Matching modulo superdevelopments
Application to second-order matching

Germain Faure

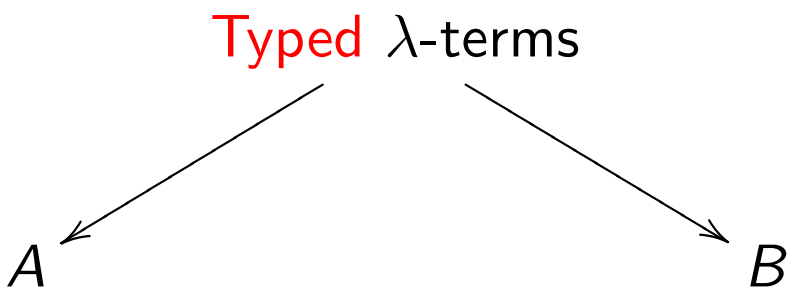
Ws Rho'06

Higher-order matching

- Usually defined as the following problem:

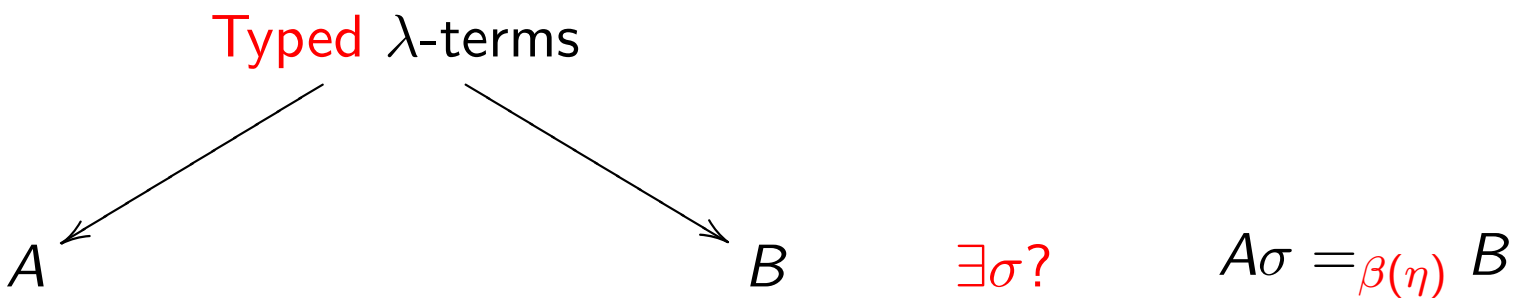
Higher-order matching

- Usually defined as the following problem:



Higher-order matching

- Usually defined as the following problem:

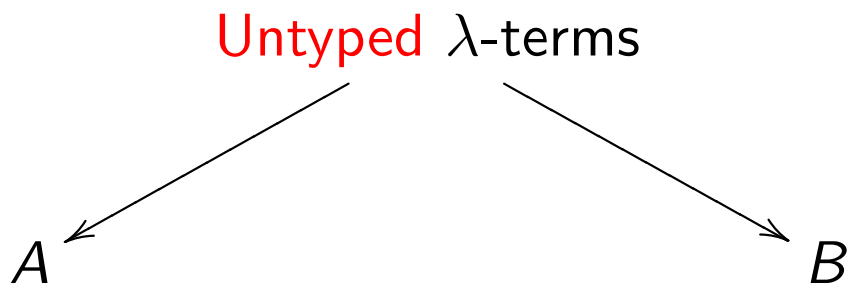


Higher-order matching (revised) [?]

- Another approach:

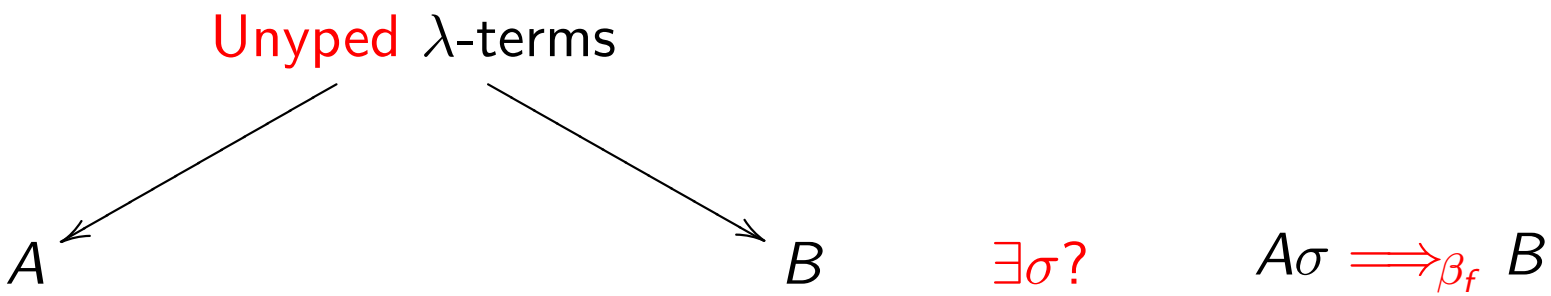
Higher-order matching (revised) [?]

- Another approach:



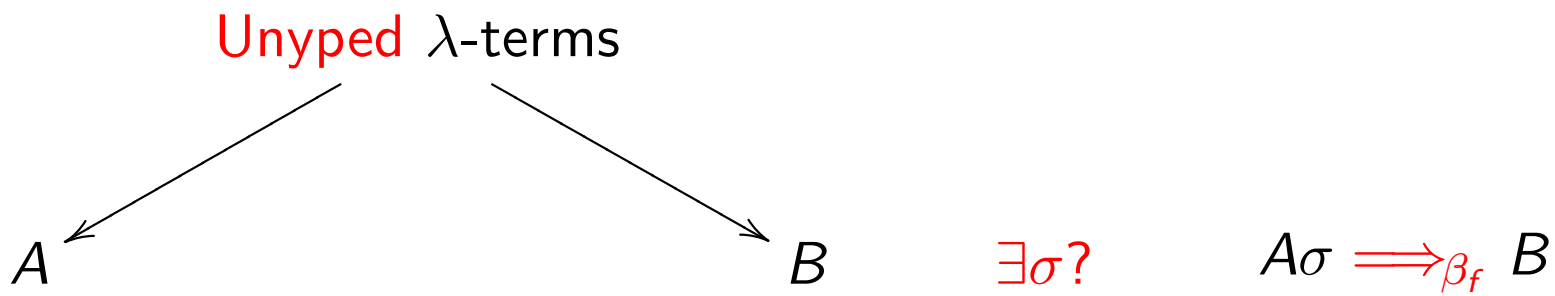
Higher-order matching (revised) [?]

- An other approach:



Higher-order matching (revised) [?]

- An other approach:

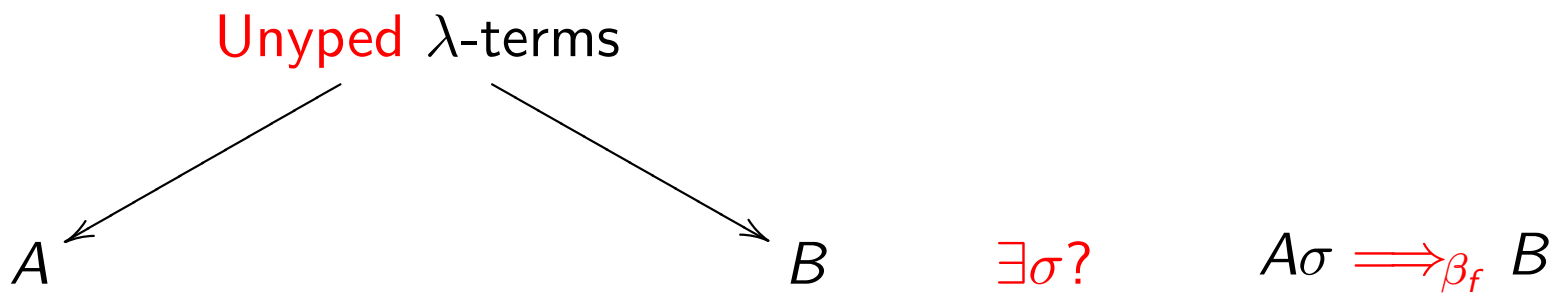


where $\Longrightarrow_{\beta_f}$ is a one-step reduction for an appropriate parallel reduction:

  - that always terminates

Higher-order matching (revised) [?]

- An other approach:



where $\Longrightarrow_{\beta_f}$ is a one-step reduction for an appropriate parallel reduction:

  - that always terminates
  - that gives an approximation of β-normal form

Motivations

- First introduced for automatic program transformation

Motivations

- First introduced for automatic program transformation
- It may be difficult to find a suitable type system s.t. equivalence on typed terms is decidable

Motivations

- First introduced for automatic program transformation
- It may be difficult to find a suitable type system s.t. equivalence on typed terms is decidable

  ➥ Example: Pattern-matching in the ρ-calculus

Higher-order matching in the ρ-calculus

1. The ρ-calculus

Higher-order matching in the ρ-calculus

1. The ρ-calculus
   - was introduced [**?**] to make all the ingredients of rewriting such as rule applications and results explicit

Higher-order matching in the ρ-calculus

1. The ρ-calculus
   - was introduced [**?**] to make all the ingredients of rewriting such as rule applications and results explicit
   - is *in fine* an extension of the λ-calculus with built-in pattern-matching and term collections

Higher-order matching in the ρ-calculus

1. The ρ-calculus
   - was introduced [**?**] to make all the ingredients of rewriting such as rule applications and results explicit
   - is *in fine* an extension of the λ-calculus with built-in pattern-matching and term collections
2. Higher-order matching in the ρ-calculus for

Higher-order matching in the ρ-calculus

1. The ρ-calculus
   - was introduced [**?**] to make all the ingredients of rewriting such as rule applications and results explicit
   - is *in fine* an extension of the λ-calculus with built-in pattern-matching and term collections
2. Higher-order matching in the ρ-calculus for
   - Transformations of programs with built-in pattern-matching (not only purely functional programs)

Higher-order matching in the ρ-calculus

1. The ρ-calculus
   - was introduced [**?**] to make all the ingredients of rewriting such as rule applications and results explicit
   - is *in fine* an extension of the λ-calculus with built-in pattern-matching and term collections
2. Higher-order matching in the ρ-calculus for
   - Transformations of programs with built-in pattern-matching (not only purely functional programs)
   - Proof theory that handles rich proof-terms in the generalized deduction modulo [**?**]

Some questions - Contributions

1. What is this appropriate parallel reduction?

Some questions - Contributions

1. What is this appropriate parallel reduction?
   - "It may be a little difficult to understand" [**?**]

Some questions - Contributions

1. What is this appropriate parallel reduction?
   - "It may be a little difficult to understand" [**?**]
   - Why is it a relevant approximation of β-normal forms?

Some questions - Contributions

1. What is this appropriate parallel reduction?
   - "It may be a little difficult to understand" [**?**]
   - Why is it a relevant approximation of β-normal forms?
2. What is the link with other h.o. matching algorithms?

Some questions - Contributions

1. What is this appropriate parallel reduction?
   - "It may be a little difficult to understand" [**?**]
   - Why is it a relevant approximation of β-normal forms?
2. What is the link with other h.o. matching algorithms?
   - Second-order matching

Some questions - Contributions

1. What is this appropriate parallel reduction?
   - "It may be a little difficult to understand" [**?**]
   - Why is it a relevant approximation of β-normal forms?
2. What is the link with other h.o. matching algorithms?
   - Second-order matching
   - H.o. matching of patterns *à la* Miller

Some questions - Contributions

1. What is this appropriate parallel reduction?
   - "It may be a little difficult to understand" [**?**]
   - Why is it a relevant approximation of β-normal forms?
2. What is the link with other h.o. matching algorithms?
   - Second-order matching
   - H.o. matching of patterns *à la* Miller

➥ Intuition
Deeper understanding

Road-map

1. Superdevelopments. Strong parallel β-reduction.
2. Matching modulo SD. Links with other h.o. matching algorithms.
3. Algorithm for matching modulo SD.
4. Second-order matching.
5. The η-equivalence.

The λ-calculus

- Syntax

A, B, C	$::=$	x	variable
	$\mid$	X	matching variable
	$\mid$	c	constant
	$\mid$	$\lambda x.A$	abstraction
	$\mid$	AB	application

The λ-calculus

- Syntax

$$
\begin{array}{rcll}
A, B, C & ::= & x & \text{variable} \\
& | & X & \text{matching variable} \\
& | & c & \text{constant} \\
& | & \lambda x.A & \text{abstraction} \\
& | & AB & \text{application}
\end{array}
$$

We note $f(A_1, \ldots, A_n)$ for $(\ldots(fA_1)\,)A_n)$

The λ-calculus

- Syntax

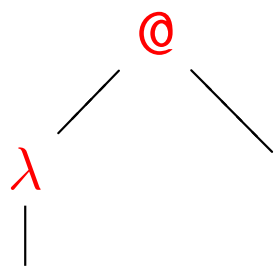
$$
\begin{array}{rcll}
A, B, C & ::= & x & \text{variable} \\
& | & X & \text{matching variable} \\
& | & c & \text{constant} \\
& | & \lambda x.A & \text{abstraction} \\
& | & AB & \text{application}
\end{array}
$$

We note $f(A_1, \ldots, A_n)$ for $(\ldots(fA_1)\,)A_n)$

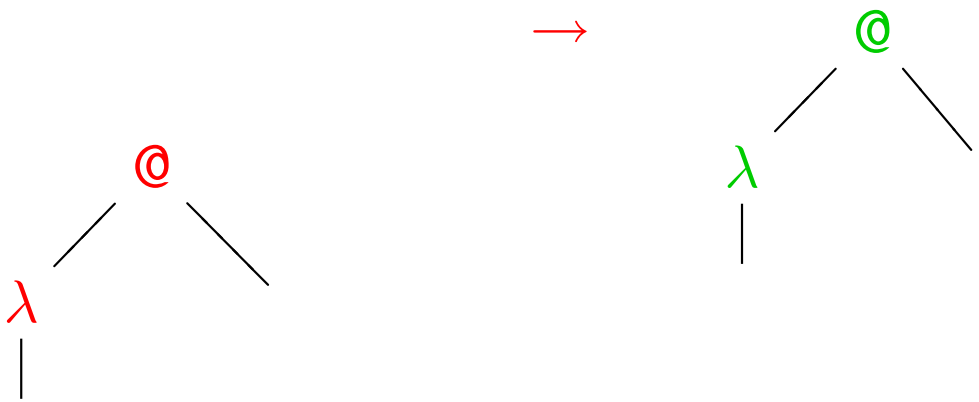
- Operational semantics

$$(\lambda x.A)B \qquad \rightarrow_\beta \qquad A[x := B]$$

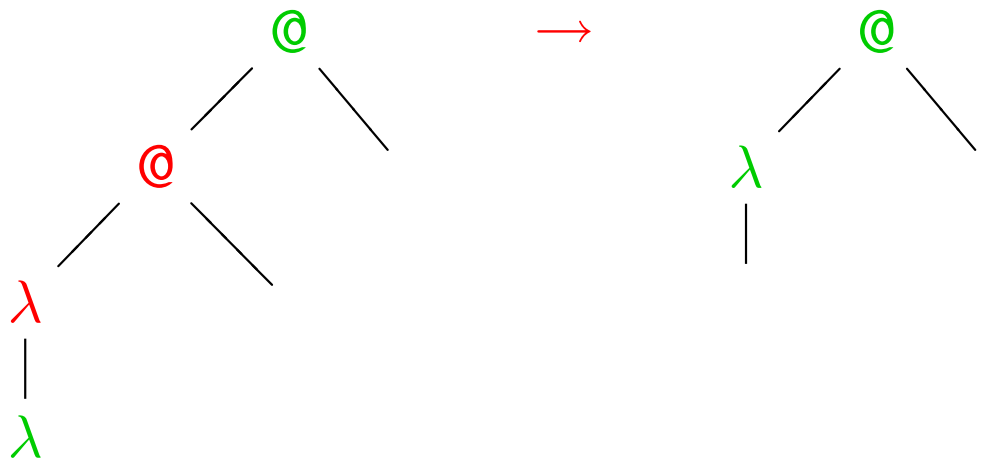
Creation of redexes in the λ-calculus (I)



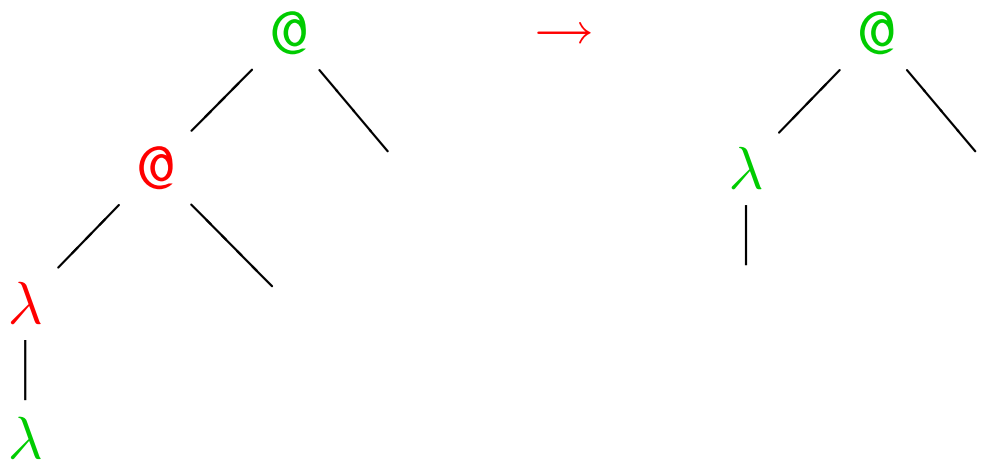
Creation of redexes in the λ-calculus (I)



Creation of redexes in the λ-calculus (I)

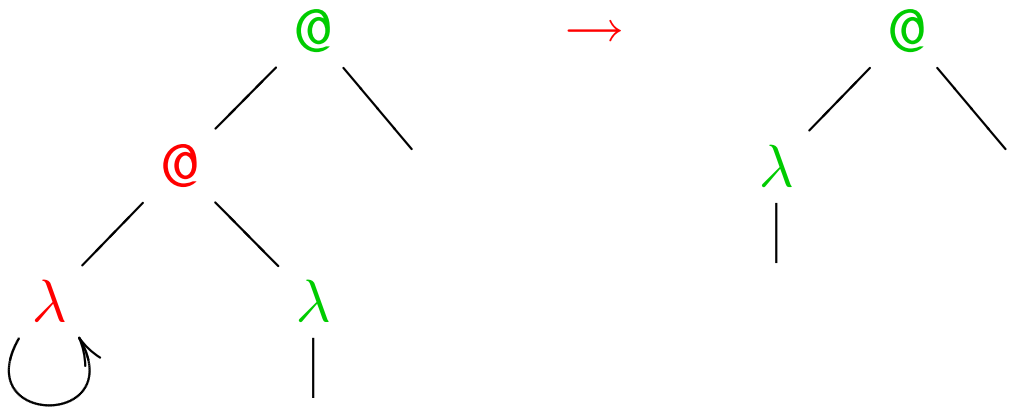


Creation of to create redexes in the λ-calculus (I)



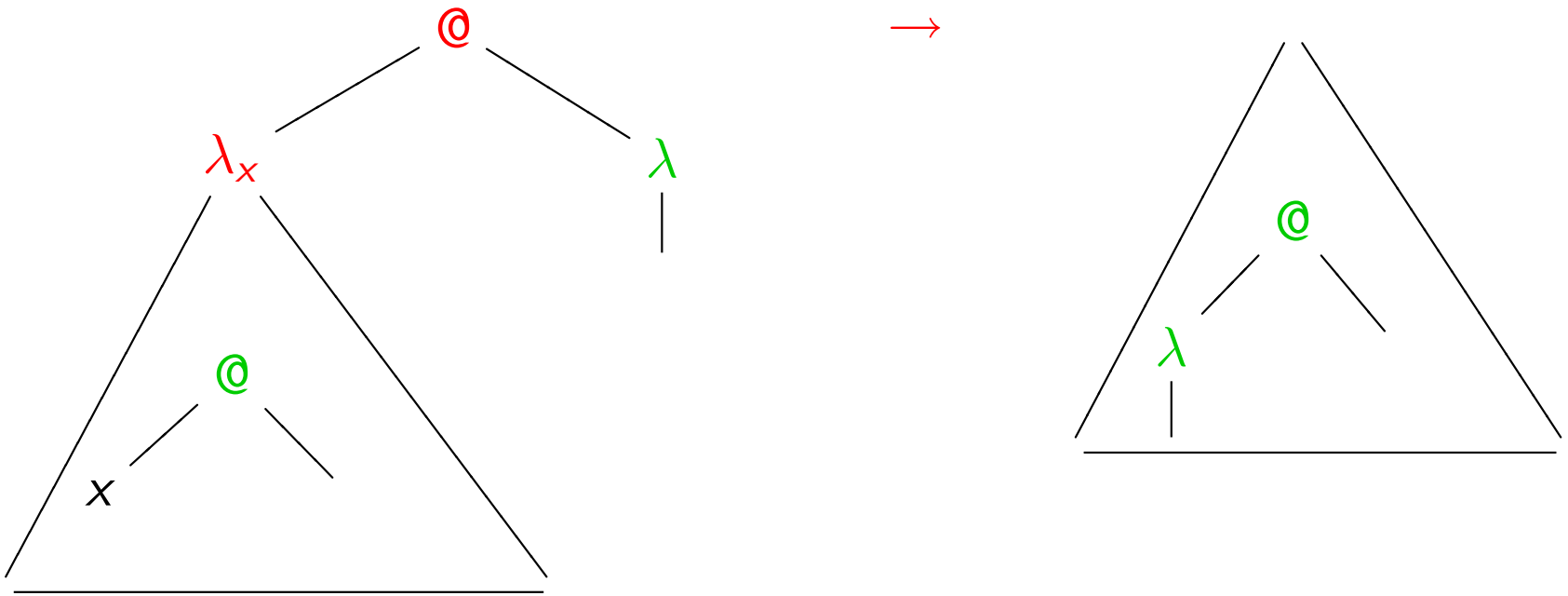
$((\lambda x.\lambda y.x)\ a)\ b \quad \rightarrow \quad (\lambda y.a)\ b$

Creation of redexes in the λ-calculus (II)



$$((\lambda x.x)\,(\lambda y.a))\;b \quad \rightarrow \quad (\lambda y.a)\;b$$

Creation of redexes in the λ-calculus (III)



$(\lambda x.xa)\,(\lambda y.y) \quad \rightarrow \quad (\lambda y.y)a$

Return

Remarks

1. What distinguishe (I) and (II) from (III)?

Remarks

1. What distinguishe (I) and (II) from (III)?
   - The reduction of the term in functional position of the green application creates a redex in the (I) and (II) way

Remarks

1. What distinguishe (I) and (II) from (III)?
   - The reduction of the term in functional position of the green application creates a redex in the (I) and (II) way
   - Whereas in (III) it is the substitution of a λ-abstraction in functional position that creates a redex

Remarks

1. What distinguishe (I) and (II) from (III)?
   - The reduction of the term in functional position of the green application creates a redex in the (I) and (II) way
   - Whereas in (III) it is the substitution of a λ-abstraction in functional position that creates a redex
2. In other words

Remarks

1. What distinguishe (I) and (II) from (III)?
   - The reduction of the term in functional position of the green application creates a redex in the (I) and (II) way
   - Whereas in (III) it is the substitution of a λ-abstraction in functional position that creates a redex
2. In other words
   - In (I) and (II) the creation is "upwards"

Remarks

1. What distinguishe (I) and (II) from (III)?
   - The reduction of the term in functional position of the green application creates a redex in the (I) and (II) way
   - Whereas in (III) it is the substitution of a λ-abstraction in functional position that creates a redex
2. In other words
   - In (I) and (II) the creation is "upwards"
   - Whereas in (III), it is "backwards"

Defining superdevelopements

Definition [?]

A superdevelopement is a β-rewrite sequence that reduces:

Defining superdevelopements

Definition [?]

A superdevelopement is a β-rewrite sequence that reduces:

1. the redexes of the term and its residuals (as in developments)

Defining superdevelopements

Definition [?]

A superdevelopement is a β-rewrite sequence that reduces:

1. the redexes of the term and its residuals (as in developments)
2. the redexes created in (I) or (II) not in (III)

Defining superdevelopements

Definition [?]

A superdevelopement is a β-rewrite sequence that reduces:

1. the redexes of the term and its residuals (as in developments)
2. the redexes created in (I) or (II) not in (III)

Defining superdevelopements

Definition [?]

A superdevelopement is a β-rewrite sequence that reduces:

1. the redexes of the term and its residuals (as in developments)
2. the redexes created in (I) or (II) not in (III)

Theorem [?]

All superdevelopments are finite

Defining superdevelopements

Definition [?]

A superdevelopement is a β-rewrite sequence that reduces:

1. the redexes of the term and its residuals (as in developments)
2. the redexes created in (I) or (II) not in (III)

Theorem [?]

All superdevelopments are finite

Examples of superdevelopments

Couter-examples of superdevelopments

Other characterization

Superdevelopments

The residuals of the redexes can be reduced as in developments

Example

$$
\begin{array}{ll}
(\lambda x.f(x,x))\,((\lambda y.y)\,a) & \\
\rightarrow_\beta & f((\lambda y.y)\,a, (\lambda y.y)\,a) \\
\rightarrow_\beta & f(a, (\lambda y.y)\,a) \\
\rightarrow_\beta & f(a,a)
\end{array}
$$

Superdevelopements

The following β-reduction is a superdevelopement:

Example

$$\begin{array}{ll} & ((\lambda x.\lambda y.f(x,y))a)b \\ \rightarrow_\beta & (\lambda y.f(a,y))b \\ \rightarrow_\beta & f(a,b) \end{array}$$

Superdevelopements

The following β-reduction is a superdevelopement:

Example

$$\begin{array}{ll} & ((\lambda x.x)(\lambda y.y))a \\ \rightarrow_\beta & (\lambda y.y)a \\ \rightarrow_\beta & a \end{array}$$

Superdevelopments

The following β-reductions are not superdeveloppements:

Example

$$
\begin{array}{ll}
(\lambda x.xa)(\lambda y.y) & \\
\rightarrow_\beta & (\lambda y.y)a \\
\rightarrow_\beta & a
\end{array}
$$

Superdevelopments

The following β-reductions are not superdeveloppements:

Example

$$\begin{array}{ll} & (\lambda x.xa)(\lambda y.y) \\ \to_\beta & (\lambda y.y)a \\ \to_\beta & a \end{array}$$

Example

$$\begin{array}{ll} & (\lambda x.xx)(\lambda x.xx) \\ \to_\beta & (\lambda x.xx)(\lambda x.xx) \\ \to_\beta & (\lambda x.xx)(\lambda x.xx) \\ \to_\beta & \ldots \end{array}$$

The parallel β-reduction [Tait Martin-Löf]

$$(\textsc{Red}\text{-}\beta)\ \frac{\lambda x.A_1 \Longrightarrow_\beta \lambda x.A_2 \qquad B_1 \Longrightarrow_\beta B_2}{(\lambda x.A_1)B_1 \Longrightarrow_\beta A_2[x := B_2]}$$

The parallel β-reduction [Tait Martin-Löf]

$$(\textsc{Red}\text{-}\beta)\ \frac{\lambda x.A_1 \Longrightarrow_\beta \lambda x.A_2 \qquad B_1 \Longrightarrow_\beta B_2}{(\lambda x.A_1)B_1 \Longrightarrow_\beta A_2[x := B_2]}$$

$$(\textsc{Red}\text{-}\lambda)\ \frac{A_1 \Longrightarrow_\beta A_2}{\lambda x.A_1 \Longrightarrow_\beta \lambda x.A_2} \qquad (\textsc{Red}\text{-}@)\ \frac{A_1 \Longrightarrow_\beta A_2 \qquad B_1 \Longrightarrow_\beta B_2}{A_1 B_1 \Longrightarrow_\beta A_2 B_2}$$

The parallel β-reduction [Tait Martin-Löf]

$$(\text{Red-}\beta)\ \frac{\lambda x.A_1 \Longrightarrow_\beta \lambda x.A_2 \qquad B_1 \Longrightarrow_\beta B_2}{(\lambda x.A_1)B_1 \Longrightarrow_\beta A_2[x := B_2]}$$

$$(\text{Red-}\lambda)\ \frac{A_1 \Longrightarrow_\beta A_2}{\lambda x.A_1 \Longrightarrow_\beta \lambda x.A_2} \qquad (\text{Red-@})\ \frac{A_1 \Longrightarrow_\beta A_2 \qquad B_1 \Longrightarrow_\beta B_2}{A_1 B_1 \Longrightarrow_\beta A_2 B_2}$$

$$(\text{Red-}\aleph)\ \frac{}{\aleph \Longrightarrow_\beta \aleph}$$

The strong parallel β-reduction [?]

$$(\textsc{Red}\text{-}\beta_f)\ \frac{A_1 \Longrightarrow_{\beta_f} \lambda x.A_2 \qquad B_1 \Longrightarrow_{\beta_f} B_2}{A_1 B_1 \Longrightarrow_{\beta_f} A_2[x := B_2]}$$

The strong parallel β-reduction [?]

$$(\textsc{Red}\text{-}\beta_f)\ \frac{A_1 \Longrightarrow_{\beta_f} \lambda x.A_2 \qquad B_1 \Longrightarrow_{\beta_f} B_2}{A_1 B_1 \Longrightarrow_{\beta_f} A_2[x := B_2]}$$

$$(\textsc{Red}\text{-}\lambda)\ \frac{A_1 \Longrightarrow_{\beta_f} A_2}{\lambda x.A_1 \Longrightarrow_{\beta_f} \lambda x.A_2} \qquad (\textsc{Red}\text{-}@)\ \frac{A_1 \Longrightarrow_{\beta_f} A_2 \qquad B_1 \Longrightarrow_{\beta_f} B_2}{A_1 B_1 \Longrightarrow_{\beta_f} A_2 B_2}$$

$$(\textsc{Red}\text{-}\aleph)\ \frac{}{\aleph \Longrightarrow_{\beta_f} \aleph}$$

Equivalence between superdevelopments

Theorem [?]

There exists a superdevelopment $A \mapsto\!\!\!\!\twoheadrightarrow_{\beta} B$

iff

$A \Longrightarrow_{\beta_f} B$

Matching in the pure λ-calculus modulo SD

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-matching equation]

a pair of (untyped) terms denoted $A \leqslant_{\beta_{sd}} B$ s.t.

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-matching equation]

a pair of (untyped) terms denoted $A \leqslant_{\beta_{sd}} B$ s.t.

- B is normal

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-matching equation]

a pair of (untyped) terms denoted $A \leqslant_{\beta_{sd}} B$ s.t.

- B is normal
- does not contain matching variables

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-matching equation]

a pair of (untyped) terms denoted $A \leqslant_{\beta_{sd}} B$ s.t.

- B is normal
- does not contain matching variables

Definition [β_{sd}-matching system]

a multiset of matching equations

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-match for $A \leqslant_{\beta_{sd}} B$]

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-match for $A \leqslant_{\beta_{sd}} B$]

A substitution ϕ on matching variables such that

$$A\phi \Longrightarrow_{\beta_f} B$$

Matching in the pure λ-calculus modulo SD

Definition [β_{sd}-match for $A \leqslant_{\beta_{sd}} B$]

A substitution ϕ on matching variables such that

$$A\phi \Longrightarrow_{\beta_f} B$$

A substitution is a match of a system if it matches each equation

Solved form

Definition

- For a matching equation $X \leqslant_{\beta_{sd}} A$: no variables in A
- For a system: each matching variable occurs once

Matching in the simply typed λ-calculus modulo β

Matching in the simply typed λ-calculus modulo β

Definition [β-equation]

a pair of β-normal typed λ-terms denoted $A \ll_\beta B$

- of the same type

Matching in the simply typed λ-calculus modulo β

Definition [β-equation]

a pair of β-normal typed λ-terms denoted $A \ll_\beta B$

- of the same type
- B does not contain matching variables

Matching in the simply typed λ-calculus modulo β

Definition [β-equation]

a pair of β-normal typed λ-terms denoted $A \ll_\beta B$

- of the same type
- B does not contain matching variables

Definition [β-match for $A \ll_\beta B$]

A substitution ϕ, that preserves types and such that

$$A\phi =_\beta B$$

Matching in the simply typed λ-calculus modulo β

Definition [β-equation]

a pair of β-normal typed λ-terms denoted $A \ll_\beta B$

- of the same type
- B does not contain matching variables

Definition [β-match for $A \ll_\beta B$]

A substitution ϕ, that preserves types and such that

$$A\phi =_\beta B$$

We can associate to a β-match equation, a β_{sd}-match equation (erase types)

An example

The equation $X^{(\iota\to\iota)\to\iota}\, Y^{(\iota\to\iota)} \ll_\beta a^{\kappa\to\iota\to\iota}(b^\kappa, c^\iota)$

An example

The equation $X^{(\iota\to\iota)\to\iota}\, Y^{(\iota\to\iota)} \ll_\beta a^{\kappa\to\iota\to\iota}(b^\kappa, c^\iota)$

	β_{sd}-match	β-match
$X \leftarrow \lambda x.a(b,c)$	✓	✓

An example

The equation $X^{(\iota\to\iota)\to\iota}\, Y^{(\iota\to\iota)} \ll_\beta a^{\kappa\to\iota\to\iota}(b^\kappa, c^\iota)$

	β_{sd}-match	β-match
$X \leftarrow \lambda x.a(b,c)$	✓	✓
$X \leftarrow ab \qquad Y \leftarrow c$	✓	✗

An example

The equation $X^{(\iota\to\iota)\to\iota}\, Y^{(\iota\to\iota)} \ll_\beta a^{\kappa\to\iota\to\iota}(b^\kappa, c^\iota)$

		β_{sd}-match	β-match
$X \leftarrow \lambda x.a(b,c)$		✓	✓
$X \leftarrow ab$	$Y \leftarrow c$	✓	✗
$X \leftarrow \lambda x.a(b,xc)$	$Y \leftarrow \lambda y.y$	✗	✓

SD and second-order matching

Proposition

Let ϕ be a β-match a second-order β-matching equation. Then it is also a β_{sd}-match.

SD and second-order matching

Proposition

Let ϕ be a β-match a second-order β-matching equation. Then it is also a β_{sd}-match.

Proof In (III) there is a 3^{rd}-order redex

Remarks

- The previous result does not extend to third-order

Remarks

- The previous result does not extend to third-order
- There are some second-order β-match equations that have no solution but such that the corresponding β_{sd}-match equation do have solutions

Remarks

- The previous result does not extend to third-order
- There are some second-order β-match equations that have no solution but such that the corresponding β_{sd}-match equation do have solutions

Remarks

- The previous result does not extend to third-order
- There are some second-order β-match equations that have no solution but such that the corresponding β_{sd}-match equation do have solutions
  - ➥ Np-completeness of second-order matching ?

SD and patterns *à la* Miller

The comparison is important: the algorithm of D. Miller does not use any type information

SD and patterns *à la* Miller

The comparison is important: the algorithm of D. Miller does not use any type information

Proposition

Let P be a pattern.

SD and patterns *à la* Miller

The comparison is important: the algorithm of D. Miller does not use any type information

Proposition

Let P be a pattern. If ϕ is a β-match of $P \ll_\beta B$ then

it it also a β_{sd}-match.

SD and patterns *à la* Miller

The comparison is important: the algorithm of D. Miller does not use any type information

Proposition

Let P be a pattern. If ϕ is a β-match of $P \ll_\beta B$ then

it it also a β_{sd}-match.

- **Proof:** Patterns do not need full β-conversion but only β_0-conversion:

$$(\lambda x.t)x \quad \rightarrow \quad t$$

and this reduction does not create redexes in (III).

Algorithm for matching modulo superdevelopments

Algorithm for matching modulo superdevelopments

$$\overline{A\sigma \Longrightarrow_{\beta_f} B}$$

Algorithm for matching modulo superdevelopments

$$\frac{}{a\sigma \Longrightarrow_{\beta_f} a}$$

Algorithm for matching modulo superdevelopments

$$(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} \quad \rightarrow_{\aleph_c} \quad \mathbb{S}$$

$$(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} \quad \rightarrow_{\aleph_v} \quad \mathbb{S}$$

$$\frac{}{a\sigma \Longrightarrow_{\beta_f} a}$$

Algorithm for matching modulo superdevelopments

$$(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} \quad \rightarrow_{\aleph_c} \quad \mathbb{S}$$
$$(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} \quad \rightarrow_{\aleph_v} \quad \mathbb{S}$$

$$\overline{X\sigma \Longrightarrow_{\beta_f} B}$$

Algorithm for matching modulo superdevelopments

$$(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} \quad \rightarrow_{\aleph_c} \quad \mathbb{S}$$

$$(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} \quad \rightarrow_{\aleph_v} \quad \mathbb{S}$$

$$(X \leqslant_{\beta_{sd}} B),\ \mathbb{S} \quad \rightarrow_{\aleph_X} \quad X \leqslant_{\beta_{sd}} B,\ \mathbb{S}\{A/X\}$$

if $\mathcal{FV}(B) = \emptyset$ **and** $X \in \mathbb{S}$

$$\frac{}{X\sigma \Longrightarrow_{\beta_f} B}$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \rightarrow_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \rightarrow_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \rightarrow_{\aleph_X} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots
\end{array}
$$

$$
\frac{}{(\lambda x.A_1)\sigma \Longrightarrow_{\beta_f}}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \rightarrow_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \rightarrow_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \rightarrow_{\aleph_X} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots
\end{array}
$$

$$
\frac{}{(\lambda x.A_1)\sigma \Longrightarrow_{\beta_f} \lambda x.B_1}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A \leqslant_{\beta_{sd}} \lambda x.B),\ \mathbb{S} & \to_{\lambda_\lambda} & (A \leqslant_{\beta_{sd}} B),\ \mathbb{S}
\end{array}
$$

$$
\frac{}{(\lambda x.A_1)\sigma \Longrightarrow_{\beta_f} \lambda x.B_1}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S}
\end{array}
$$

$$
\frac{}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f}}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S}
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1 B_2}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S}
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1 B_2}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S}
\end{array}
$$

$$
\frac{}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f}}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S}
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1[x := B_2]}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S}
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1[x := B_2]}
$$

$x \in B_1$?

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \rightarrow_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \rightarrow_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \rightarrow_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \rightarrow_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \rightarrow_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S}
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1}
$$

$x \notin B_1$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_x} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} & \to_{@_\pi} & A_1 \leqslant_{\beta_{sd}} \lambda x.B_1 \\
 & & \text{if } x \notin B_1
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1}
$$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \rightarrow_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \rightarrow_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \rightarrow_{\aleph_X} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \rightarrow_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \rightarrow_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} & \rightarrow_{@_\pi} & A_1 \leqslant_{\beta_{sd}} \lambda x.B_1 \quad \textbf{if} \ldots
\end{array}
$$

$$(A_1 A_2)\sigma \Longrightarrow_{\beta_f}$$

$x \in B_1$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_X} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} & \to_{@_\pi} & A_1 \leqslant_{\beta_{sd}} \lambda x.B_1 \quad \textbf{if} \ldots
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1[x := B_2]}
$$

$x \in B_1$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_X} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} & \to_{@_\pi} & A_1 \leqslant_{\beta_{sd}} \lambda x.B_1 \quad \textbf{if} \ldots \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1[x := B_2]),\ \mathbb{S} & \to_{@_\beta} & (A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S}
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1[x := B_2]}
$$

$x \in B_1$

Algorithm for matching modulo superdevelopments

$$
\begin{array}{lll}
(a \leqslant_{\beta_{sd}} a),\ \mathbb{S} & \to_{\aleph_c} & \mathbb{S} \\
(x \leqslant_{\beta_{sd}} x),\ \mathbb{S} & \to_{\aleph_v} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A),\ \mathbb{S} & \to_{\aleph_X} & X \leqslant_{\beta_{sd}} A,\ \mathbb{S}\{A/X\} \quad \textbf{if} \ldots \\
(\lambda x.A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ \mathbb{S} & \to_{\lambda_\lambda} & (A_1 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1 B_2),\ \mathbb{S} & \to_{@_@} & (A_1 \leqslant_{\beta_{sd}} B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S} \\
(A_1 A_2 \leqslant_{\beta_{sd}} B_1),\ \mathbb{S} & \to_{@_\pi} & A_1 \leqslant_{\beta_{sd}} \lambda x.B_1 \quad \textbf{if} \ldots \\
(A_1 A_2 \leqslant_{\beta_{sd}} C),\ \mathbb{S} & \to_{@_\beta} & (A_1 \leqslant_{\beta_{sd}} \lambda x.B_1),\ (A_2 \leqslant_{\beta_{sd}} B_2),\ \mathbb{S} \\
 & & \textbf{where } C = B_1[x := B_2] \\
 & & \textbf{and } x \in B_1
\end{array}
$$

$$
\frac{A_1\sigma \Longrightarrow_{\beta_f} \lambda x.B_1 \qquad A_2\sigma \Longrightarrow_{\beta_f} B_2}{(A_1 A_2)\sigma \Longrightarrow_{\beta_f} B_1[x := B_2]}
$$

Example: $XY \leqslant_{\beta_{sd}} ab$

- Rule @@: $X \leqslant_{\beta_{sd}} a,\ Y \leqslant_{\beta_{sd}} b$

Example: $XY \leqslant_{\beta_{sd}} ab$

- Rule $@_{@}$: $X \leqslant_{\beta_{sd}} a,\ Y \leqslant_{\beta_{sd}} b$
- Rule $@_{\pi}$: $X \leqslant_{\beta_{sd}} \lambda x.ab$

Example: $XY \leqslant_{\beta_{sd}} ab$

- Rule $@_{@}$: $X \leqslant_{\beta_{sd}} a,\ Y \leqslant_{\beta_{sd}} b$
- Rule $@_{\pi}$: $X \leqslant_{\beta_{sd}} \lambda x.ab$
- Rule $@_{\beta}$:
  -
  -
  -

Example: $XY \leqslant_{\beta_{sd}} ab$

- Rule $@_@$: $X \leqslant_{\beta_{sd}} a,\ Y \leqslant_{\beta_{sd}} b$
- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.ab$
- Rule $@_\beta$:
  - $Y \leqslant_{\beta_{sd}} ab$
  - $Y \leqslant_{\beta_{sd}} a$
  - $Y \leqslant_{\beta_{sd}} b$

Example: $XY \leqslant_{\beta_{sd}} ab$

- Rule $@_@$: $X \leqslant_{\beta_{sd}} a,\ Y \leqslant_{\beta_{sd}} b$
- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.ab$
- Rule $@_\beta$:
  - $X \leqslant_{\beta_{sd}} \lambda x.x,\quad Y \leqslant_{\beta_{sd}} ab$
  - $X \leqslant_{\beta_{sd}} \lambda x.xb,\ Y \leqslant_{\beta_{sd}} a$
  - $X \leqslant_{\beta_{sd}} \lambda x.ax,\ Y \leqslant_{\beta_{sd}} b$

Example: $X(YX) \leqslant_{\beta_{sd}} a$

- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.a$

Example: $X(YX) \leqslant_{\beta_{sd}} a$

- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.a$
- Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ YX \leqslant_{\beta_{sd}} a$

Example: $X(YX) \leqslant_{\beta_{sd}} a$

- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.a$
- Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ YX \leqslant_{\beta_{sd}} a$
  - Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.a$

Example: $X(YX) \leqslant_{\beta_{sd}} a$

- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.a$
- Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ YX \leqslant_{\beta_{sd}} a$
  - Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.a$
  - Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.x,\ X \leqslant_{\beta_{sd}} a.$

Example: $X(YX) \leqslant_{\beta_{sd}} a$

- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.a$
- Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ YX \leqslant_{\beta_{sd}} a$
  - Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.a$
  - Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.x,\ X \leqslant_{\beta_{sd}} a.$
    - Rule $\aleph_X$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.x,\ \lambda x.x \leqslant_{\beta_{sd}} a$

Example: $X(YX) \leqslant_{\beta_{sd}} a$

- Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.a$
- Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ YX \leqslant_{\beta_{sd}} a$
  - Rule $@_\pi$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.a$
  - Rule $@_\beta$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.x,\ X \leqslant_{\beta_{sd}} a.$
    - Rule $\aleph_X$: $X \leqslant_{\beta_{sd}} \lambda x.x,\ Y \leqslant_{\beta_{sd}} \lambda x.x,\ \lambda x.x \leqslant_{\beta_{sd}} a$

      ➥ Not a solved form

Main properties

- No new matching variables

Main properties

- No new matching variables
- Termination

Main properties

- No new matching variables
- Termination
- Soundness

Main properties

- No new matching variables
- Termination
- Soundness
- Completeness

Main properties

- No new matching variables
- Termination
- Soundness
- Completeness
- Finite complete match set

Back to second-order

Applying the previous algorithm in a typed context:

We get a new algorithm for second-order matching

Matching modulo sd and η

The customization is easy

Matching modulo sd and η

The customization is easy

- Only generate $\beta\eta$-normal forms for the rule $@_\beta$

Matching modulo sd and η

The customization is easy

- Only generate $\beta\eta$-normal forms for the rule $@_\beta$
- Perform η-expansion on the fly

Matching modulo sd and η

The customization is easy

- Only generate $\beta\eta$-normal forms for the rule $@_\beta$
- Perform η-expansion on the fly

$$\lambda x.A \leqslant^{\eta}_{\beta_{sd}} B,\ \mathbb{S} \qquad \rightarrow_{\lambda\_} \qquad A \leqslant^{\eta}_{\beta_{sd}} Bx,\ \mathbb{S}$$

if B is not an abstraction
and x fresh

Matching modulo sd and η

The customization is easy

- Only generate $\beta\eta$-normal forms for the rule $@_\beta$
- Perform η-expansion on the fly

$$\lambda x.A \leqslant^{\eta}_{\beta_{sd}} B,\ \mathbb{S} \qquad \rightarrow_{\lambda\_} \qquad A \leqslant^{\eta}_{\beta_{sd}} Bx,\ \mathbb{S}$$

if B is not an abstraction
and x fresh

Proposition

In the context of patterns *à la* Miller, this algorithm gives the most general match.

Conclusion

- A new approach to higher-order matching:

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional
- The simple algorithms
  - properties: termination, soudness, completeness

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional
- The simple algorithms
  - properties: termination, soudness, completeness
  - intuitive and simple proofs (provides by the use of SD)

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional
- The simple algorithms
  - properties: termination, soudness, completeness
  - intuitive and simple proofs (provides by the use of SD)
  - second-order matching if applied in a typed context

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional
- The simple algorithms
  - properties: termination, soudness, completeness
  - intuitive and simple proofs (provides by the use of SD)
  - second-order matching if applied in a typed context
- An implementation of the algorithm is available in the Tom language

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional
- The simple algorithms
  - properties: termination, soudness, completeness
  - intuitive and simple proofs (provides by the use of SD)
  - second-order matching if applied in a typed context
- An implementation of the algorithm is available in the Tom language
- Higher-order rewriting with the untyped λ-calculus with superdevelopments as a meta-language

Conclusion

- A new approach to higher-order matching: in the untyped λ-calculus with a restricted notion of β-equivalence where superdevelopments give the right intuitions
  - Complete for second-order
  - Complete for patterns *à la* Miller
  - Use of η-equivalence is optional
- The simple algorithms
  - properties: termination, soudness, completeness
  - intuitive and simple proofs (provides by the use of SD)
  - second-order matching if applied in a typed context
- An implementation of the algorithm is available in the Tom language
- Higher-order rewriting with the untyped λ-calculus with superdevelopments as a meta-language
- Higher-order matching in the ρ-calculus