Uma Comparação do Método de Unificação de Ordem Superior de Huet e Unificação via Cálculos de Substituições Explícitas

F. L. C. de Moura
Seminário de Computação - GTC/UnB

27 de abril de 2005

Outline

- Introduction
- Unification Tree Notation
- The $\lambda\sigma$-calculus
- The λs_e-calculus
- Comparing the $\lambda\sigma$- and the λs_e-styles of unification
- Conclusion

Motivation

- Higher-Order terms appear frequently in Mathematics, Logic, Automated Reasoning, etc.
- Higher Order Unification (HOU) is a basic operation extensively used in computational systems based on the λ-calculus such as functional programming languages and proof assistants.
- Explicit substitutions are a refinement of the λ-calculus in which the substitution operation is not treated as a meta-operation but as an operation of the calculus itself.

Simply typed λ-calculus in de Bruijn notation

Definition

The set $\Lambda_{dB}(\mathcal{X})$ of untyped λ-terms in de Bruijn notation:

$$a ::= \underline{\mathtt{n}} \mid X \mid (a\ b) \mid \lambda.a \qquad \text{where } n \in \mathbb{N} \text{ and } X \in \mathcal{X}.$$

The syntax of simply typed λ-calculus in de Bruijn notation:

Types	$A ::= K \mid A \to B$	
Contexts	$\Gamma ::= nil \mid A.\Gamma$	
Terms	$a ::= \underline{\mathtt{n}} \mid X \mid (a\ b) \mid \lambda_A.a$	where $n \in \mathbb{N}$ and $X \in \mathcal{X}$.

The type of the term a is indicated by $\tau(a)$.

Simply typed λ-calculus in de Bruijn notation

Definition

1. Every λ-term in β-normal form (β-nf) has the form

$$\lambda_{A_1} \dots \lambda_{A_n}.(h\ e_1 \dots e_p)$$

   where $n, p \geq 0$, h is a variable (or a constant) called its *head* and $e_1, \dots, e_p$ are λ-terms in β-nf called its *arguments*.
2. A λ-term in β-nf is *rigid* if its head is a constant or a bound variable. If it is a meta-variable, the term is *flexible*.

Simply typed λ-calculus in de Bruijn notation

3 Let $a \in \Lambda_{dB}(\mathcal{X})$ be a λ-term in de Bruijn notation of type $A_1 \to \ldots \to A_m \to B$ with B atomic. The η-*long* form of a β-nf term a, written a', is inductively defined as follows:

- if $a = \lambda_A.b$ then $a' = \lambda_A.b'$.
- if $a = (\underline{\mathtt{n}}\ b_1 \ldots b_q)$ then $a' = \lambda_{A_1} \ldots \lambda_{A_m}.(\underline{\mathtt{n}+\mathtt{m}}\ c_1 \ldots c_q\ \underline{\mathtt{m}}' \ldots \underline{\mathtt{1}}')$, where c_j $(1 \leq j \leq q)$ is the η-long form of the normal form of $U_0^{m+1}(b_j)$.
- if $a = (X\ b_1 \ldots b_q)$ then $a' = \lambda_{A_1} \ldots \lambda_{A_m}.(X\ c_1 \ldots c_q\ \underline{\mathtt{m}}' \ldots \underline{\mathtt{1}}')$, where c_j $(1 \leq j \leq q)$ is the η-long form of the normal form of $U_0^{m+1}(b_j)$.

Unification problems

Definition

A *unification equation* is an equation of the form $a =^? b$ where a and b are λ-terms of the same type and under the same context.
A *unification problem* is a finite set of unification equations.

Examples:

- $X_A^{A \cdot nil} =^? \underline{1}_A^{A \cdot nil}$

Unification problems

Definition

A *unification equation* is an equation of the form $a =^? b$ where a and b are λ-terms of the same type and under the same context. A *unification problem* is a finite set of unification equations.

Examples:

- $X_A^{A \cdot nil} =^? \underline{1}_A^{A \cdot nil}$
- Solution: $X/\underline{1}$

Unification problems

Definition

A *unification equation* is an equation of the form $a =^? b$ where a and b are λ-terms of the same type and under the same context. A *unification problem* is a finite set of unification equations.

Examples:

- $X_A^{A \cdot nil} =^? \underline{1}_A^{A \cdot nil}$
- Solution: $X/\underline{1}$
- $(X_{A \to A}^{A \cdot nil}\ \underline{1}_A^{A \cdot nil}) =^? (\underline{2}_{A \to A}^{A \cdot nil}(Y_{A \to A}^{A \cdot nil}\ \underline{1}_A^{A \cdot nil}))$

Unification problems

Definition

A *unification equation* is an equation of the form $a =^? b$ where a and b are λ-terms of the same type and under the same context. A *unification problem* is a finite set of unification equations.

Examples:

- $X_A^{A \cdot nil} =^? \underline{1}_A^{A \cdot nil}$
- Solution: $X/\underline{1}$
- $(X_{A \to A}^{A \cdot nil}\ \underline{1}_A^{A \cdot nil}) =^? (\underline{2}_{A \to A}^{A \cdot nil}(Y_{A \to A}^{A \cdot nil}\ \underline{1}_A^{A \cdot nil}))$
- Solutions: $\sigma_1 = \{X/\lambda_A.(\underline{3}\ \underline{1}), Y/\lambda_A.\underline{1}\}$
  $\sigma_2 = \{X/\lambda_A.(\underline{3}\ \underline{2}), Y/\lambda_A.\underline{1}\}$

Unification problems

- Let $\Delta = A \to A \cdot A \cdot nil$ be a context.
  $\lambda_A.(\underline{2}^{A\cdot\Delta}_{A\to A}\ X^{A\cdot\Delta}_A) =^? \lambda_A.(\underline{2}^{A\cdot\Delta}_{A\to A}\underline{3}^{A\cdot\Delta}_A)$

Unification problems

- Let $\Delta = A \to A \cdot A \cdot nil$ be a context.
  $\lambda_A.(\underline{2}^{A\cdot\Delta}_{A\to A}\ X^{A\cdot\Delta}_{A}) =^? \lambda_A.(\underline{2}^{A\cdot\Delta}_{A\to A}\underline{3}^{A\cdot\Delta}_{A})$
- Solution: $X/\underline{2}$

Unification problems

- Let $\Delta = A \rightarrow A \cdot A \cdot nil$ be a context. $\lambda_A.(\underline{2}^{A\cdot\Delta}_{A\rightarrow A}\ X^{A\cdot\Delta}_{A}) =^? \lambda_A.(\underline{2}^{A\cdot\Delta}_{A\rightarrow A}\underline{3}^{A\cdot\Delta}_{A})$
- Solution: $X/\underline{2}$
- $\lambda_A.X^{A\cdot\Gamma}_A =^? \lambda_A.\underline{1}^{A\cdot\Gamma}_A$, where Γ is any context, does not have solutions.

The procedure SIMPL

INPUT: A unif. problem P with at least one rigid-rigid equation:

$$\lambda_{A_1}\ldots\lambda_{A_r}.(\underline{\mathtt{n}}\ e^1_1\ldots e^1_{p_1}) =^? \lambda_{A_1}\ldots\lambda_{A_r}.(\underline{\mathtt{m}}\ e^2_1\ldots e^2_{p_2}) \wedge P'$$

where $r, p_1, p_2 \geq 0$ and $n, m > 0$.
WHILE there exists a rigid-rigid equation in P DO
If $n \neq m$ then stop and report a failure status else let $p = p_1 = p_2$ and replace the selected equation by the conjunction

$$\lambda_{A_1}\ldots\lambda_{A_r}.e^1_1 =^? \lambda_{A_1}\ldots\lambda_{A_r}.e^2_1 \wedge\ldots\wedge \lambda_{A_1}\ldots\lambda_{A_r}.e^1_p =^? \lambda_{A_1}\ldots\lambda_{A_r}.e^2_p$$

in P and call the result $\overline{P}$ (the simplified version of P).
DONE.
If there exists a flexible-rigid equation in $\overline{P}$ then return $\overline{P}$ else stop and report a success status.

Example of SIMPL

$$\lambda_{A\to A\to A}\lambda_A.(\underline{2}_{A\to A\to A}\ X_A\ \underline{1}_A) =^? \lambda_{A\to A\to A}\lambda_A.(\underline{2}_{A\to A\to A}\ \underline{3}_A\ (Y_{A\to A}\ \underline{1}))$$

simplifies to

Example of SIMPL

$$\lambda_{A\to A\to A}\lambda_A.(\underline{2}_{A\to A\to A}\ X_A\ \underline{1}_A) =^? \lambda_{A\to A\to A}\lambda_A.(\underline{2}_{A\to A\to A}\ \underline{3}_A\ (Y_{A\to A}\ \underline{1}))$$

simplifies to

$$\lambda_{A\to A\to A}\lambda_A.X_A =^? \lambda_{A\to A\to A}\lambda_A.\underline{3}_A$$

$$\wedge$$

Example of SIMPL

$$\lambda_{A\to A\to A}\lambda_A.(\underline{2}_{A\to A\to A}\ X_A\ \underline{1}_A) =^? \lambda_{A\to A\to A}\lambda_A.(\underline{2}_{A\to A\to A}\ \underline{3}_A\ (Y_{A\to A}\ \underline{1}))$$

simplifies to

$$\lambda_{A\to A\to A}\lambda_A.X_A =^? \lambda_{A\to A\to A}\lambda_A.\underline{3}_A$$

$$\wedge$$

$$\lambda_{A\to A\to A}\lambda_A.\underline{1}_A =^? \lambda_{A\to A\to A}\lambda_A.(Y_{A\to A}\ \underline{1})$$

The procedure MATCH

- Takes a flexible-rigid equation as argument and returns a finite set of substitutions called Σ.

The procedure MATCH

- Takes a flexible-rigid equation as argument and returns a finite set of substitutions called Σ.
- Input: A flexible-rigid equation of the form:

$$\lambda_{A_1} \ldots \lambda_{A_r}.(X\ e^1_1 \ldots e^1_{p_1}) =^? \lambda_{A_1} \ldots \lambda_{A_r}.(\underline{\mathrm{n}}\ e^2_1 \ldots e^2_{p_2}) \tag{1}$$

where $\tau(X) = B_1 \to \ldots \to B_{p_1} \to C$, where $p_1, p_2, r \geq 0$, $n > 0$ and C is atomic.

The procedure MATCH

- Takes a flexible-rigid equation as argument and returns a finite set of substitutions called Σ.
- Input: A flexible-rigid equation of the form:

$$\lambda_{A_1} \ldots \lambda_{A_r}.(X\ e^1_1 \ldots e^1_{p_1}) =^? \lambda_{A_1} \ldots \lambda_{A_r}.(\underline{\mathtt{n}}\ e^2_1 \ldots e^2_{p_2}) \qquad (1)$$

  where $\tau(X) = B_1 \to \ldots \to B_{p_1} \to C$, where $p_1, p_2, r \geq 0$, $n > 0$ and C is atomic.
- The procedure MATCH is based on two rules named *imitation* and *projection*.

The imitation rule

The imitation substitution corresponds exactly to the η-long term of the type of X, whose head corresponds to the head of the rigid term:

$$X/\lambda_{B_1}\ldots\lambda_{B_{p_1}}.(\underline{\mathtt{p}_1+\mathtt{n}-\mathtt{r}}\,(X_1\ \underline{\mathtt{p}_1}\ldots\underline{1})\ldots(X_{p_2}\ \underline{\mathtt{p}_1}\ldots\underline{1}))$$

where $X_1,\ldots,X_{p_2}$ are meta-variables with appropriate type and all sub-terms are in η-normal form.

Imitation example

Consider the equation:

$$\lambda_A\lambda_A.(X_{A\to A}\ \underline{1}_A) =^? \lambda_A\lambda_A.(\underline{3}_{A\to A}(Y_{A\to A}(\underline{4}_{A\to A}\ \underline{1}_A)))$$

Generated imitation substitution:

$$X_{A\to A}/\lambda_A.(\underline{2}_{A\to A}\ (X_{1_{A\to A}}\ \underline{1}_A))$$

The projection rule

- A *projection* can be used in case the head of the rigid term is a constant or a bound variable.

The projection rule

- A *projection* can be used in case the head of the rigid term is a constant or a bound variable.
- The projection rule consists in "projecting" the head of the flexible term onto one of its arguments which eventually contains the index that corresponds to the head of the rigid term.

The projection rule

- A *projection* can be used in case the head of the rigid term is a constant or a bound variable.
- The projection rule consists in "projecting" the head of the flexible term onto one of its arguments which eventually contains the index that corresponds to the head of the rigid term.
- The projections substitutions always have the form $\lambda_{B_1} \ldots \lambda_{B_{p_1}}.(\underline{\mathrm{i}}\ (X_1\ \underline{\mathrm{p}_1} \ldots \underline{1}) \ldots (X_k\ \underline{\mathrm{p}_1} \ldots \underline{1}))$, where $1 \leq i \leq p_1$.

The projection rule

- A *projection* can be used in case the head of the rigid term is a constant or a bound variable.
- The projection rule consists in "projecting" the head of the flexible term onto one of its arguments which eventually contains the index that corresponds to the head of the rigid term.
- The projections substitutions always have the form $\lambda_{B_1} \ldots \lambda_{B_{p_1}}.(\underline{i}\ (X_1\ \underline{p_1} \ldots \underline{1}) \ldots (X_k\ \underline{p_1} \ldots \underline{1}))$, where $1 \leq i \leq p_1$.
- This gives at most p_1 possible different projections, one for each argument of X.

Projection example

Consider the equation:

$$\lambda_A \lambda_A.(X_{A\to A}\ \underline{1}_A) =^? \lambda_A \lambda_A.\underline{1}_A$$

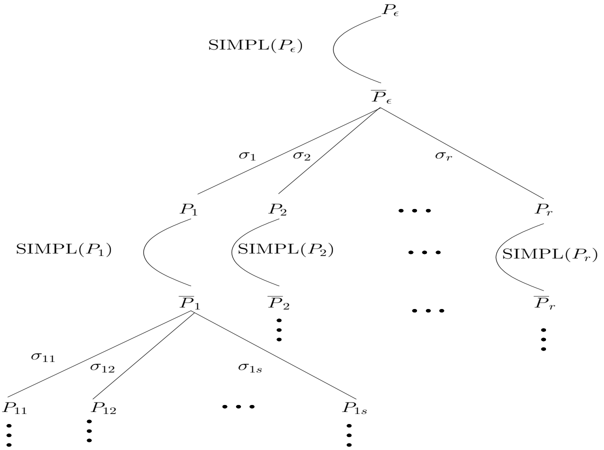
Generated projection substitution:

$$X_{A\to A}/\lambda_A.\underline{1}_A$$

Unification Tree Notation

- The unification tree notation is obtained from the matching tree of Huet by adding labels to the unification problems as well as to the generated substitutions.
- These labels provide information about the position of the unification problems and of the substitutions in the matching tree.
- Facilitates the computation of the solutions.

Visualising the Tree

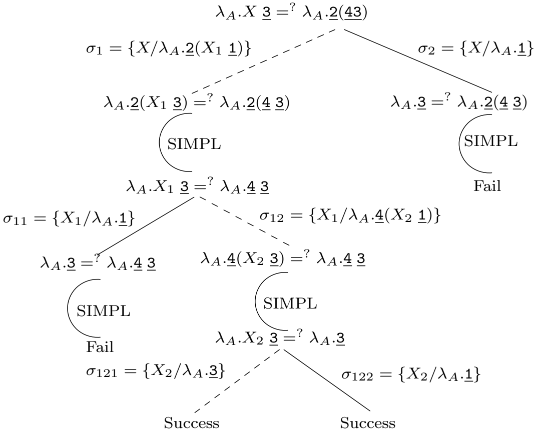


Formal Construction

A unification tree, for a given unification problem P, is given by:

1. Label P with ϵ (the empty position) as a subscript, i.e., P_ϵ.
2. For a node labeled with P_q, its sibling node is labeled with $\overline{P_q}$ whenever the unification problem derives by applying the procedure SIMPL. This step is represented by a curly line in the unification.
3. For a node labeled with P_q containing a flexible-rigid equation, call $\sigma_{q1}, \sigma_{q2}, \ldots, \sigma_{qk}$ the incremental substitutions generated by an application of the procedure MATCH to this equation.
4. The sibling nodes of P_q, written $P_{q1}, \ldots, P_{qk}$ are defined by the composition $P_{qi} := \overline{P_q}\sigma_{qi}$, for $i = 1, \ldots, k$.

Example



_ _ _ _ dashed lines denote imitation steps
——— solid lines denote projection steps

The $\lambda\sigma$-calculus

The syntax of typed $\lambda\sigma$-calculus is given by

Types	A	$::=$	$K \mid A \rightarrow B$
Contexts	Γ	$::=$	$nil \mid A \cdot \Gamma$
Terms	a	$::=$	$\underline{1} \mid X \mid (a\ b) \mid \lambda_A.a \mid a[s]$
Substitutions	s	$::=$	$id \mid \uparrow \mid a \cdot s \mid s \circ s$

The $\lambda\sigma$-calculus

The typing rules:

(var) $A.\Gamma \vdash \underline{1} : A$

(lambda) $$\frac{A.\Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \to B}$$

(app) $$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B}$$

(clos) $$\frac{\Gamma \vdash s \rhd \Gamma' \quad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A}$$

(id) $\Gamma \vdash id \rhd \Gamma$

(shift) $A.\Gamma \vdash \uparrow \rhd \Gamma$

(cons) $$\frac{\Gamma \vdash a : A \quad \Gamma \vdash s \rhd \Gamma'}{\Gamma \vdash a.s \rhd A.\Gamma'}$$

(comp) $$\frac{\Gamma \vdash s'' \rhd \Gamma'' \quad \Gamma'' \vdash s' \rhd \Gamma'}{\Gamma \vdash s' \circ s'' \rhd \Gamma'}$$

(meta) $\Gamma \vdash X : A$, where Γ is any context.

The $\lambda\sigma$-calculus

(Beta)	$(\lambda.a)b$	$\longrightarrow$	$a\,[b \cdot id]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda.a)[s]$	$\longrightarrow$	$\lambda(a\,[\underline{1} \cdot (s \circ \uparrow)])$
(Clos)	$(a\,[s])[t]$	$\longrightarrow$	$a\,[s \circ t]$
(VarCons)	$\underline{1}\,[a \cdot s]$	$\longrightarrow$	a
(Id)	$a[id]$	$\longrightarrow$	a
(Assoc)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$
(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a\,[t] \cdot (s \circ t)$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\longrightarrow$	s
(VarShift)	$\underline{1} \cdot \uparrow$	$\longrightarrow$	id
(SCons)	$\underline{1}[s] \cdot (\uparrow \circ s)$	$\longrightarrow$	s
(Eta)	$\lambda.(a\ \underline{1})$	$\longrightarrow$	b if $a =_\sigma b[\uparrow]$

Unification in the $\lambda\sigma$-calculus

Motivation:

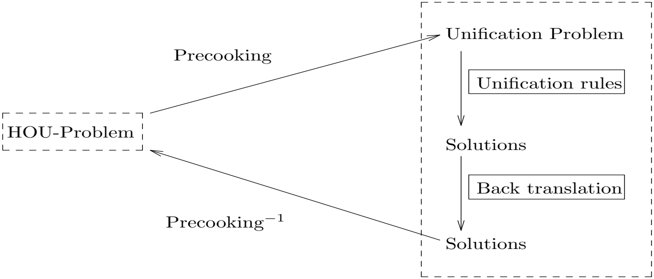
- Reduce substitution to grafting.
- Remain closer to implementations.
- Development of a programming language based on ES that includes HOU in a lower level.

Unification in the $\lambda\sigma$-calculus

Motivation:

- Reduce substitution to grafting.
- Remain closer to implementations.
- Development of a programming language based on ES that includes HOU in a lower level.
- Possible drawback:
  - Inclusion of a non-trivial equational theory (??).

Unification in Explicit Substitutions Calculi



Language of the λ-calculus
substitution

Language of the explicit substitutions calculus.
grafting

The $\lambda\sigma$-unification rules (part I)

The $\lambda\sigma$-simplification rules:

Dec-λ $$\frac{P \wedge \lambda_A.e_1 =^?_{\lambda\sigma} \lambda_A.e_2}{P \wedge e_1 =^?_{\lambda\sigma} e_2}$$

Dec-App $$\frac{P \wedge (\underline{\mathtt{n}}\ e^1_1 \ldots e^1_p) =^?_{\lambda\sigma} (\underline{\mathtt{n}}\ e^2_1 \ldots e^2_p)}{P \wedge e^1_1 =^?_{\lambda\sigma} e^2_1 \wedge \ldots \wedge e^1_p =^?_{\lambda\sigma} e^2_p}$$

Dec-Fail $$\frac{P \wedge (\underline{\mathtt{n}}\ e^1_1 \ldots e^1_{p_1}) =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ e^2_1 \ldots e^2_{p_2})}{\mathit{Fail}}, \text{ if } \mathtt{m} \neq \mathtt{n}.$$

The SIMPL$_{\lambda\sigma}$ procedure

`INPUT`: A unification problem P_q (in the language of the $\lambda\sigma$-calculus) with at least one rigid-rigid equation.
`OUTPUT`: A terminal (failure or success) status or an equivalent unification problem $\overline{P_q}$ without rigid-rigid equations and containing at least one flexible-rigid equation.

Assume that **Dec-**λ is applied eagerly.
`WHILE` there exists a rigid-rigid equation in P_q `DO`

1. Apply **Dec-Fail**, if possible.
2. Apply **Dec-App**, and if the resulting unification problem contains a flexible-rigid equation, call it $\overline{P_q}$ and give $\overline{P_q}$ as result, else stop and report a success status.

`DONE`.

The SIMPL$_{\lambda\sigma}$ procedure

Theorem

The application of the procedure SIMPL$_{\lambda\sigma}$ to any unification problem P (in the language of the $\lambda\sigma$-calculus) always terminates.

Proof.[Sketch] Applications of the simplification rules decrease the size of the terms in the equations. □

The SIMPL and SIMPL$_{\lambda\sigma}$ correspondence

Theorem

If P is a unification problem in the pure λ-calculus and P_F its precooked image, then:

1. *$SIMPL(P)$ fails $\Leftrightarrow$ $SIMPL_{\lambda\sigma}(P_F)$ fails.*
2. *$SIMPL(P)$ stops and reports a success status $\Leftrightarrow$ $SIMPL_{\lambda\sigma}(P_F)$ stops and reports a success status;*
3. *$SIMPL(P)$ returns a unification problem containing at least one flexible-rigid equation $\Leftrightarrow$ $SIMPL_{\lambda\sigma}(P_F)$ returns a unification problem containing at least one flexible-rigid equation.*

Solved forms

Definition (DHK00)

A unification problem P is in $\lambda\sigma$-*solved form* if all its meta-variables are of atomic type and it is a conjunction of nontrivial equations of the following forms:

- **Solved**: $X =^?_{\lambda\sigma} a$ where the meta-variable X does not appear anywhere else in P and a is in **Eta**-long form. Such an equation is said to be *solved* in P and the variable X is also said to be solved.
- **Flexible-flexible**: $X[a_1.\cdots.a_p.\uparrow^n] =^?_{\lambda\sigma} Y[b_1.\cdots.b_q.\uparrow^m]$, where $X[a_1.\cdots.a_p.\uparrow^n]$ and $Y[b_1.\cdots.b_q.\uparrow^m]$ are **Eta**-long terms and the equation is not solved.

The $\lambda\sigma$-unification rules (part II)

Exp-λ
$$\frac{P}{\exists Y : (A.\Gamma \vdash B), P \wedge X =^?_{\lambda\sigma} \lambda_A Y}$$
if $(X : \Gamma \vdash A \to B) \in \mathcal{TV}ar(P)$, $Y \notin \mathcal{TV}ar(P)$, and X is not a solved variable.

Normalise
$$\frac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e_1' =^?_{\lambda\sigma} e_2'}$$ if e_1 or e_2 is not in long form,
where e_1' (resp. e_2') is the long form of e_1 (resp. e_2) if e_1 (resp. e_2) is not a solved variable and e_1 (resp. e_2) otherwise.

Replace
$$\frac{P \wedge X =^?_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda\sigma} t}$$ if $X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t)$ and if t is a constant then $t \in \mathcal{TV}ar(P)$.

The $\lambda\sigma$-unification rules (part III)

$$\textbf{Exp-App}\ \frac{P \wedge X[a_1.\cdots.a_p.\uparrow^n] =^?_{\lambda\sigma} \underline{\mathtt{m}}(b_1,\ldots,b_q)}{P \wedge X[a_1.\cdots.a_p.\uparrow^n] =^?_{\lambda\sigma} \underline{\mathtt{m}}(b_1,\ldots,b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \ldots \exists H_k, X =^?_{\lambda\sigma} \underline{\mathtt{r}}(H_1,\ldots,H_k)}$$

if X has an atomic type and is not solved.
where $H_1, \ldots, H_k$ are variables of appropriate types, not occurring in P, with the contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{1, \ldots, p\}$ such that $\underline{\mathtt{r}}(H_1, \ldots, H_k)$ has the right type, $R_i =$if $m \geq n+1$ then $\{m-n+p\}$ else $\emptyset$.

The procedure MATCH$_{\lambda\sigma}$

`INPUT`: A unification system P_q with at least one flexible-rigid equation.
`OUTPUT`: A disjunction of equivalent unification systems, written $P_{q1} \vee \ldots \vee P_{qk}$.
Assume that the rule **Dec-**λ is applied eagerly.

1. Apply **Exp-**λ and **Replace** as much as possible to the selected equation and call P'_q the resulting unification system.
2. Apply **Exp-App** and **Replace** and **Normalise** to P'_q and call $P_{q1} \vee \ldots \vee P_{qk}$ the resulting unification problem.

The procedure MATCH$_{\lambda\sigma}$

Definition

Let X/a be a substitution generated in the pure λ-calculus by Huet's algorithm. We say that the equation $Y =^?_\xi b$ *corresponds (or is associated)* to the substitution X/a if X and Y are two meta-variables of the same type and the terms a and b have the same headings, where $\xi \in \{\lambda\sigma, \lambda s_e\}$.

Correspondence from MATCH to MATCH$_{\lambda\sigma}$

Theorem

Let $\lambda_{A_1}\ldots\lambda_{A_r}.(X\ e^1_1\ldots e^1_{p_1}) =^? \lambda_{A_1}\ldots\lambda_{A_r}.(\underline{\mathrm{n}}\ e^2_1\ldots e^2_{p_2})$ *be a flexible-rigid equation in* η*-long form in the pure* λ*-calculus where* $p_1, p_2, r \geq 0$ *and* $\tau(X) = B_1 \to \ldots \to B_{p_1} \to B$ *with* B *atomic. Then, for each substitution generated by the procedure MATCH, when applied to this equation, there exists a corresponding equation in the* $\lambda\sigma$*-calculus generated by the procedure MATCH*$_{\lambda\sigma}$ *to the precooked version of the given equation.*

Correspondence from MATCH$_{\lambda\sigma}$ to MATCH

Theorem

For each new generated equation by the rule **Exp-App**, *when applied to a flexible-rigid equation which is in the image of the precooking translation, there exists a corresponding substitution in the pure λ-calculus in the following sense: for each element in R_p there exists a corresponding substitution in the pure λ-calculus and, if $R_i \neq \emptyset$ then there exists an imitation in the pure λ-calculus for the inverse of the precooking translation applied to this equation.*

The MATCH and MATCH$_{\lambda\sigma}$ correspondence

Theorem

Let eq be a flexible-rigid equation in η-long form in the pure λ-calculus and eq_F its precooked image. Then, MATCH applied to eq generates a substitution σ if and only if MATCH$_{\lambda\sigma}$ applied to eq_F generates a substitution equivalent to σ.

The Main Procedure

`INPUT`: A unification system P_ϵ.
`OUTPUT`: A success or a failure status and in the former case the solutions are the solved equations whose left-hand side corresponds to the meta-variables of the initial problem. If the initial problem is non-unifiable the algorithm may not terminate.

1. If P_q contains a rigid-rigid equation, then apply SIMPL$_{\lambda\sigma}$ and go to the next step, else if P_q contains a non-solved flex-rig equation then rename it to $\overline{P_q}$ and go to the next step.
2. Apply MATCH$_{\lambda\sigma}$ to $\overline{P_q}$ and let $P_{q1} \vee \ldots \vee P_{qr}$ be the resulting unification problem.
3. If the current unification problem contains a unification system not in solved form then select it and go to step 1, else stop and report a success status.

The λs_e-grammar

Terms of the λs_e-**calculus** are given by:

$$\Lambda s_e \ ::= \ \underline{\mathtt{n}} \mid X \mid \Lambda s_e \Lambda s_e \mid \lambda.\Lambda s_e \mid \Lambda s_e \sigma^j \Lambda s_e \mid \varphi^i_k \Lambda s_e,$$

where $n, j, i \geq 1$, $k \geq 0$ and $X \in \mathcal{X}$.

The typing rules:

(var) $A.\Gamma \vdash \underline{1} : A$

(varn) $\dfrac{\Gamma \vdash \underline{\mathtt{n}} : B}{A.\Gamma \vdash \underline{\mathtt{n}+1} : B}$

(app) $\dfrac{\Gamma \vdash a : A \rightarrow B \quad \Gamma \vdash b : A}{\Gamma \vdash \ (a\ b) : B}$

(lambda) $\dfrac{A.\Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \rightarrow B}$

(sigma) $\dfrac{\Gamma_{\geq i} \vdash b : B \quad \Gamma_{<i}.B.\Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a\sigma^i b : A}$

(phi) $\dfrac{\Gamma_{\leq k}.\Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi^i_k a : A}$

(meta) $\Gamma \vdash X : A$, where Γ is any context.

(σ-generation)	$(\lambda.a)\ b$	$\longrightarrow$	$a\,\sigma^1\, b$
(σ-λ-transition)	$(\lambda.a)\sigma^i b$	$\longrightarrow$	$\lambda.(a\sigma^{i+1} b)$
(σ-app-transition)	$(a_1\ a_2)\sigma^i b$	$\longrightarrow$	$(a_1\sigma^i b)(a_2\sigma^i b)$
(σ-destruction)	$\underline{\mathtt{n}}\sigma^i b$	$\longrightarrow$	$\begin{cases} \underline{\mathtt{n-1}} & \text{if} \quad n > i \\ \varphi_0^i\ b & \text{if} \quad n = i \\ \underline{\mathtt{n}} & \text{if} \quad n < i \end{cases}$
(φ-λ-transition)	$\varphi_k^i(\lambda.a)$	$\longrightarrow$	$\lambda.(\varphi_{k+1}^i a)$
(φ-app-transition)	$\varphi_k^i(a_1\ a_2)$	$\longrightarrow$	$(\varphi_k^i a_1)(\varphi_k^i a_2)$
(φ-destruction)	$\varphi_k^i\underline{\mathtt{n}}$	$\longrightarrow$	$\begin{cases} \underline{\mathtt{n+i-1}} & \text{if} \quad n > k \\ \underline{\mathtt{n}} & \text{if} \quad n \leq k \end{cases}$
(Eta)	$\lambda.(a\ \underline{1})$	$\longrightarrow$	b if $a =_{s_e} \varphi_0^2 b$

(σ-σ-transition)	$(a \sigma^i b)\, \sigma^j\, c \longrightarrow (a\, \sigma^{j+1}\, c)\, \sigma^i\, (b\, \sigma^{j-i+1}\, c)$	if $i \le j$
(σ-φ-transition 1)	$(\varphi^i_k\, a)\, \sigma^j\, b \longrightarrow \varphi^{i-1}_k\, a$	if $k < j < k+i$
(σ-φ-transition 2)	$(\varphi^i_k\, a)\, \sigma^j\, b \longrightarrow \varphi^i_k (a\, \sigma^{j-i+1}\, b)$	if $k+i \le j$
(φ-σ-transition)	$\varphi^i_k (a\, \sigma^j\, b) \longrightarrow (\varphi^i_{k+1}\, a)\, \sigma^j\, (\varphi^i_{k+1-j}\, b)$	if $j \le k+1$
(φ-φ-transition 1)	$\varphi^i_k\, (\varphi^j_l\, a) \longrightarrow \varphi^j_l\, (\varphi^i_{k+1-j}\, a)$	if $l+j \le k$
(φ-φ-transition 2)	$\varphi^i_k\, (\varphi^j_l\, a) \longrightarrow \varphi^{j+i-1}_l\, a$	if $l \le k < l+j$

λs_e-unification rules (part I)

Dec-λ
$$\frac{P \wedge \lambda_A.e_1 =^? \lambda_A.e_2}{P \wedge e_1 =^? e_2}$$

Dec-App
$$\frac{P \wedge \underline{\mathtt{n}}(e_1^1, \ldots, e_p^1) =^? \underline{\mathtt{n}}(e_1^2, \ldots, e_p^2)}{P \wedge e_1^1 =^? e_1^2 \wedge \ldots \wedge e_p^1 =^? e_p^2}$$

App-Fail
$$\frac{P \wedge \underline{\mathtt{n}}(e_1^1, \ldots, e_{p_1}^1) =^? \underline{\mathtt{m}}(e_1^2, \ldots, e_{p_2}^2)}{Fail}, \text{ if } \mathtt{m} \neq \mathtt{n}.$$

λs_e-unification rules (part II)

Exp-λ

$$\frac{P}{\exists Y : (A.\Gamma \vdash B), P \wedge X =^?_{\lambda\sigma} \lambda_A Y}$$

if $(X : \Gamma \vdash A \rightarrow B) \in \mathcal{TV}ar(P)$, $Y \notin \mathcal{TV}ar(P)$, and X is not a solved variable.

Replace

$$\frac{P \wedge X =^?_{\lambda s_e} t}{\{X/t\}(P) \wedge X =^?_{\lambda s_e} t}$$ if $X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t)$ and if $t \in \mathcal{X} \Rightarrow t \in \mathcal{TV}ar(P)$.

Normalise

$$\frac{P \wedge e_1 =^?_{\lambda s_e} e_2}{P \wedge e_1' =^?_{\lambda\sigma} e_2'}$$ if e_1 or e_2 is not in long form.

where e_1' (resp. e_2') is the long form of e_1 (resp. e_2), if e_1 (resp. e_2) is not solved and e_1 (resp. e_2) otherwise.

λs_e-unification rules (part III)

Exp-App

$$\frac{P \wedge \psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} \underline{\mathtt{m}}(b_1, \ldots, b_q)}{P \wedge \psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda\sigma} \underline{\mathtt{m}}(b_1, \ldots, b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \ldots \exists H_k, X =^?_{\lambda s_e} \underline{\mathtt{r}}(H_1, \ldots, H_k)}$$

if $\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p)$ is the skeleton of a λs_e normal term, and X has an atomic type and is not solved, where $H_1, \ldots, H_k$ are meta-variables of appropriate types, not occurring in P, with the contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{i_1, \ldots, i_p\}$ of superscripts of the σ operator such that $\underline{\mathtt{r}}(H_1, \ldots, H_k)$ has the right type, $R_i = \bigcup_{k=0}^{p}$ if $i_k \geq m + p - k - \Sigma_{l=k+1}^{p} j_l > i_{k+1}$ then $\{m + p - k - \Sigma_{l=k+1}^{p} j_l\}$ else $\emptyset$, where $i_0 = \infty$ and $i_{p+1} = 0$.

SIMPL$_{\lambda s_e}$

`INPUT`: A unification problem P_q with at least one rigid-rigid equation.
`OUTPUT`: A terminal (failure or success) status or an equivalent unification problem $\overline{P_q}$ without rigid-rigid equations and containing at least one flexible-rigid equation.

Assume that **Dec-λ** is applied eagerly.
`WHILE` there exists a rigid-rigid equation in P_q `DO`:

1. Apply **Dec-App-λ** or **App-Fail**.
2. Apply **Dec-App** and, if the resulting unification problem contains a flexible-rigid equation, call it $\overline{P_q}$ and give $\overline{P_q}$ as result, else stop and report a success status.

`DONE`.

MATCH$_{\lambda s_e}$

INPUT: A unification system P_q with at least one flexible-rigid equation.
OUTPUT: A disjunction of equivalent unification systems, written $P_{q1} \vee \ldots \vee P_{qk}$.
Assume that **Dec-λ** is applied eagerly.

1. Apply **Exp-λ** and **Replace** as much as possible to the selected flexible-rigid equation and call P'_q the resulting unification system.
2. Apply **Exp-App** and **Replace** and **Normalise** to P'_q and call $P_{q1} \vee \ldots P_{qr}$ the resulting unification problem.

The Main Procedure

INPUT: A unification system P_ϵ.
OUTPUT: A success or a failure status and in the former case the solutions are the solved equations whose left-hand side corresponds the meta-variables of the initial problem. If the initial problem is non-unifiable the algorithm may not terminate.

1. If P_q contains a rigid-rigid equation then apply SIMPL$_{\lambda s_e}$ to it, else if P_q contains a non-solved flexible-rigid equation then rename it to $\overline{P_q}$ and go to the next step.
2. Apply MATCH$_{\lambda s_e}$ to $\overline{P_q}$ and let $P_{q1} \vee \ldots \vee P_{qr}$ be the resulting unification problem.
3. If the current unification problem contains a unification system not in solved form then select it and go to step 1, else stop and report a success status.

Corresponding equations

Definition

Let $X =^?_{\lambda\sigma} a$ and $X =^?_{\lambda s_e} a'$ be two flexible-rigid equations in the $\lambda\sigma$- and λs_e-calculus respectively. These equations are said to be *corresponding (or associated)* if a and a' have the same heading.

Translating λs_e-terms into $\lambda\sigma$-terms

Definition

The operator $T : \Lambda_{\lambda s_e} \to \Lambda_{\lambda\sigma}$ is defined inductively as:

1. $T(X) = X$
2. $T(\underline{n}) = \underline{1}[\uparrow^{n-1}]$
3. $T(a\ b) = T(a)\ T(b)$
4. $T(\lambda.a) = \lambda.T(a)$
5. $T(a\sigma^i b) = T(a)[\underline{1}.\underline{2}.\cdots.\underline{i-1}.T(b)[\uparrow^{i-1}].\uparrow^{i-1}]$, where $i \geq 1$.
6. $T(\varphi^i_k(a)) = T(a)[\underline{1}.\underline{2}.\cdots.\underline{k}.\uparrow^{k+i-1}]$, where $k \geq 0$ and $i \geq 1$.

If $r = 0$ in the list $\underline{1}.\cdots.\underline{r}$, then it is to be interpreted as the empty list. In addition, $\uparrow^0 = id$.

Preservation of types by T

Theorem

Let Γ be a context, A a type and a a term in the language of the λs_e-calculus such that $\Gamma \vdash a : A$. Then $\Gamma \vdash T(a) : A$.

λs_e-skeleton

Definition (**Ayala & Kamareddine 2001**)

Let t be a λs_e-normal term whose root operator is either σ or φ and let X be its leftmost innermost meta-variable. Denote by $\psi_{i_k}^{j_k}$ the k-th operator following the sequence of operators σ and φ, considering only left arguments of the σ operators, in the innermost outermost ordering. Additionally, if $\psi_{i_k}^{j_k}$ corresponds to an operator φ then j_k and i_k denote its superscripts and subscripts, respectively, and if $\psi_{i_k}^{j_k}$ corresponds to an operator σ then $j_k = 0$ and i_k denote its superscript. Let a_k denote the corresponding right argument of the k-th operator if $\psi_{i_k}^{j_k} = \sigma^{i_k}$ and the empty argument if $\psi_{i_k}^{j_k} = \varphi_{i_k}^{j_k}$. The *skeleton* of t, written as $sk(t)$, is $\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p)$.

Correspondence from MATCH$_{\lambda s_e}$ to MATCH$_{\lambda\sigma}$

Theorem

Let $\psi^{j_p}_{i_p} \ldots \psi^{j_1}_{i_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} (\underline{\mathtt{m}}\ b_1 \ldots b_q)$ *be a flexible-rigid equation in the* λs_e*-calculus, where* X *has atomic type. Then, for each equation generated by the rule* **Exp-App**$_{\lambda s_e}$ *there exists a corresponding equation in the* $\lambda\sigma$*-calculus generated by the rule* **Exp-App**$_{\lambda\sigma}$ *for the equation*

$$T(\psi^{j_p}_{i_p} \ldots \psi^{j_1}_{i_1}(X, a_1, \ldots, a_p)) =^?_{\lambda\sigma} T(\underline{\mathtt{m}}\ b_1 \ldots b_q)$$

Example

Consider the unification problem
$\varphi^4_1(((\varphi^3_7 X)\sigma^5 a)\sigma^3 b) =^?_{\lambda s_e} (\underline{6}\ b_1 \ldots b_q)$
The **Exp-App**$_{\lambda s_e}$ rule generates the equation $X =^?_{\lambda s_e} (\underline{4}\ H_1 \ldots H_q)$.
The $\lambda\sigma$-normal form of $T(\varphi^4_1(((\varphi^3_7 X)\sigma^5 a)\sigma^3 b))$ is computed by:
$T(\varphi^4_1(((\varphi^3_7 X)\sigma^5 a)\sigma^3 b)) =$
$T(((\varphi^3_7 X)\sigma^5 a)\sigma^3 b)[\underline{1}.\uparrow^4] =$
$T((\varphi^3_7 X)\sigma^5 a)[\underline{1}.\underline{2}.T(b)[\uparrow^2].\uparrow^2][\underline{1}.\uparrow^4] \rightarrow^*_\sigma$
$T((\varphi^3_7 X)\sigma^5 a)[\underline{1}.\underline{5}.T(b)[\uparrow^5].\uparrow^5] =$
$T(\varphi^3_7 X)[\underline{1}.\underline{2}.\underline{3}.\underline{4}.T(a)[\uparrow^4].\uparrow^4])[\underline{1}.\underline{5}.T(b)[\uparrow^5].\uparrow^5] \rightarrow^*_\sigma$
$T(\varphi^3_7 X)[\underline{1}.\underline{5}.T(b)[\uparrow^5].\underline{6}.T(a)[\uparrow^6].\uparrow^6] =$
$X[\underline{1}.\underline{2}.\underline{3}.\underline{4}.\underline{5}.\underline{6}.\underline{7}.\uparrow^9][\underline{1}.\underline{5}.T(b)[\uparrow^5].\underline{6}.T(a)[\uparrow^6].\uparrow^6] \rightarrow^*_\sigma$
$X[\underline{1}.\underline{5}.T(b)[\uparrow^5].\underline{6}.T(a)[\uparrow^6].\underline{7}.\underline{8}.\uparrow^{10}]$

Example

The rule **Exp-App**$_{\lambda\sigma}$ generates the corresponding equation $X =^?_{\lambda\sigma} \underline{4}(Y_1 \ldots Y_q)$ which corresponds to the selection of the de Bruijn index $\underline{6}$ inside the explicit substitution $[\underline{1}.\underline{5}.T(b)[\uparrow^5].\underline{6}.T(a)[\uparrow^6].\underline{7}.\underline{8}.\uparrow^{10}]$.

Translating $\lambda\sigma$-terms into λs_e-terms

Definition

The operator $L : \Lambda_{\lambda\sigma-\mathtt{terms}} \rightarrow \Lambda_{\lambda s_e}$ is defined inductively as:
$L(X) = X$
$L(\underline{1}[\uparrow^{m-1}]) = \underline{\mathtt{m}}$, where $m \in \mathbb{N}$
$L(a\ b) = L(a)\ L(b)$
$L(\lambda.a) = \lambda.L(a)$
$L(a[a_1.a_2.\cdots.a_p.\uparrow^n]) =$
$\sigma^1 \ldots \sigma^{p-1}\sigma^p\varphi_p^{n+1}(L(a), L(a_p), L(a_{p-1}), \ldots, L(a_2), L(a_1))$, where $a_1.a_2.\cdots.a_p.\uparrow^n$ is a substitution in $\lambda\sigma$-normal form, and $n, p \geq 0$.

Preservation of types by L

Theorem

Let Γ be a context, A a type and a a term in the language of the $\lambda\sigma$-calculus such that $\Gamma \vdash a : A$. Then $\Gamma \vdash L(a) : A$.

Correspondence from MATCH$_{\lambda\sigma}$ to MATCH$_{\lambda s_e}$

Theorem

Let $X[a_1.\cdots.a_p.\uparrow^n] =^?_{\lambda\sigma} (\underline{\mathrm{m}}\ b_1 \ldots b_q)$ *be a flexible-rigid equation in the* $\lambda\sigma$*-calculus, where* X *has atomic type and* $a_1.\cdots.a_p.\uparrow^n$ *is a* $\lambda\sigma$*-normal substitution. Then, for each equation generated by the rule* **Exp-App**$_{\lambda\sigma}$ *there exists a corresponding equation in the* λs_e*-calculus generated by the rule* **Exp-App**$_{\lambda s_e}$ *for the equation*

$$L(X[a_1.\cdots.a_p.\uparrow^n]) =^?_{\lambda s_e} (\underline{\mathrm{m}}\ L(b_1)\ldots L(b_q))$$

Correspondence between MATCH$_{\lambda s_e}$ and MATCH$_{\lambda\sigma}$

Theorem

Let P be a unification problem in the simply typed λ-calculus, and P_ξ its precooking translation to the ξ-calculus of explicit substitutions, where $\xi \in \{\lambda\sigma, \lambda s_e\}$. Then $P_{\lambda\sigma}$ is unifiable if and only if $P_{\lambda s_e}$ is unifiable. Moreover, whenever unifiers exist, they are associated.

$\lambda\sigma$ and λs_e correspondence

Corollary

Let $P_{\lambda\sigma}$ be a unification problem in the $\lambda\sigma$-calculus of explicit substitutions, and $L(P_{\lambda\sigma})$ its translation to the λs_e-calculus of explicit substitutions. Then $P_{\lambda\sigma}$ is unifiable if and only if $L(P_{\lambda\sigma})$ is unifiable. Moreover, whenever unifiers exists, they are associated.

Conclusion

- In this work we compared the $\lambda\sigma$- and the λs_e-styles of unification.
- To do so, we presented the *unification tree* notation which allows a clear presentation of the Huet's algorithm in de Bruijn notation.
- This notation was applied to unification problems in de Bruijn notation, but it can be applied to λ-terms with names with minor modifications.

Conclusion

- We compared the classical method of Huet for HOU and the one of Dowek, Hardin and Kirchner for the $\lambda\sigma$-calculus.
- We described the counterpart of the procedures SIMPL and MATCH, called SIMPL$_{\lambda\sigma}$ and MATCH$_{\lambda\sigma}$.
- We concluded that there exists a correspondence between the substitutions generated by Huet's algorithm and the graftings generated by the $\lambda\sigma$-HOU algorithm for unification problems which are in the image of the precooking translation.
- This comparison was extended to the λs_e-HOU algorithm.

Conclusion

- We concluded that the $\lambda\sigma$- and the λs_e-HOU algorithms generate associated graftings.