Matching via Explicit Substitutions

F. L. C. de Moura

GTC/UnB

June 1, 2005

Outline

Motivation

- Matching is a mechanism extensively used in computation for implementing proof assistants and programming languages.

Motivation

- Matching is a mechanism extensively used in computation for implementing proof assistants and programming languages.
- This is useful when we consider low-level implementations in which matching algorithms are to be implemented in the level of the language itself.

Motivation

- Matching is a mechanism extensively used in computation for implementing proof assistants and programming languages.
- This is useful when we consider low-level implementations in which matching algorithms are to be implemented in the level of the language itself.
- Explicit substitutions provide an adequate framework, closer to implementations, for reason theoretically about operational aspects of evaluation in the λ-calculus.

Motivation

- Matching is a mechanism extensively used in computation for implementing proof assistants and programming languages.
- This is useful when we consider low-level implementations in which matching algorithms are to be implemented in the level of the language itself.
- Explicit substitutions provide an adequate framework, closer to implementations, for reason theoretically about operational aspects of evaluation in the λ-calculus.
- In this work we present algorithms that decide second and third-order matching problems in the simply typed $\lambda\sigma$-calculus.

Notation

- *Matching equation*:

$$a \ll^? b$$

  where a and b are two λ-terms of the same type under the same context and b is ground.
- A substitution σ is a solution of the matching equation $a \ll^? b$ iff $a\sigma =_{\beta\eta} b$.
- A *second-order (third-order, resp.) matching problem* is a finite set of matching equations in which all meta-variables are at most second-order (third-order, resp.).

A Simple Example

- $append\ (X\ 1)\ (2 \cdot nil) \ll^{?} 1 \cdot 1 \cdot 2 \cdot nil$
- Solutions:

A Simple Example

- $\mathit{append}\ (X\ 1)\ (2 \cdot \mathit{nil}) \ll^{?} 1 \cdot 1 \cdot 2 \cdot \mathit{nil}$
- Solutions:
  - $X/\lambda y.(1 \cdot 1 \cdot \mathit{nil})$

A Simple Example

- $append\ (X\ 1)\ (2 \cdot nil) \ll^{?} 1 \cdot 1 \cdot 2 \cdot nil$
- Solutions:
  - $X/\lambda y.(1 \cdot 1 \cdot nil)$
  - $X/\lambda y.(1 \cdot y \cdot nil)$

A Simple Example

- $append\ (X\ 1)\ (2 \cdot nil) \ll^{?} 1 \cdot 1 \cdot 2 \cdot nil$
- Solutions:
  - $X/\lambda y.(1 \cdot 1 \cdot nil)$
  - $X/\lambda y.(1 \cdot y \cdot nil)$
  - $X/\lambda y.(y \cdot 1 \cdot nil)$

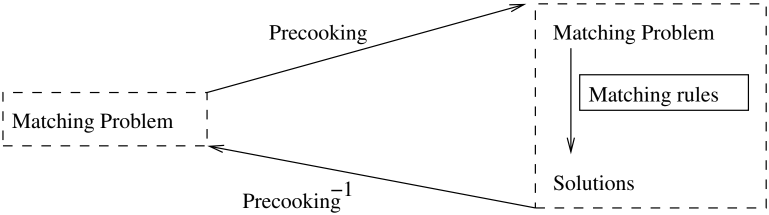
A Simple Example

- $append\ (X\ 1)\ (2 \cdot nil) \ll^? 1 \cdot 1 \cdot 2 \cdot nil$
- Solutions:
  - $X/\lambda y.(1 \cdot 1 \cdot nil)$
  - $X/\lambda y.(1 \cdot y \cdot nil)$
  - $X/\lambda y.(y \cdot 1 \cdot nil)$
  - $X/\lambda y.(y \cdot y \cdot nil)$
  - Note that there is no more general solution!

The $\lambda\sigma$-calculus

- Developed by M. Abadi, L. Cardelli, P.-L. Curien and J.J. Lévy in 1991[ACCL91].
- It uses two sorts:
  `terms:` $t ::= \underline{1} \mid X \mid (t\ t) \mid \lambda_A.t \mid t[s]$, where $X \in \mathcal{X}$.
  `substitutions:` $s ::= id \mid \uparrow \mid t \cdot s \mid s \circ s$
- Properties of the typed $\lambda\sigma$-calculus:
  1. Confluent.
  2. Weakly Terminating.

The precooking translation



Language of the Lambda calculus

Substitution

Language of the ES calculus

Grafting

The precooking translation

Definition (Precooking [DHK00])

Let $a \in \Lambda_{dB}(\mathcal{X})$ such that $\Gamma \vdash a : A$. To every meta-variable X of type B in the term a, we associate the type B and the context Γ in the $\lambda\sigma$-calculus. The *precooking* of a from $\Lambda_{dB}(\mathcal{X})$ to the set $\Lambda_{\lambda\sigma}(\mathcal{X})$ of $\lambda\sigma$-terms is given by $a_F = F(a, 0)$, where $F(a, n)$ is defined by:

1. $F((\lambda_B.a), n) = \lambda_B.F(a, n+1)$.
2. $F(\underline{\mathrm{k}}, n) = \underline{1}[\uparrow^{k-1}]$.
3. $F((a\ b), n) = (F(a, n)\ F(b, n))$.
4. $F(X, n) = X[\uparrow^n]$.

Remarks on decidability

- Second-Order Matching (SOM) is decidable for the simply typed λ-calculus [?].

Remarks on decidability

- Second-Order Matching (SOM) is decidable for the simply typed λ-calculus [**?**].
- The method of Dowek, Hardin and Kirchner does not decide arbitrary second-order $\lambda\sigma$-matching problems:

Remarks on decidability

- Second-Order Matching (SOM) is decidable for the simply typed λ-calculus [**?**].
- The method of Dowek, Hardin and Kirchner does not decide arbitrary second-order $\lambda\sigma$-matching problems:
- The counter-example: Consider $m \leq n$ and A an atomic type.

$$X_A^{A \to A \cdot \Delta}[(\lambda_A . \underline{1}_A^{A \cdot \Gamma}) \cdot \uparrow^{n}{}^{\Gamma}_{\Delta}] =^?_{\lambda\sigma} (\underline{\mathtt{m}}^{\Gamma}{}_{B_1 \to \ldots \to B_q \to A}\ b_1{}^{\Gamma}{}_{B_1} \ldots b_q{}^{\Gamma}{}_{B_q})$$

Exp-App

$$X[(\lambda.\underline{1})\cdot\uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \rightarrow^{\textbf{Exp}-\textbf{App}}$$

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

if X has an atomic type and is not solved
where $P' = P \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q)$, $H_1, \ldots, H_k$ are variables of appropriate types, not occurring in P, with 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 $m > n$ then $\{m - n + p\}$ else $\emptyset$.

Exp-App

$$X[(\lambda.\underline{1})\cdot\uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \rightarrow^{\textbf{Exp-App}}$$
$$X[(\lambda.\underline{1})\cdot\uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1)$$

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

if X has an atomic type and is not solved

where $P' = P \wedge X[a_1\cdot\ldots\cdot a_p\cdot\uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)$, $H_1,\ldots,H_k$ are variables of appropriate types, not occurring in P, with contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{1,\ldots,p\}$ such that $(\underline{r}\ H_1 \ldots H_k)$ has the right type, $R_i =$ if $m > n$ then $\{m-n+p\}$ else $\emptyset$.

Replace

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q) \rightarrow^{\mathbf{Exp-App}}$$
$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1)$$

Replace

$$\frac{P \wedge X =^?_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda\sigma} t} \text{ if } X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t) \text{ and,}$$

if t is a meta-variable then $t \in \mathcal{TV}ar(P)$.

Replace

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \rightarrow^{\textbf{Exp}-\textbf{App}}$$
$$X[(\lambda.\underline{1})\cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \rightarrow^{\textbf{Replace}}$$

Replace

$$\frac{P \wedge X =^?_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda\sigma} t} \text{ if } X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t) \text{ and,}$$

if t is a meta-variable then $t \in \mathcal{TV}ar(P)$.

Replace

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \dots b_q) \rightarrow^{\textbf{Exp-App}}$$

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \dots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \rightarrow^{\textbf{Replace}}$$

$$(\underline{1}\ H_1)[(\lambda.\underline{1}). \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \dots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1)$$

Replace

$$\frac{P \wedge X =^?_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda\sigma} t} \text{ if } X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t) \text{ and,}$$

if t is a meta-variable then $t \in \mathcal{TV}ar(P)$.

Normalise

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \longrightarrow^{\textbf{Exp}-\textbf{App}}$$
$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \longrightarrow^{\textbf{Replace}}$$
$$(\underline{1}\ H_1)[(\lambda.\underline{1}). \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1)$$

Normalise

$\dfrac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e'_1 =^?_{\lambda\sigma} e'_2}$ if e_1 or e_2 is not in η-long normal form, where e'_1 (resp. e'_2) is the η-long normal 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.

Normalise

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \rightarrow^{\textbf{Exp}-\textbf{App}}$$
$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \rightarrow^{\textbf{Replace}}$$
$$(\underline{1}\ H_1)[(\lambda.\underline{1}). \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \rightarrow^{\textbf{Normalise}}$$

Normalise

$\dfrac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e'_1 =^?_{\lambda\sigma} e'_2}$ if e_1 or e_2 is not in η-long normal form, where e'_1 (resp. e'_2) is the η-long normal 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.

Normalise

$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \rightarrow^{\mathbf{Exp-App}}$$
$$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \rightarrow^{\mathbf{Replace}}$$
$$(\underline{1}\ H_1)[(\lambda.\underline{1}). \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \rightarrow^{\mathbf{Normalise}}$$
$$H_1[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1)$$

Normalise

$\dfrac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e'_1 =^?_{\lambda\sigma} e'_2}$ if e_1 or e_2 is not in η-long normal form, where e'_1 (resp. e'_2) is the η-long normal 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.

Normalise

$X[(\lambda.\underline{1}) \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \to^{\textbf{Exp}-\textbf{App}}$

$X[(\lambda.\underline{1})\cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \to^{\textbf{Replace}}$

$(\underline{1}\ H_1)[(\lambda.\underline{1}).\ \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1) \to^{\textbf{Normalise}}$

$H_1[(\lambda.\underline{1})\cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q) \wedge X =^?_{\lambda\sigma} (\underline{1}\ H_1)$

Normalise

$\dfrac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e_1' =^?_{\lambda\sigma} e_2'}$ if e_1 or e_2 is not in η-long normal form, where e_1' (resp. e_2') is the η-long normal 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.

Characterisation of Matching Problems

Theorem

Let M be a second-order matching problem which is in the image of the precooking translation. Then every flexible term occurring in M' which is in the matching path of M, and of the form $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$, with $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$ in σ-normal form, is such that $a_1, \ldots, a_p$ are of atomic type.

Graphically:

$$X[\ \underbrace{a_1 \cdot \ldots \cdot a_p \cdot}_{\substack{\text{atomic}\\ \text{type}}}\ \underbrace{\underline{\mathtt{n}+1} \cdot \underline{\mathtt{n}+2} \cdot \ldots}_{\substack{\text{at most}\\ 2^{nd}\text{-order type}}}\ ] \equiv X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$$

Matching Rules

$$\textbf{Dec}_m\text{-}\lambda \ \frac{\langle \sigma, P \cup \{\lambda_A.a \ll^?_{\lambda\sigma} \lambda_A.b\}\rangle}{\langle \sigma, P \cup \{a \ll^?_{\lambda\sigma} b\}\rangle}$$

$$\textbf{Dec}_m\textbf{-App} \ \frac{\langle \sigma, P \cup \{(\underline{\mathtt{n}}\ a_1 \dots a_p) \ll^?_{\lambda\sigma} (\underline{\mathtt{n}}\ b_1 \dots b_p)\}\rangle}{\langle \sigma, P \cup \{a_1 \ll^?_{\lambda\sigma} b_1, \dots, a_p \ll^?_{\lambda\sigma} b_p\}\rangle}$$

$$\textbf{Dec}_m\textbf{-Fail} \ \frac{\langle \sigma, P \cup \{(\underline{\mathtt{n}}\ a_1 \dots a_p) \ll^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \dots b_q)\}\rangle}{Fail},$$

if $\mathtt{m} \neq \mathtt{n}$.

Matching Rules

Imit

$$\frac{\langle \sigma, P \cup \{X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q)\}\rangle}{\langle \sigma', P\sigma' \cup \{(\underline{\mathtt{m-n+p}}\ H_1 \ldots H_q)[a_1\sigma' \cdot \ldots \cdot a_p\sigma' \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{\mathtt{m}} b_1 \ldots b_q)\}\rangle}$$

if X has atomic type and $m > n$, where $\sigma' = \sigma\{X \mapsto (\underline{\mathtt{m-n+p}}\ H_1 \ldots H_q)\}$, $H_1, \ldots, H_q$ are meta-variables with appropriate type and with contexts $\Gamma_{H_i} = \Gamma_X (\forall 1 \leq i \leq q)$, and $\underline{\mathtt{m-n+p}}$ is at most third order.

Proj

$$\frac{\langle \sigma, P \cup \{X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q)\}\rangle}{\langle \sigma\{X \mapsto \underline{\mathtt{j}}\}, \{P\{X \mapsto \underline{\mathtt{j}}\} \cup \{a_j\{X \mapsto \underline{\mathtt{j}}\} \ll^?_{\lambda\sigma} (\underline{\mathtt{m}}\ b_1 \ldots b_q)\}\rangle} \text{ if}$$

X has atomic type, and the j-th element $(1 \leq j \leq p)$ of the explicit substitution $[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ has the same type of X.

Termination, Correctness and Completeness

Theorem

Applications of the previous rules to second-order matching problems, whose terms satisfy the previous theorem, always terminate.

Theorem

Solved forms of the algorithm derived from the presented second-order matching rules are in the image of the precooking translation.

Theorem

The presented second-order matching rules are correct and complete, in the sense that the set of matchers remains unchanged by applications of the matching rules.

Third-order Matching via Explicit Substitutions

- Third-order matching is decidable in the simply typed λ-calculus [Dow94].

Third-order Matching via Explicit Substitutions

- Third-order matching is decidable in the simply typed λ-calculus [Dow94].
- We proved that the Dowek's decision procedure can be adapted to the simply typed $\lambda\sigma$-calculus.

Third-order Matching via Explicit Substitutions

- Third-order matching is decidable in the simply typed λ-calculus [Dow94].
- We proved that the Dowek's decision procedure can be adapted to the simply typed $\lambda\sigma$-calculus.
- This is useful when we consider low-level implementations in which matching algorithms are to be implemented in the level of the language itself.

Third-order Matching via Explicit Substitutions

- Third-order matching is decidable in the simply typed λ-calculus [Dow94].
- We proved that the Dowek's decision procedure can be adapted to the simply typed $\lambda\sigma$-calculus.
- This is useful when we consider low-level implementations in which matching algorithms are to be implemented in the level of the language itself.
- The decision procedure is achieved firstly by reducing matching problems to interpolation problems in the language of the $\lambda\sigma$-calculus.

Third-order Matching via Explicit Substitutions

- Third-order matching is decidable in the simply typed λ-calculus [Dow94].
- We proved that the Dowek's decision procedure can be adapted to the simply typed $\lambda\sigma$-calculus.
- This is useful when we consider low-level implementations in which matching algorithms are to be implemented in the level of the language itself.
- The decision procedure is achieved firstly by reducing matching problems to interpolation problems in the language of the $\lambda\sigma$-calculus.
- After that we show that if an interpolation problem has a solution then it also has a solution which depends only the initial matching problem.

From Matching Problems to Interpolation Problems

Definition

Let $a \ll^?_{\lambda\sigma} b$ be a matching equation and σ a ground solution to this equation, i.e., the $\lambda\sigma$-normal form of $a\sigma$ is equal to b. We define the interpolation problem $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$ inductively over the number of occurrences of a as follows:

- If $a = \lambda_A.c$ then b is also an abstraction of the form $\lambda_A.d$ and then σ is also a solution of $c \ll^?_{\lambda\sigma} d$ and we let $\Phi(a \ll^?_{\lambda\sigma} b, \sigma) = \Phi(c \ll^?_{\lambda\sigma} d, \sigma)$.
- If $a = (\underline{\mathrm{k}}\ c_1 \ldots c_m)$ then b is also of the form $(\underline{\mathrm{k}}\ d_1 \ldots d_m)$ because $a \ll^?_{\lambda\sigma} b$ is solvable and we let $\Phi(a \ll^?_{\lambda\sigma} b, \sigma) = \bigcup_i \Phi(c_i \ll^?_{\lambda\sigma} d_i, \sigma)$.

From Matching Problems to Interpolation Problems

Definition (cont.)

- If $a = (X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c_1 \ldots c_m)$ then we let
  $\Phi(a \ll^?_{\lambda\sigma} b, \sigma) =$
  $\{(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c_1\sigma \ldots c_m\sigma) \ll^?_{\lambda\sigma} b\} \bigcup_i H_i$, where

$$H_i = \begin{cases} \Phi(c_i \ll^?_{\lambda\sigma} c_i\sigma, \sigma), \text{ if the dummy symbol } \diamond \text{ occurs} \\ \text{ in the normal form of} \\ (X\sigma[a_1\sigma \cdot \ldots \cdot a_p\sigma \cdot \uparrow^n]\ c_1\sigma \ldots c_{i-1}\sigma \diamond c_{i+1}\sigma \ldots c_m\sigma); \\ \emptyset, \text{ otherwise.} \end{cases}$$

From Matching Problems to Interpolation Problems

Theorem

Let $a \ll^?_{\lambda\sigma} b$ be a matching equation and σ a ground solution to this equation. Then the substitution σ is a solution to $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$ and, conversely, if σ' is a solution to $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$ then σ' is also a solution to the matching equation $a \ll^?_{\lambda\sigma} b$.

Definition

Let Ψ be a third-order matching problem and σ be a solution to Ψ. We let $\Phi(\Psi, \sigma)$ be the following third-order interpolation problem:

$$\Phi(\Psi, \sigma) = \bigcup_{a \ll^?_{\lambda\sigma} b \in \Psi} \Phi(a \ll^?_{\lambda\sigma} b, \sigma).$$

$\lambda\sigma$-Böhm Trees

Definition ($\lambda\sigma$-Böhm Trees)

A $\lambda\sigma$-Böhm tree is a tree whose nodes are labeled with pairs $\langle l, v_A^\Delta \rangle$ such that l is a positive integer and v_A^Δ is a $\lambda\sigma$-term of type A under context Δ.

$\lambda\sigma$-Böhm tree of a $\lambda\sigma$-term in normal form

Definition ($\lambda\sigma$-Böhm tree of a $\lambda\sigma$-term in normal form)

Let $a^\Gamma_A = \lambda_{A_1} \cdots \lambda_{A_k}.(h^\Sigma_{B_1\to\ldots\to B_m\to B}\ {b_1}^\Sigma_{B_1} \cdots {b_m}^\Sigma_{B_m})$ be a term in $\lambda\sigma$-nf, where $\Sigma = A_1 \cdot \ldots \cdot A_k \cdot \Gamma$. The Böhm tree of a^Γ_A is recursively defined as the tree whose root is labeled with the pair $\langle k, h^\Sigma_{B_1\to\ldots\to B_m\to B}\rangle$ and whose sons are the $\lambda\sigma$-Böhm trees of:

1. ${b_1}^\Sigma_{B_1},\ldots,{b_m}^\Sigma_{B_m}$, if $h^\Sigma_{B_1\to\ldots\to B_m\to B}$ is a de Bruijn index;
2. ${a_1}^\Sigma_{A_1},\ldots,{a_p}^\Sigma_{A_p}, {b_1}^\Sigma_{B_1},\ldots,{b_m}^\Sigma_{B_m}$, if $h^\Sigma_{B_1\to\ldots\to B_m\to B}$ is a meta-variable of the form $X^\Gamma_A[{a_1}^\Sigma_{A_1} \cdot \ldots \cdot {a_p}^\Sigma_{A_p} \cdot \uparrow^{n\Sigma}_{\Delta}]$, where ${a_1}^\Sigma_{A_1} \cdot \ldots \cdot {a_p}^\Sigma_{A_p} \cdot \uparrow^{n\Sigma}_{\Delta}$ is a substitution in $\lambda\sigma$-nf.

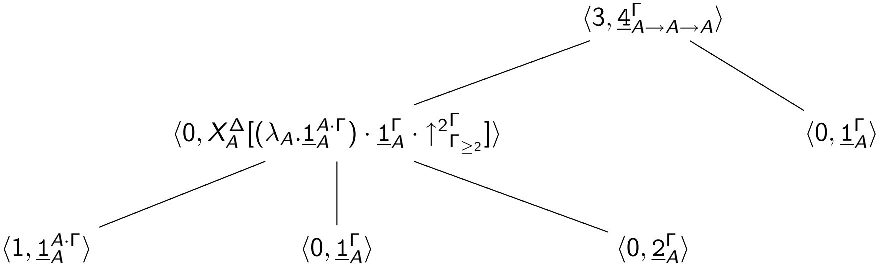
Example

The $\lambda\sigma$-Böhm tree of the term $\lambda_A\lambda_A\lambda_A.(\underline{4}^{\Gamma}_{A\rightarrow A\rightarrow A}\; X^{\Gamma}_{A}\; \underline{1}^{\Gamma}_{A})$, where $\Gamma = A \cdot A \cdot A \cdot A \rightarrow A \rightarrow A \cdot nil$ is given by:

$$
\begin{array}{ccc}
 & \langle 3, \underline{4}^{\Gamma}_{A\rightarrow A\rightarrow A} \rangle & \\
\swarrow & & \searrow \\
\langle 0, X^{\Gamma}_{A} \rangle & & \langle 0, \underline{1}^{\Gamma}_{A} \rangle
\end{array}
$$

Another Example

The $\lambda\sigma$-Böhm tree of the term $\lambda_A\lambda_A\lambda_A.(\underline{4}^{\Gamma}_{A\to A\to A}\ (X^{\Delta}_{A\to A}[(\lambda_A.\underline{1}^{A\cdot\Gamma}_{A})\cdot\underline{1}^{\Gamma}_{A}\cdot\uparrow^{2}{}^{\Gamma}_{\Gamma_{\geq 2}}]\ \underline{2}^{\Gamma}_{A})\ \underline{1}^{\Gamma}_{A})$, where $\Gamma = A\cdot A\cdot A\cdot A\to A\to A\cdot nil$ and $\Delta = A\to A\cdot A\cdot\Gamma_{\geq 2}$ is given by:



Accessible Occurrence

Definition

Consider an equation of the form $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]c_1 \ldots c_q) = b$ and the term $t = \lambda_{C_1} \ldots \lambda_{C_q}.u$ with the same type of X. The set of occurrences in the $\lambda\sigma$-Böhm tree of t *accessible* w.r.t. the equation $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c_1 \ldots c_q) = b$ is inductively defined as:

- the root of the $\lambda\sigma$-Böhm tree of t is accessible.
- if α is an accessible occurrence labeled with a de Bruijn index $\underline{j}$ with $1 \leq j \leq p+q$ and d_j is relevant in its r-th argument then the occurrence $\alpha\langle r\rangle$ is accessible, where:
  $$d_j = \begin{cases} a_j & \text{if } q < j \leq p+q, \\ c_{q-i+1} & \text{if } 1 \leq j \leq q. \end{cases}$$

Accessible Occurrence

Definition (cont.)

- if α is an accessible occurrence labeled with a de Bruijn index greater than $p + q$ or with a meta-variable then all the sons of α are accessible.
- if α is an accessible occurrence labeled with a meta-variable then each son of α is accessible.

Accessible term

Definition (Occurrence accessible w.r.t. an interpolation problem [Dow94])

An occurrence is accessible with respect to an interpolation problem if it is accessible with respect to one of the equations of this problem.

Definition ($\lambda\sigma$-term accessible w.r.t. to an interpolation problem)

A $\lambda\sigma$-term is accessible with respect to an interpolation problem if all occurrences of its $\lambda\sigma$-Böhm tree which are not leaves are accessible with respect to this problem.

Accessible Solution

Definition (Accessible solution built from a solution)

Let Φ be an interpolation problem and let σ be a ground solution to this problem. For each meta-variable X occurring in the equations of Φ consider the $\lambda\sigma$-term t such that $\{X \mapsto t\} \subseteq \sigma$. In the $\lambda\sigma$-Böhm tree of t, we prune all occurrences non accessible (that are not leaves) with respect to the equations of Φ in which X has an occurrence and put $\lambda\sigma$-Böhm trees of ground terms of depth 0 of the expected type as leaves. Call t' the term whose $\lambda\sigma$-Böhm is obtained this way and $\widehat{\sigma}$ the resulting substitution.

Accessible Solution

Theorem

Let Φ be an interpolation problem generated from a precooked matching problem and let σ be a ground solution to Φ. Then the accessible solution $\widehat{\sigma}$, built from σ, is a solution to Φ.

Compact $\lambda\sigma$-term

Definition

$\lambda\sigma$-term $t = \lambda_{C_1} \ldots \lambda_{C_q}.u$ (u atomic) is *compact* w.r.t. an interpolation problem Φ if no de Bruijn index $\underline{\mathtt{j}}$ with $1 \leq j \leq q$ appears free in a path of the $\lambda\sigma$-Böhm tree of u more than $h + 1$ times, where h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ.

Compact Solution

Definition

Let Φ be an interpolation problem, $\widehat{\sigma}$ be an accessible solution to this problem and h be the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ. The grafting $\widehat{\sigma}$ is a *compact accessible solution built from an accessible solution* to Φ if, for all meta-variable X occurring in Φ, the term $t = X\widehat{\sigma} = \lambda_{C_1} \ldots \lambda_{C_q}.u$ (u atomic) is such that there is no path in the $\lambda\sigma$-Böhm tree of u containing more than $h+1$ occurrences labeled with the de Bruijn index $\underline{\mathtt{j}}$ $(1 \leq j \leq q)$. If there exists a path in the $\lambda\sigma$-Böhm tree of u that has more than $h+1$ free occurrences of the de Bruijn index $\underline{\mathtt{j}}$ $(1 \leq j \leq q)$ then the compact accessible solution σ' is built as follows: we replace all these occurrences of $\underline{\mathtt{j}}$ by $\lambda_{B_1} \ldots \lambda_{B_p}.\underline{\mathtt{r}}$.

Compact Solution

Theorem

Let Φ be an interpolation problem, σ a solution to Φ, $\widehat{\sigma}$ be the accessible solution built from σ and σ' be the compact accessible solution built from $\widehat{\sigma}$. Then σ' is also a solution to Φ.

Compact Solution

Theorem

Let Φ be an interpolation problem, σ be a solution to Φ, $\widehat{\sigma}$ be the accessible solution built from σ and σ' be the compact accessible solution built from $\widehat{\sigma}$. If h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ, then for every meta-variable X of arity q, the depth of the $\lambda\sigma$-Böhm tree of $X\sigma' = \lambda_{C_1} \ldots \lambda_{C_q}.u'$ is less than or equal to $(q+1)(h+1)-1$.

Compact Solution

Corollary

Let Φ be a third-order interpolation problem, σ be a solution to Φ, $\widehat{\sigma}$ be the accessible solution built from σ and σ' be the compact accessible solution built from $\widehat{\sigma}$. If h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ, then for every meta-variable X of arity q, the depth of the $\lambda\sigma$-Böhm tree of $X\sigma' = \lambda_{C_1} \ldots \lambda_{C_q}.u'$ is less than or equal to $(q+1)(h+1)-1$.

The Decision Procedure

Theorem

The class of third-order $\lambda\sigma$-matching problems that come from the simply typed λ-calculus is decidable.

Proof.

Let Ψ be a third-order matching problem in the $\lambda\sigma$-calculus. Enumerate all ground substitutions for the meta-variables occurring in the equations of the form $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c_1 \ldots c_q) \ll^?_{\lambda\sigma} b$ of Ψ, such that the terms to be substituted for X have depth less than or equal to $(q+1)(h+1)-1$, where h is the depth of the $\lambda\sigma$-Böhm tree of b. If none of these substitutions is a solution Φ then Φ is not solvable. Otherwise, it is solvable. $\square$

Conclusion

- We presented a second-order matching algorithm which uses an adequate notation that does not mix graftings with matching equations.
- This algorithm decides all second-order matching problems that are originated in the simply typed λ-calculus.
- We adapted the Dowek's decision procedure for third-order matching in the simply-typed $\lambda\sigma$-calculus.
- To do so, we defined the notion of $\lambda\sigma$-Böhm tree, which extends the usual notion of Böhm tree for the $\lambda\sigma$-calculus.
- This work is important for considering low-level implementations of languages based on the simply typed λ-calculus in which matching algorithms are to be implemented in the level of the language itself.

Future Work

- Extension of this work to other styles of explicit substitutions.
- Implementation of the algorithms to evaluate performance.

M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy.
Explicit Substitutions.
J. of Func. Programming, 1(4):375–416, 1991.

G. Dowek, T. Hardin, and C. Kirchner.
Higher-order unification via explicit substitutions.
Information and Computation, 157:183–235, 2000.

G. Dowek.
Third-order matching is decidable.
APAL 69:135–155, 1994.

G. Huet.
A Unification Algorithm for Typed λ-Calculus.
TCS, 1:27–57, 1975.

G. Huet.
Résolution d'équations dans les langages d'ordre 1,2,...,ω.
PhD thesis, University Paris-7, 1976.

F. L. C. de Moura, F. Kamareddine, and M. Ayala-Rincón.
Second-Order Matching via Explicit Substitutions
Springer-Verlag LNAI 3452, 2005.

F. L. C. de Moura, F. Kamareddine, and M. Ayala-Rincón.
Third-Order Matching via Explicit Substitutions
Submitted, 2005.

Solved Forms

Definition

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 η-long normal form. Such an equation is said to be *solved* in P and the variable X is also said to be solved.
- **Flexible-flexible**: $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} Y[b_1 \cdot \ldots \cdot b_q \cdot \uparrow^m]$, where $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ and $Y[b_1 \cdot \ldots \cdot b_q \cdot \uparrow^m]$ are $\lambda\sigma$-terms in η-long normal form and the equation is not solved.

return