(Higher-Order) Unification via λs_e-style of explicit substitution

Mauricio Ayala-Rincón
Departamento de Matemática
Universidade de Brasília
Brasília D. F., Brasil

Fairouz Kamareddine
Computer and Electrical Engineering
Heriot-Watt University
Edinburgh, Scotland

ZIC, Eindhoven the Netherlands — October 3, 2000

Talk's Plan

1. What is HOU?
2. HOU in explicit substitution calculi
3. Unification in the λs_e-style of explicit substitution
4. Checking arithmetic constraints (versus shifts and composition in $\lambda\sigma$)
5. Strategies for λs_e-unification
6. Translations between the pure λ-calculus and the λs_e-calculus
7. Related work, Future work and Conclusions

What is HOU?

- Higher order objects arise naturally in many fields of computer science.
- $\textsc{map}(f, \textsc{nil}) \to \textsc{nil}$; $\textsc{map}(f, \textsc{cons}(x, l)) \to \textsc{cons}(f(x), \textsc{map}(f, l))$
- Observe that f appears both as a variable and as a functional symbol.
- The function $\textsc{map}$ is a typical example of a second-order function.
- useful third- until sixth-order functions were presented in the context of combinator parsing. In Okasaki'97

Examples of HOU

- $F(f(a)) = f(F(a))$.
- Solution is: $\{F/\lambda_x.x\}$
- Other solutions are: $\{F(x)/f^n(x) \mid n \in \mathbb{N}\}$.
- Robinson, Huet ('75)
- Huet's algorithm is a semi-decision one that may never stop when the input unification problem has no unifiers, but when the problem has a solution it always presents an explicit unifier.

HOU's status

- Unification for second-order logic was proved undecidable in general by Goldfarb in '81.
- Goldfarb's proof is based on a reduction from Hilbert's Tenth Problem.
- For the second-order case, unification is decidable, when the language is restricted to monadic functions (Farmer '88).
- Other problems of HOU include that most general unifiers may not exist.
- Huet has shown that equations of the form $(\lambda_x.F \ \ a) =^? (\lambda_x.G \ \ b)$ (called *flex-flex*) of third-order may not have MGUs.

HOU in explicit substitution calculi

$$\boxed{\text{HOU}\left\{\begin{array}{c}\text{Given two simply-typed lambda terms } a \text{ and } b\\ \text{find a } \mathit{substitution}\ \theta \text{ such that}\\ \theta(a) =_{\beta\eta} \theta(b)\end{array}\right.}$$

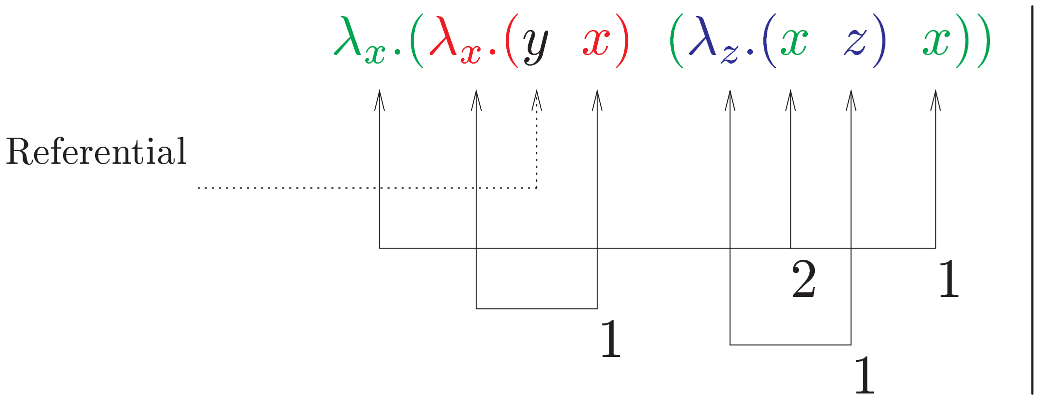
- HOU essential for generalizations of the Robinson's first-order resolution principle.

- HOU applied in $\left\{\begin{array}{l}\text{– Automated (Higher order) reasoning}\\ \text{– Higher order proof assistants}\\ \text{– Higher order logic programming}\end{array}\right.$

Why *making substitutions explicit* is adequate for reasoning about HOU?

- Substitution is the key operation for HOU.
- *Implicitness* of substitution is the "Achilles heel" of the λ-calculus:
  - β-reduction is given via informal/implicit variable renaming
- $\boxed{\text{Implicit substitution does not provide any formal mechanism for analysing essential computational properties}}$ such as $\left\{ \begin{array}{l} \text{– time and} \\ \text{– space complexity} \end{array} \right.$

• Terms in de Bruijn notation, $\Lambda_{dB}(\mathcal{X})$: $a ::= \mathbb{N} \mid \mathcal{X} \mid (a\ \ a) \mid \lambda.a$, where $\mathcal{X}$ meta-variables and $\mathbb{N}$ set of de Bruijn indices.


$\lambda_x.(\lambda_x.(y\ \ x)\ \ (\lambda_z.(x\ \ z)\ \ x))$
Referential
2
1
1
1

$$\to_\beta \ \lambda_x.(y\ \ (\lambda_z.(x\ \ z)\ \ x))$$
$$\to_\beta \ \lambda_x.(y\ \ (x\ \ x))$$

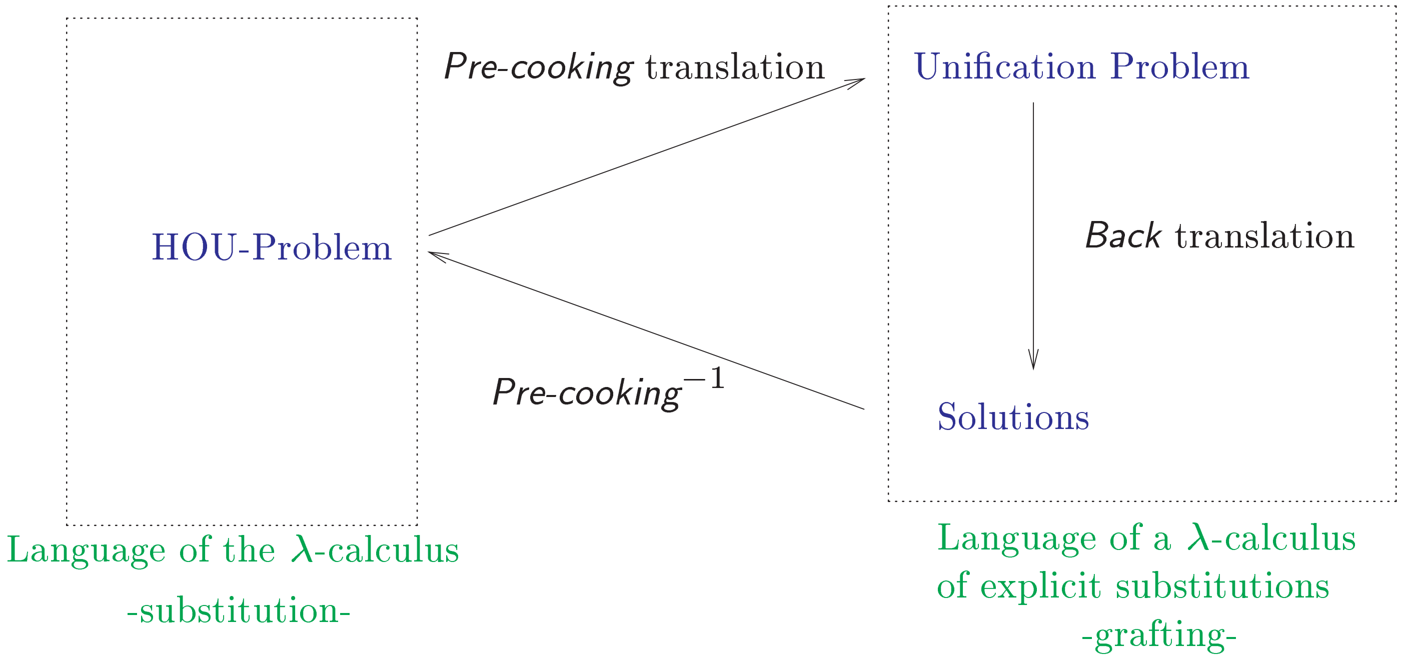
For instance, for the referential $x, y, z, \ldots$: $\boxed{\lambda.(\lambda.(4\ \ 1)\ \ (\lambda.(2\ \ 1)\ \ 1))}$

β-reduction: $\boxed{\lambda.(\lambda.(4\ \ 1)\ \ (\lambda.(2\ \ 1)\ \ 1)) \to_\beta \lambda.(3\ \ (\lambda.(2\ \ 1)\ \ 1)) \to_\beta \lambda.(3\ \ (1\ \ 1))}$

- Higher order *substitution*: $\boxed{\{X/1\}(\lambda.(1\ \ X)\ \ X) = (\lambda.(1\ \ 2)\ \ 1)}$

$$\boxed{\begin{array}{ccc}
\text{substitution} & \neq & \textit{grafting} \\
\{X/a\}(\lambda.X) & & (\lambda.X)\{X/a\} \\
\| & & \| \\
\lambda.\{X/a^+\}X & & \lambda.X\{X/a\} \\
\| & & \| \\
\lambda.\underbrace{a^+}_{\text{lift}} & \neq & \lambda.a
\end{array}}$$

$$\boxed{\begin{array}{c}
\beta\text{-reduction} \\
(\lambda.a\ \ b) \rightarrow \{1/b\}a
\end{array}}$$


Pre-cooking translation
Unification Problem
HOU-Problem
Back translation
Pre-cooking^{-1}
Solutions
Language of the λ-calculus
-substitution-
Language of a λ-calculus
of explicit substitutions
-grafting-

- Introduced by G. Dowek, T. Hardin and C. Kirchner using the $\lambda\sigma$-calculus.
- Subsumes Huet's HOU method.

De Bruijn's λ-calculus

$$a ::= \mathtt{n} \mid X \mid (a \;\; a) \mid \lambda.a \qquad \text{where } X \in \mathcal{X} \text{ and } \mathtt{n} \in \mathbb{N} \setminus \{0\}.$$

The *updating functions* $U^i_k : \Lambda_{dB}(\mathcal{X}) \to \Lambda_{dB}(\mathcal{X})$ for $k \geq 0$ and $i \geq 1$ are defined inductively by:

$$\begin{array}{l}
U^i_k(a \;\; b) = (U^i_k(a) \;\; U^i_k(b)) \\
U^i_k(\lambda.a) = \lambda.U^i_{k+1}(a) \\
U^i_k(X) = X, \quad X \in \mathcal{X} \\
U^i_k(\mathtt{n}) = \begin{cases} \mathtt{n+i-1} & \text{if} \quad n > k \\ \mathtt{n} & \text{if} \quad n \leq k\,. \end{cases}
\end{array}$$

The *meta-substitutions at level* j, for $j \geq 1$, of a term $b \in \Lambda$ in a term $a \in \Lambda_{dB}(\mathcal{X})$, denoted $a\{\!\{\mathtt{j} \leftarrow b\}\!\}$, is defined inductively on a as follows:

$$\begin{aligned}
(a_1\ \ a_2)\{\!\{\mathtt{j} \leftarrow b\}\!\} &= (a_1\{\!\{\mathtt{j} \leftarrow b\}\!\}\ \ a_2\{\!\{\mathtt{j} \leftarrow b\}\!\}) \\
(\lambda.a)\{\!\{\mathtt{j} \leftarrow b\}\!\} &= \lambda.a\{\!\{\mathtt{j}+\mathtt{1} \leftarrow b\}\!\} \\
X\{\!\{\mathtt{j} \leftarrow b\}\!\} &= X, \quad X \in \mathcal{X} \\
\mathtt{n}\{\!\{\mathtt{j} \leftarrow b\}\!\} &= \begin{cases} \mathtt{n}-\mathtt{1} & \text{if } n > j \\ U_0^j(b) & \text{if } n = j \\ \mathtt{n} & \text{if } n < j\,. \end{cases}
\end{aligned}$$

Properties of meta-substitutions and updating

Let $a, b, c \in \Lambda_{dB}(\mathcal{X})$. We have:

1. for $k < n < k + i$: $U_k^{i-1}(a) = U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\}$.
2. for $l \leq k < l + j$: $U_k^i(U_l^j(a)) = U_l^{j+i-1}(a)$.
3. for $k + i \leq n$: $U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U_k^i(a\{\!\{\mathtt{n} - \mathtt{i} + \mathtt{1} \leftarrow b\}\!\})$.
4. for $i \leq n$: $a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n} + \mathtt{1} \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n} - \mathtt{i} + \mathtt{1} \leftarrow c\}\!\}\}\!\}$.
5. for $l + j \leq k + 1$: $U_k^i(U_l^j(a)) = U_l^j(U_{k+1-j}^i(a))$.
6. for $n \leq k + 1$: $U_k^i(a\{\!\{\mathtt{n} \leftarrow b\}\!\}) = U_{k+1}^i(a)\{\!\{\mathtt{n} \leftarrow U_{k-n+1}^i(b)\}\!\}$.

beta and eta rules

- (β-rule) $(\lambda.a \ \ b) \to_\beta a\{\!\{1 \leftarrow b\}\!\}$
- (η-rule) $\lambda.(a \ \ 1) \to_\eta b$ if $\exists b\, U_0^2(b) = a$

λs_e calculus

• Terms in λs_e: $a ::= \mathcal{X} \mid \mathbb{N} \mid (a\ \ a) \mid \lambda.a \mid a\sigma^j a \mid \varphi^i_k a$, for $j,\, i \geq 1\,,\quad k \geq 0$
where $\mathcal{X}$ *meta-variables* and $\mathbb{N}$ set of de Bruijn indices.

Table 1: The Rewriting System of the λs-calculus with η-rule

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

Table 2: The Rewriting System of the λs_e-calculus without rules in Table 1

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

Origins of λs_e-calculus

- (σ-generation rule:)

  $(t_1\delta)(t_2\lambda) \to_\sigma (t_1\delta)(t_2\lambda)(t_1\sigma^{(1)})$

- (σ-transition rules:)

$$\begin{array}{lll} (t_1\sigma^{(i)})(t_2\lambda) \to_\sigma ((t_1\sigma^{(i)})t_2\lambda)(t_1\sigma^{(i+1)}) & & (\sigma_{01\lambda} - transition) \\ (t_1\sigma^{(i)})(t_2\delta) \to_\sigma ((t_1\sigma^{(i)})t_2\delta)(t_1\sigma^{(i)}) & & (\sigma_{01\delta} - transition) \end{array}$$

- (σ-destruction rules:)

  $(t_1\sigma^{(i)})i \to_\sigma \mathtt{ud}^{(i)}(t_1)$

  $(t_1\sigma^{(i)})x \to_\sigma x$ if $x \neq i$.

$\lambda\sigma$-calculus

$\textsc{terms}\ a \ ::= \ 1 \mid X \mid (a \ \ a) \mid \lambda a \mid a[s]$ and $\textsc{subs}\ s \ ::= \ id \mid \uparrow \mid a.s \mid s \circ s.$

Table 3: The $\lambda\sigma$ Rewriting System of the $\lambda\sigma$-calculus

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

Unification rules

Table 4: The Boolean simplification rules for unification problems

(Trivial)	$P \wedge s =^? s$	$\rightarrow$	P
(AndIdem)	$P \wedge e \wedge e$	$\rightarrow$	$P \wedge e$
(OrIdem)	$P \vee e \vee e$	$\rightarrow$	$P \vee e$
(SimpAndT)	$P \wedge \mathbb{T}$	$\rightarrow$	P
(SimpAndF)	$P \vee \mathbb{F}$	$\rightarrow$	$\mathbb{F}$
(SimpOrT)	$P \vee \mathbb{T}$	$\rightarrow$	$\mathbb{T}$
(SimpOrF)	$P \vee \mathbb{F}$	$\rightarrow$	P
(Distrib)	$P \wedge (Q \vee R)$	$\rightarrow$	$(P \wedge Q) \vee (P \wedge R)$
(Propag)	$\exists \vec{z}(P \vee Q)$	$\rightarrow$	$\exists \vec{z} P \vee \exists \vec{z} Q$
(ElimQE)	$\exists z P$	$\rightarrow$	P, if $z \notin var(P)$
(ElimBV)	$\exists z \quad z =^? t \wedge P$	$\rightarrow$	P, if $z \notin var(P) \cup var(t)$

Table 5: The λs_e-unification rules

(Dec-λ)	$P \wedge \lambda_A.a =^?_{\lambda s_e} \lambda_A.b \quad \to \quad P \wedge a =^?_{\lambda s_e} b$
(Dec-App)	$P \wedge (\mathtt{n}\, a_1 \ldots a_p) =^?_{\lambda s_e} (\mathtt{n}\, b_1 \ldots b_p) \quad \to \quad P \bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$
(App-Fail)	$P \wedge (\mathtt{n}\, a_1 \ldots a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \ldots b_q) \quad \to \quad \mathbb{F}$ if $\mathtt{n} \neq \mathtt{m}$
(Dec-σ)	$P \wedge a\sigma^i b =^?_{\lambda s_e} c\sigma^i d \quad \to \quad P \wedge a =^?_{\lambda s_e} c \wedge b =^?_{\lambda s_e} d$
(σ-Fail)	$P \wedge a\sigma^i b =^?_{\lambda s_e} c\sigma^j d \quad \to \quad \mathbb{F}$
	if $i \neq j$ and $a\sigma^i b =^?_{\lambda s_e} c\sigma^j d$ is not *flex-flex*
(Dec-φ)	$P \wedge \varphi^i_k a =^?_{\lambda s_e} \varphi^i_k b \quad \to \quad P \wedge a =^?_{\lambda s_e} b$
(φ-Fail)	$P \wedge \varphi^i_k a =^?_{\lambda s_e} \varphi^j_l b \quad \to \quad \mathbb{F}$
	if $i \neq j$ or $k \neq l$ and $\varphi^i_k a =^?_{\lambda s_e} \varphi^j_l b$ is not *flex-flex*
(Exp-λ)	$P \quad \to \quad \exists(Y$ where $A.\Gamma \vdash Y : B), P \wedge X =^?_{\lambda s_e} \lambda_A.Y$
	if $(\Gamma \vdash X : A \to B) \in \mathcal{T}var(P), Y \notin \mathcal{T}var(P)$, and X is a unsolved variable

Table 6: The λs_e-unification rules

(Exp-App) $P \wedge \psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \ldots b_q) \quad \to$

$$P \wedge \psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \ldots b_q) \quad \wedge$$

$$\bigvee_{r \in R_p \cup R_i} \exists H_1, \ldots, H_k, X =^?_{\lambda s_e} (\mathtt{r}\, H_1 \ldots H_k)$$

- 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
- X has an atomic type and is not solved
- $H_1, \ldots, H_k$ are variables of appropriate types, not occurring in P, with the environments $\Gamma_{H_i} = \Gamma_X$
- $R_p \subseteq \{i_1, \ldots, i_p\}$ of superscripts of the σ operator such that $(\mathtt{r}\, H_1 \ldots H_k)$ has the right type $R_i = \begin{cases} \bigcup_{j=0}^{p}\{q\} & \text{if } q > i_{k+1} \\ \emptyset & \text{otherwise} \end{cases}$

and $q = m + p - k - \sum_{l=k+1}^{p} j_l, \quad i_0 = \infty, \quad i_{p+1} = 0$

Table 7: The λs_e-unification rules

(Replace)	$P \wedge X =^?_{\lambda s_e} a \quad \rightarrow \quad \{X/a\}P \wedge X =^?_{\lambda s_e} a$
	if $X \in \mathcal{T}var(P), X \notin \mathcal{T}var(a)$ and $a \in \mathcal{X} \Rightarrow a \in \mathcal{T}var(P)$
(Normalize)	$P \wedge a =^?_{\lambda s_e} b \quad \rightarrow \quad P \wedge a' =^?_{\lambda s_e} b'$
	if a or b is not in long normal form,
	$a' = \begin{cases} \text{the long normal form of } a & \text{if } a \text{ is an unsolved variable} \\ a & \text{otherwise} \end{cases}$
	b' is defined from b similarly to a' from a.

Unification in the λs_e-style of explicit substitution

- A λs_e-**unification problem** P is: $\left\{ \bigvee_{j \in J} \quad \underbrace{\exists \vec{w}_j \bigwedge_{i \in I_j} s_i =^?_{\lambda s_e} t_i}_{\text{unification system}} \right.$

- A **unifier** of $\underbrace{\exists \vec{w} \bigwedge_{i \in I} s_i =^?_{\lambda s_e} t_i}_{\text{unification system}}$ is a **grafting** σ such that $\boxed{\exists \vec{w} \bigwedge_{i \in I} s_i\sigma = t_i\sigma}$

$$
\begin{array}{lrcl}
\textbf{Example :} & (\lambda.(\lambda.(X\ \ 2)\ \ 1)\ \ Y) & =^?_{\lambda s_e} & (\lambda.(Z\ \ 1)\ \ U) \\
 & & \downarrow & X, Z : A \to A;\ Y, U : A \\
\textit{Normalize} & ((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) & =^?_{\lambda s_e} & (Z\sigma^1 U\ \ \varphi^1_0 U) \\
 & & \downarrow & \\
\textit{Dec-App} & (X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 U & \wedge & \varphi^1_0 Y =^?_{\lambda s_e} \varphi^1_0 U \\
 & & \downarrow & \\
\textit{Dec-}\varphi & (X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 U & \wedge & Y =^?_{\lambda s_e} U \\
 & & \downarrow & \\
\textit{Replace} & (X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 Y & \wedge & Y =^?_{\lambda s_e} U \\
 & & \downarrow_* & \\
\begin{array}{l}\textit{Exp-}\lambda\ + \\ \textit{Replace}\end{array} & ((\lambda.X')\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} (\lambda.Z')\sigma^1 Y & \wedge & \left\{\begin{array}{l} Y =^?_{\lambda s_e} U \\ X =^?_{\lambda s_e} \lambda.X' \\ Z =^?_{\lambda s_e} \lambda.Z' \end{array}\right. \\
 & & \downarrow_* & \\
\begin{array}{l}\textit{Normalize} + \\ \textit{Dec-}\lambda\end{array} & (X'\sigma^3 Y)\sigma^2(\varphi^1_0 Y) =^?_{\lambda s_e} Z'\sigma^2 Y & \wedge & \left\{\begin{array}{l} Y =^?_{\lambda s_e} U \\ X =^?_{\lambda s_e} \lambda.X' \\ Z =^?_{\lambda s_e} \lambda.Z' \end{array}\right.
\end{array}
$$

$$\left.\begin{array}{ll} \bullet\ \textit{Solved}\text{ equations:} & \left\{\begin{array}{l} Y =^?_{\lambda s_e} U \\ X =^?_{\lambda s_e} \lambda.X' \\ Z =^?_{\lambda s_e} \lambda.Z' \end{array}\right. \\ \bullet\ \textit{Flex-Flex}\text{ equations:} & (X'\sigma^3 Y)\sigma^2(\varphi^1_0 Y) =^?_{\lambda s_e} Z'\sigma^2 Y \end{array}\right\} \textit{Solved Forms}$$

• Solutions: $\{Y/X_1, U/X_1\} \;\bigcup\;$ solutions for X and Z given by the *Flex-Flex* equation.

Take, for instance, $\{Y/X_1, U/X_1\} \;\bigcup\; \{X/\lambda.\mathtt{n}+1, Z/\lambda.\mathtt{n}\}$ with $n > 2$:
$\underline{(\lambda.(\lambda.(\lambda.\mathtt{n}+1\ \ 2)\ \ 1)\ \ X_1)} \rightarrow_\beta (\lambda.(\lambda.\mathtt{n}\ \ 2)\ \ X_1) \rightarrow_\beta (\lambda.\mathtt{n}-1\ \ X_1) \rightarrow_\beta \underline{\mathtt{n}-2}$
and
$\underline{(\lambda.(\lambda.\mathtt{n}\ \ 1)\ \ X_1)} \rightarrow_\beta (\lambda.\mathtt{n}-1\ \ X_1) \rightarrow_\beta \underline{\mathtt{n}-2}$

- Correctness: If P reduces to P' then every unifier of P' is a unifier of P.
- Completeness: If P reduces to P' then every unifier of P is a unifier of P'.

Theorem [Correctness and Completeness]

The λs_e-unification rules are correct and complete.

3. Checking arithmetic constraints (versus shifts and composition in $\lambda\sigma$)

λs_e-calculus and λ-calculus $\rightarrow$ $\left.\begin{array}{r}\textit{Term}\\ \textit{Substitution}\end{array}\right\}$ objects $\lambda\sigma$-calculus

λs_e uses all de Bruijn indices: $\mathbb{N}$

$\lambda\sigma$ uses only $\mathtt{1}$, "shift" and "composition": $\mathtt{n} \equiv \mathtt{1}[\underbrace{\uparrow \circ \cdots \circ \uparrow}_{n-1}]$

Exp-App $\lambda\sigma$-unification rule

$$P \wedge X[a_1 \ldots a_p . \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\, b_1 \ldots b_q) \quad \rightarrow$$

$$\bigwedge \begin{cases} P \\ X[a_1 \ldots a_p . \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\, b_1 \ldots b_q) \\ \bigvee_{r \in R_p \cup R_i} \exists H_1 \ldots H_k, X =^?_{\lambda\sigma} (\mathtt{r}\; H_1 \ldots H_k) \end{cases}$$

X not solved and atomic; $H_1, \ldots, H_k$ variables of appropriate types; $\Gamma_{H_i} = \Gamma_X$, $R_p \subseteq \{1, \ldots, p\}$ such that $(\mathtt{r}\; H_1 \ldots H_k)$ has the right type,
$R_i =$ if $m \geq n+1$ then $\{m - n + p\}$ else $\emptyset$

Exp-App λs_e-unification rule

$$P \wedge \psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \ldots b_q) \quad \to$$

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

$\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p)$ skeleton of a λs_e-normal term; X atomic and not solved; $\Gamma_{H_i} = \Gamma_X$, $R_p \subseteq \{i_1, \ldots, i_p\}$ of superscripts of the σ operator such that $(\mathtt{r}\ H_1 \ldots H_k)$ has the right type, $R_i = \bigcup_{k=0}^{p}$ if $i_k \geq m + p - k - \sum_{l=k+1}^{p} j_l > i_{k+1}$ then $\{m + p - k - \sum_{l=k+1}^{p} j_l\}$ else $\emptyset$, where $i_0 = \infty, i_{p+1} = 0$

In the $\lambda\sigma$-calculus

$\boxed{X[a_1 \dots a_p.\uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\, b_1 \dots b_q)}$ has solutions of the form:

$$\left(\mathtt{1}[\underbrace{\uparrow \circ \cdots \circ \uparrow}_{r-1}] \quad \underbrace{H_1 \quad \dots \quad H_k}_{\text{of appropriate type}} \right)$$

$$\mathtt{1}[\underbrace{\uparrow \circ \cdots \circ \uparrow}_{r-1}] \quad [a_1 \dots a_p.\uparrow^n] = \begin{cases} a_i, \text{ if } 1 \le r = i \le p \\ \mathtt{1}[\underbrace{\uparrow \circ \cdots \circ \uparrow}_{r-1-p}] \quad [\underbrace{\uparrow \circ \cdots \circ \uparrow}_{n}] \quad \textit{otherwise.} \end{cases}$$

In the λs_e-calculus

$\boxed{\psi_{k_p}^{j_p} \ldots \psi_{k_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \ldots b_q)}$ solutions of the form:

$$\left(\mathtt{n} \quad \underbrace{H_1 \quad \ldots \quad H_k}_{\text{of appropriate type}} \right)$$

such that for some i, $\left[\begin{array}{c} k_{i+1} < n \leq k_i \\ \text{and} \\ n - (p - i) + \sum_{r=i+1}^{p} j_r = m \end{array} \right]$

4. Translations between the pure λ-calculus and the λs_e-calculus

- A unifier of $\lambda.X =_{\beta\eta} \lambda.a$ is not a $\{X/b\}$ such that $b =_{\beta\eta} a$:

$$\{X/b\}(\lambda.X) = \lambda.(\{X/b^+\}X) = \lambda.(X\{X/b^+\}) = \lambda.b^+$$

- The **pre-cooking** of a λ-term in de Bruijn notation into the λs_e-calculus is defined by $\boldsymbol{a_{pc}} = PC(a, 0)$ where $PC(a, n)$ is defined by:

1. $PC(\lambda_B.a, n) = \lambda_B.PC(a, n+1)$
2. $PC((a\ \ b), n) = (PC(a, n)\ \ PC(b, n))$
3. $PC(\mathtt{k}, n) = \mathtt{k}$
4. $PC(X, n) = \begin{cases} \text{if } n = 0 \text{ then } X \\ \text{else } \varphi_0^{n+1} X \end{cases}$

Proposition[Semantics of pre-cooking]

$$\boxed{\underbrace{(\{X_1/b_1, \ldots, X_p/b_p\}(a))_{pc}}_{\text{Substitution}} = \underbrace{a_{pc}\{X_1/b_{1_{pc}}, \ldots, X_p/b_{p_{pc}}\}}_{\text{Grafting}}}$$

Proposition[Correspondence between solutions]

$$\boxed{\exists N_1,\ldots,N_p \quad \underbrace{\{X_1/N_1,\ldots,X_p/N_p\}}_{\text{substitution}}(a) =_{\beta\eta} \underbrace{\{X_1/N_1,\ldots,X_p/N_p\}}_{\text{substitution}}(b)}$$

$$\Longleftrightarrow$$

$$\boxed{\exists M_1,\ldots,M_p \quad a_{pc}\underbrace{\{X_1/M_1,\ldots,X_p/M_p\}}_{\text{grafting}} =_{\lambda s_e} b_{pc}\underbrace{\{X_1/M_1,\ldots,X_p/M_p\}}_{\text{grafting}}}$$

5. A simple example

Problem: $\boxed{\lambda.(X\ \ 2) =^?_{\beta\eta} \lambda.2, \quad 2 : A, \ \ X : A \rightarrow A}$

$$\begin{array}{lr}
\lambda.(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} \lambda.2 & \rightarrow Dec\text{-}\lambda \\
(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} 2 & \rightarrow Exp\text{-}\lambda \\
\exists Y (\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y & \rightarrow Replace \\
\exists Y (\varphi^2_0(\lambda.Y)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y & \rightarrow Normalize \\
\exists Y (\varphi^2_1 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y & \rightarrow Exp\text{-}app \\
(\exists Y (\varphi^2_1 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y) \wedge (Y =^?_{\lambda s_e} 1 \vee Y =^?_{\lambda s_e} 2) & \rightarrow Replace \\
((\varphi^2_1 1)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee ((\varphi^2_1 2)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2) & \rightarrow Normalize \\
(2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee (2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2) & \equiv \\
(X =^?_{\lambda s_e} \lambda.1) \vee (X =^?_{\lambda s_e} \lambda.2) &
\end{array}$$

Problem: $\boxed{\lambda.(X\ \ 2) =^?_{\beta\eta} \lambda.2, \quad 2 : A, \quad X : A \to A}$

Solutions: $\left\{ \begin{array}{l} \{X/\lambda.1\} \\ \{X/\lambda.2\} \end{array} \right.$

Note that we have:

$$\begin{array}{r}\{X/\lambda.1\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.1)^+\}(X)\ \ 2) = \\ \lambda.(\lambda.1^{+1}\ \ 2) = \lambda.(\lambda.1\ \ 2) =_\beta \lambda.2\end{array}$$

and

$$\begin{array}{r}\{X/\lambda.2\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.2)^+\}(X)\ \ 2) = \\ \lambda.(\lambda.2^{+1}\ \ 2) = \lambda.(\lambda.3\ \ 2) =_\beta \lambda.2\end{array}$$

6. Related work

Our development of the λs_e-HOU was based on the ones of Dowek, Hardin and Kirchner for the $\lambda\sigma$-calculus of explicit substitutions.

One of our motivations was, in the practical setting of HOU, to compare the advantages and disadvantages of the two styles of explicit substitutions. This provides objective facts about that interesting theoretical question.

We think that our method can be adapted for applications in/for systems as the λProlog, Maude and ELAN.

Additional facts about the *back* transformation and practical considerations for an eventual implementation are available in Ayala-Rincón & Kamareddine *"On Applying λs_e-Style of Unification for Simply-Typed Higher Order Unification in the Pure λ-Calculus"* at `http://www.cee.hw.ac.uk/ultra/pubs.html`.

7. Future work and Conclusions

$$\text{To be done}\begin{cases}\bullet\ \text{Prototype implementation.}\\ \bullet\ \text{Comparison with the } \textit{suspension} \text{ calculus.}\end{cases}$$

- $\lambda\sigma$-(HO)Unification and λs_e-(HO)Unification strategies don't differ.
- Pre-cooking (and back) translations in $\lambda\sigma$ and λs_e differ:
  - A simple selection of the scripts for the operators φ and σ in λs_e corresponds to the manipulation of substitution objects in the $\lambda\sigma$-HOU approach.
  - Use of all de Bruijn indices makes our approach simpler.

References

G. Dowek, T. Hardin, and C. Kirchner. *Higher-order Unification via Explicit Substitutions*, Information and Computation, 157(1/2):183-235, 2000.

P. Borovanský. *Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions*. In M. Bartošek, J. Staudek, and J. Wiedermann, editors, *Proceedings of the SOFSEM'95: Theory and Practice of Informatics*, LNCS, 1012:363-368, 1995.

G. Nadathur and D.S. Wilson. *A Notation for Lambda Terms A Generalization of Environments*, Theoretical Computer Science, 198:49-98, 1998.

G. Nadathur. *A Fine-Grained Notation for Lambda Terms and Its Use in Intensional Operations*, The Journal of Functional and Logic Programming, 1999(2):1-62, 1999.