambda x.t^{\circ\circ} & \\ t\ u^{\circ\circ} & := & t^{\circ\circ}\ u^{\circ\circ} & \text{if } t^{\circ\circ} \neq \lambda \\ t\ u^{\circ\circ} & := & t_1\{x/u^{\circ\circ}\} & \text{if } t^{\circ\circ} = \lambda x.t_1 \end{array}$$

Developments

- Developments and Superdevelopments can be characterized in new ways in $\lambda_{\mathtt{ls}}$ and $\lambda\mathtt{j}$.
- The idea is that a (Super)development can be seen as the ***normal form*** of some subreductions of $\lambda_{\mathtt{ls}}$ or $\lambda\mathtt{j}$.
- But two ***new notions*** of developments can also be defined.
- One reducing only creations of ***type 1***.
- One reducing creations of type 1, 2 and a ***linear case of type 3***.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
  - ***Simple***: 3 rules;
  - ***Solid***: propagations can be modularly added;
  - ***Expressive***: head linear reduction, developments;
  - ***Perfect rewriting theory***: residuals, short PSN proof.
  - ***Logiacl foundation***: inspired by Linear Logic,
  - ***Graphical syntax***: Proof-Nets.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
  - ***Simple***: 3 rules;
  - ***Solid***: propagations can be modularly added;
  - ***Expressive***: head linear reduction, developments;
  - ***Perfect rewriting theory***: residuals, short PSN proof.
  - ***Logiacl foundation***: inspired by Linear Logic,
  - ***Graphical syntax***: Proof-Nets.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
  - ***Simple***: 3 rules;
  - ***Solid***: propagations can be modularly added;
  - ***Expressive***: head linear reduction, developments;
  - ***Perfect rewriting theory***: residuals, short PSN proof.
  - ***Logiacl foundation***: inspired by Linear Logic,
  - ***Graphical syntax***: Proof-Nets.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
  - ***Simple***: 3 rules;
  - ***Solid***: propagations can be modularly added;
  - ***Expressive***: head linear reduction, developments;
  - ***Perfect rewriting theory***: residuals, short PSN proof.
  - ***Logiacl foundation***: inspired by Linear Logic,
  - ***Graphical syntax***: Proof-Nets.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
  - ***Simple***: 3 rules;
  - ***Solid***: propagations can be modularly added;
  - ***Expressive***: head linear reduction, developments;
  - ***Perfect rewriting theory***: residuals, short PSN proof.
  - ***Logiacl foundation***: inspired by Linear Logic,
  - ***Graphical syntax***: Proof-Nets.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
    - ***Simple***: 3 rules;
    - ***Solid***: propagations can be modularly added;
    - ***Expressive***: head linear reduction, developments;
    - ***Perfect rewriting theory***: residuals, short PSN proof.
    - ***Logiacl foundation***: inspired by Linear Logic,
    - ***Graphical syntax***: Proof-Nets.

Conclusions

- The linear substitution calculus is the ***best*** refinement of λ-calculus ***I know of***:
  - ***Simple***: 3 rules;
  - ***Solid***: propagations can be modularly added;
  - ***Expressive***: head linear reduction, developments;
  - ***Perfect rewriting theory***: residuals, short PSN proof.
  - ***Logiacl foundation***: inspired by Linear Logic,
  - ***Graphical syntax***: Proof-Nets.