Matching via Explicit Substitutions

F. L. C. de Moura

GTC/UnB

June 1, 2005
Outline

Introduction
  Motivation
  Definition and a Small Example
Explicit Substitutions
  The $\lambda\sigma$-calculus
Second-Order Matching via Explicit Substitutions
  The precooking translation
  Remarks on decidability
  The counter-example
  Characterisation of Matching Problems
  Matching Rules
  Termination, Correctness and Completeness
Third-order Matching via Explicit Substitutions
  Interpolation Problems
  $\lambda\sigma$-Böhm Trees
  Examples
  Accessible Solution
  Compact Solution
  The Decision Procedure
Conclusion and Future Work
  Conclusion
Matching is a mechanism extensively used in computation for implementing proof assistants and programming languages.
Introduction

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 $\lambda$-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 $\lambda$-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:**
  \[ a \triangleleft b \]
  where \( a \) and \( b \) are two \( \lambda \)-terms of the same type under the same context and \( b \) is ground.

- A substitution \( \sigma \) is a solution of the matching equation \( a \triangleleft b \) 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

- $\text{append } (X \ 1) \ (2 \cdot \text{nil}) \iff 1 \cdot 1 \cdot 2 \cdot \text{nil}$

- Solutions:
A Simple Example

- $append \ (X \ 1) \ (2 \cdot \text{nil}) \ <\ ? \ 1 \cdot 1 \cdot 2 \cdot \text{nil}$
- Solutions:
  - $X/\lambda y.(1 \cdot 1 \cdot \text{nil})$
A Simple Example

- \textit{append} \ ((X \ 1) \ (2 \cdot \text{nil})) \triangleleft\triangleleft ? \ 1 \cdot 1 \cdot 2 \cdot \text{nil}

- Solutions:
  - \( X/\lambda y. (1 \cdot 1 \cdot \text{nil}) \)
  - \( X/\lambda y. (1 \cdot y \cdot \text{nil}) \)

Note that there is no more general solution!
A Simple Example

▶ append \((X \ 1) \ (2 \cdot \text{nil})\) \(\triangleleft\) ? \(1 \cdot 1 \cdot 2 \cdot \text{nil}\)

▶ Solutions:

▶ \(X/\lambda y.(1 \cdot 1 \cdot \text{nil})\)
▶ \(X/\lambda y.(1 \cdot y \cdot \text{nil})\)
▶ \(X/\lambda y.(y \cdot 1 \cdot \text{nil})\)

Note that there is no more general solution!
A Simple Example

- `append (X 1) (2 · nil) ≲? 1 · 1 · 2 · nil`

- Solutions:
  - `X/λy.(1 · 1 · nil)`
  - `X/λy.(1 · y · nil)`
  - `X/λy.(y · 1 · nil)`
  - `X/λy.(y · y · nil)`
  - Note that there is no more general solution!

It uses two sorts:
- **terms:** \( t ::= 1 | X | (t t) | \lambda A.t | t[s], \) where \( X \in X. \)
- **substitutions:** \( s ::= id | \uparrow | t \cdot s | s \circ s \)

Properties of the typed \( \lambda \sigma \)-calculus:
1. Confluent.
2. Weakly Terminating.
The precooking translation

Matching Problem

Precooking

Precooking$^{-1}$

Matching Problem

Matching rules

Solutions

Language of the Lambda calculus

Substitution

Language of the ES calculus

Grafting

F.L.C. de Moura

Matching via Explicit Substitutions
The precooking translation

Definition (Precooking [DHK00])

Let \( a \in \Lambda_{dB}(X) \) 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 \( \Gamma \) in the \( \lambda\sigma \)-calculus. The precooking of \( a \) from \( \Lambda_{dB}(X) \) to the set \( \Lambda_{\lambda\sigma}(X) \) 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. \( F(k, n) = 1[\uparrow^{k-1}] \).
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 \( \lambda \)-calculus [?].
Remarks on decidability

- Second-Order Matching (SOM) is decidable for the simply typed $\lambda$-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 \( \lambda \)-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.

\[
\chi^A_{A \rightarrow A \cdot \Delta} \left[ (\lambda A.1^A_{A \cdot \Gamma}) \cdot \uparrow^n_{\Delta} \right] = ?_{\lambda \sigma} \left( \frac{m}{B_1 \rightarrow \ldots \rightarrow B_q \rightarrow A} b_1 \frac{B_1}{b_2 \ldots b_q \frac{B_q}{}} \right)
\]
Exp-App

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

\[
\begin{align*}
\text{Exp-App} & \quad P \land X[a_1 \ldots a_p \cdot \uparrow^n] =_\lambda^\sigma (m \ b_1 \ldots b_q) \\
& \quad P' \land \bigvee_{r \in R_p \cup R_i} X =_\lambda^\sigma (r \ H_1 \ldots H_k) \\
& \quad \text{if } X \text{ has an atomic type and is not solved} \\
\end{align*}
\]

if \( X \) has an atomic type and is not solved

where \( P' = P \land X[a_1 \ldots a_p \cdot \uparrow^n] =_\lambda^\sigma (m \ b_1 \ldots b_q) \),

\( H_1, \ldots, H_k \) 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 \( (r \ H_1 \ldots H_k) \) has the right type, \( R_i = \) if \( m > n \) then \( \{m - n + p\} \) else \( \emptyset \).
Exp-App

\[ X[(\lambda.1) \cdot \uparrow^n] = \lambda^\sigma (m \ b_1 \ldots b_q) \rightarrow_{\text{Exp-App}} \]

\[ X[(\lambda.1) \cdot \uparrow^n] = \lambda^\sigma (m \ b_1 \ldots b_q) \land X = \lambda^\sigma (1 \ H_1) \]

\[
\begin{align*}
\text{Exp-App} & \quad P \land X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] = \lambda^\sigma (m \ b_1 \ldots b_q) \\
& \quad P' \land \bigvee_{r \in R_p \cup R_i} X = \lambda^\sigma (r \ H_1 \ldots H_k) \\
\end{align*}
\]

if \( X \) has an atomic type and is not solved

where \( P' = P \land X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] = \lambda^\sigma (m \ b_1 \ldots b_q) \),

\( H_1, \ldots, H_k \) 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 \( (r \ H_1 \ldots H_k) \) has the right type, \( R_i = \) if \( m > n \) then \( \{m - n + p\} \) else \( \emptyset \).
Explicit Substitutions

Second-Order Matching via Explicit Substitutions

Third-order Matching via Explicit Substitutions

Conclusion and Future Work

The precooking translation

Remarks on decidability

The counter-example

Characterisation of Matching Problems

Matching Rules

Termination, Correctness and Completeness

Replace

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

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

Replace

\[
\frac{P \land X = ?_{\lambda\sigma} t}{{X \mapsto t}(P) \land X = ?_{\lambda\sigma} t}
\]

if \( X \in T\text{Var}(P) \), \( X \notin T\text{Var}(t) \) and, if \( t \) is a meta-variable then \( t \in T\text{Var}(P) \).
Replace

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

\[
X[(\lambda.1) \cdot \uparrow^n] = ?_{\lambda\sigma} (m \ b_1 \ldots b_q) \land X = ?_{\lambda\sigma} (1 \ H_1) \rightarrow^{\text{Replace}}
\]

Replace

\[
P \land X = ?_{\lambda\sigma} t
\]

\[
\frac{\{X \mapsto t\}(P) \land X = ?_{\lambda\sigma} t}{\text{if } X \in TVar(P), X \not\in TVar(t) \text{ and,}}
\]

\[
\text{if } t \text{ is a meta-variable then } t \in TVar(P).
\]
Replace

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

Replace

\[ P \land X =^?=_{\lambda \sigma} t \]
\[ \{X \mapsto t\}(P) \land X =^?=_{\lambda \sigma} t \] if \( X \in TVar(P) \), \( X \not\in TVar(t) \) and, if \( t \) is a meta-variable then \( t \in TVar(P) \).
Normalise

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

**Normalise**

\[ P \land e_1 = ?_{\lambda\sigma} e_2 \]

if \( e_1 \) or \( e_2 \) is not in \( \eta \)-long normal form, where \( e'_1 \) (resp. \( e'_2 \)) is the \( \eta \)-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

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

$$X[(\lambda.1) \cdot \uparrow^n] = ?_{\lambda \sigma} (m \ b_1 \ldots b_q) \land X = ?_{\lambda \sigma} (1 \ H_1) \rightarrow \text{Replace}$$

$$(1 \ H_1)[(\lambda.1) \cdot \uparrow^n] = ?_{\lambda \sigma} (m \ b_1 \ldots b_q) \land X = ?_{\lambda \sigma} (1 \ H_1) \rightarrow \text{Normalise}$$

**Normalise**

$$P \land e_1 = ?_{\lambda \sigma} e_2$$

if $e_1$ or $e_2$ is not in $\eta$-long normal form, where

$$P \land e'_1 = ?_{\lambda \sigma} e'_2$$

$e'_1$ (resp. $e'_2$) is the $\eta$-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

\[ X[(\lambda.1) \cdot \uparrow^n] =^?_{\lambda\sigma} (m\ b_1 \ldots b_q) \rightarrow_{\text{Exp-App}} \]

\[ X[(\lambda.1) \cdot \uparrow^n] =^?_{\lambda\sigma} (m\ b_1 \ldots b_q) \land X =^?_{\lambda\sigma} (1 \ H_1) \rightarrow_{\text{Replace}} \]

\[ (1 \ H_1)[(\lambda.1) \cdot \uparrow^n] =^?_{\lambda\sigma} (m\ b_1 \ldots b_q) \land X =^?_{\lambda\sigma} (1 \ H_1) \rightarrow_{\text{Normalise}} \]

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

**Normalise**

\[
\begin{align*}
P \land e_1 &=^?_{\lambda\sigma} e_2 \\
P \land e'_1 &=^?_{\lambda\sigma} e'_2
\end{align*}
\]

if \( e_1 \) or \( e_2 \) is not in \( \eta \)-long normal form, where

\( e'_1 \) (resp. \( e'_2 \)) is the \( \eta \)-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

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

**Normalise**

\[
P \wedge e_1 = ?_{\lambda \sigma} e_2 \quad \text{if } e_1 \text{ or } e_2 \text{ is not in } \eta\text{-long normal form, where}
\]
\[
P \wedge e'_1 = ?_{\lambda \sigma} e'_2
\]

if \(e_1\) or \(e_2\) is not in \(\eta\)-long normal form, where \(e'_1\) (resp. \(e'_2\)) is the \(\eta\)-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 $\sigma$-normal form, is such that $a_1, \ldots, a_p$ are of atomic type.

**Graphically:**

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

\[
\begin{align*}
& \text{Dec}_{m-\lambda} \quad \frac{\langle \sigma, P \cup \{ \lambda_A.a \ll_{\lambda_\sigma} \lambda_A.b \} \rangle}{\langle \sigma, P \cup \{ a \ll_{\lambda_\sigma} b \} \rangle} \\
\ & \text{Dec}_{m-\text{App}} \quad \frac{\langle \sigma, P \cup \{ (n\ a_1 \ldots a_p) \ll_{\lambda_\sigma} (n\ b_1 \ldots b_p) \} \rangle}{\langle \sigma, P \cup \{ a_1 \ll_{\lambda_\sigma} b_1, \ldots, a_p \ll_{\lambda_\sigma} b_p \} \rangle} \\
& \text{Dec}_{m-\text{Fail}} \quad \frac{\langle \sigma, P \cup \{ (n\ a_1 \ldots a_p) \ll_{\lambda_\sigma} (m\ b_1 \ldots b_q) \} \rangle}{\text{Fail}}, \quad \text{if } m \neq n.
\end{align*}
\]
Matching Rules

Imit

\[
\langle \sigma, P \cup \{X[a_1 \ldots \cdot a_p \cdot \uparrow^n] \ll\ll_{\lambda_\sigma} (m \ b_1 \ldots b_q)\} \rangle \quad \langle \sigma', P \sigma' \cup \{(m-n+p \ H_1\ldots H_q)[a_1 \sigma' \ldots \cdot a_p \sigma' \cdot \uparrow^n] \ll\ll_{\lambda_\sigma} (m b_1 \ldots b_q)\} \rangle
\]

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

Proj

\[
\langle \sigma, P \cup \{X[a_1 \ldots \cdot a_p \cdot \uparrow^n] \ll\ll_{\lambda_\sigma} (m b_1 \ldots b_q)\} \rangle \quad \langle \sigma\{X \mapsto j\}, \{P\{X \mapsto j\} \cup \{a_j\{X \mapsto j\} \ll\ll_{\lambda_\sigma} (m b_1 \ldots b_q)\}\} \rangle
\]

if \( X \) has atomic type, and the \( j \)-th element (\( 1 \leq j \leq p \)) of the explicit substitution \([a_1 \ldots \cdot a_p \cdot \uparrow^n]\) 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 \( \lambda \)-calculus [Dow94].
Third-order Matching via Explicit Substitutions

- Third-order matching is decidable in the simply typed \(\lambda\)-calculus [Dow94].
- We proved that the Dowek’s decision procedure can be adapted to the simply typed \(\lambda\sigma\)-calculus.
Third-order matching is decidable in the simply typed $\lambda$-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 is decidable in the simply typed $\lambda$-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 \( \lambda \)-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 on the initial matching problem.
Definition

Let \( a \ll_{\lambda\sigma}? b \) be a matching equation and \( \sigma \) 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 \( \sigma \) 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 \( a = (k \ c_1 \ldots c_m) \) then \( b \) is also of the form \( (k \ d_1 \ldots d_m) \) 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 $a = (X[a_1 \ldots a_p \uparrow^n] c_1 \ldots c_m)$ then we let
  $$\Phi(a \ll_{\lambda\sigma}^? b, \sigma) = \{(X[a_1 \ldots a_p \uparrow^n] c_1 \sigma \ldots c_m \sigma) \ll_{\lambda\sigma}^? b\} \bigcup_{i} H_i,$$

where

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

**Theorem**

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

**Definition**

Let $\Psi$ be a third-order matching problem and $\sigma$ be a solution to $\Psi$. 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 \( \langle l, \nu_A^A \rangle \) such that \( l \) is a positive integer and \( \nu_A^A \) is a \( \lambda\sigma \)-term of type \( A \) under context \( \Delta \).
\( \lambda \sigma \)-Böhm tree of a \( \lambda \sigma \)-term in normal form

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

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

1. \( b_1^\Sigma_{B_1}, \ldots, b_m^\Sigma_{B_m} \), if \( h^\Sigma_{B_1 \rightarrow \cdots \rightarrow B_m \rightarrow B} \) 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 \( h^\Sigma_{B_1 \rightarrow \cdots \rightarrow B_m \rightarrow B} \) is a meta-variable of the form \( X^\Gamma_A[a_1^\Sigma_{A_1} \cdots a_p^\Sigma_{A_p} \cdot \uparrow^n_{\Delta}] \), where \( a_1^\Sigma_{A_1} \cdots a_p^\Sigma_{A_p} \cdot \uparrow^n_{\Delta} \) is a substitution in \( \lambda \sigma \)-nf.
Example

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

$$\langle 3, 4_{A A A A} \rangle$$

$$\langle 0, X_A \rangle$$

$$\langle 0, 1_A \rangle$$
Another Example

The $\lambda\sigma$-Böhm tree of the term

\[
\lambda A \lambda A \lambda A \cdot (4\Gamma \rightarrow A \rightarrow A \cdot X^\Delta A \rightarrow A [\lambda A \cdot 1^A \cdot \Gamma] \cdot 1^A \cdot \Gamma \cdot \uparrow^{2^\Gamma \Gamma \geq 2} \cdot 2^\Delta A \cdot 1^A)\),
\]

where $\Gamma = A \cdot A \cdot A \cdot A \rightarrow A \rightarrow A \cdot \text{nil}$ and $\Delta = A \rightarrow A \cdot A \cdot \Gamma \geq 2$ is given by:
Accessible Occurrence

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

- the root of the \(λσ\)-Böhm tree of \(t\) is accessible.
- if \(α\) is an accessible occurrence labeled with a de Bruijn index \(j\) with \(1 ≤ j ≤ p + q\) and \(d_j\) is relevant in its \(r\)-th argument then the occurrence \(α\langle r \rangle\) is accessible, where:

\[
d_j = \begin{cases} 
  a_j & \text{if } q < j ≤ p + q, \\
  c_{q-i+1} & \text{if } 1 ≤ j ≤ q.
\end{cases}
\]
Accessible Occurrence

Definition (cont.)

- if $\alpha$ is an accessible occurrence labeled with a de Bruijn index greater than $p + q$ or with a meta-variable then all the sons of $\alpha$ are accessible.
- if $\alpha$ is an accessible occurrence labeled with a meta-variable then each son of $\alpha$ is accessible.
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 $\Phi$ be an interpolation problem and let $\sigma$ be a ground solution to this problem. For each meta-variable $X$ occurring in the equations of $\Phi$ 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 $\Phi$ 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 $\hat{\sigma}$ the resulting substitution.
Accessible Solution

Theorem
Let $\Phi$ be an interpolation problem generated from a precooked matching problem and let $\sigma$ be a ground solution to $\Phi$. Then the accessible solution $\hat{\sigma}$, built from $\sigma$, is a solution to $\Phi$. 
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 $\Phi$ if no de Bruijn index $j$ 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 $\Phi$. 
Compact Solution

Definition
Let $\Phi$ be an interpolation problem, $\hat{\sigma}$ 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 $\Phi$. The grafting $\hat{\sigma}$ is a compact accessible solution built from an accessible solution to $\Phi$ if, for all meta-variable $X$ occurring in $\Phi$, the term $t = X\hat{\sigma} = \lambda_{C_1} \ldots \lambda_{C_q}.u$ ($u$ 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 $j$ ($1 \leq j \leq q$). 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 $j$ ($1 \leq j \leq q$) then the compact accessible solution $\sigma'$ is built as follows: we replace all these occurrences of $j$ by $\lambda_{B_1} \ldots \lambda_{B_p}.r$. 
Compact Solution

**Theorem**

Let $\Phi$ be an interpolation problem, $\sigma$ a solution to $\Phi$, $\hat{\sigma}$ be the accessible solution built from $\sigma$ and $\sigma'$ be the compact accessible solution built from $\hat{\sigma}$. Then $\sigma'$ is also a solution to $\Phi$. 
Compact Solution

Theorem

Let $\Phi$ be an interpolation problem, $\sigma$ be a solution to $\Phi$, $\hat{\sigma}$ be the accessible solution built from $\sigma$ and $\sigma'$ be the compact accessible solution built from $\hat{\sigma}$. If $h$ is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of $\Phi$, 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 $(q + 1)(h + 1) - 1$. 
Corollary

Let $\Phi$ be a third-order interpolation problem, $\sigma$ be a solution to $\Phi$, $\hat{\sigma}$ be the accessible solution built from $\sigma$ and $\sigma'$ be the compact accessible solution built from $\hat{\sigma}$. If $h$ is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of $\Phi$, 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 $\lambda$-calculus is decidable.

Proof.

Let $\Psi$ 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 $\Psi$, 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 $\Phi$ then $\Phi$ is not solvable. Otherwise, it is solvable.
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 $\lambda$-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 $\lambda$-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.
Explicit Substitutions.

G. Dowek, T. Hardin, and C. Kirchner.
Higher-order unification via explicit substitutions.

G. Dowek.
Third-order matching is decidable.

G. Huet.
*A Unification Algorithm for Typed λ-Calculus.*

G. Huet.
*Résolution d’équations dans les langages d’ordre 1,2,…,ω.*

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 $\eta$-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**: $X[a_1 \cdot \ldots \cdot a_p \uparrow^n] = ?_{\lambda\sigma} Y[b_1 \cdot \ldots \cdot b_q \uparrow^m]$, where $X[a_1 \cdot \ldots \cdot a_p \uparrow^n]$ and $Y[b_1 \cdot \ldots \cdot b_q \uparrow^m]$ are $\lambda\sigma$-terms in $\eta$-long normal form and the equation is not solved.