Introduction to Unification Theory

Higher-Order Unification

Temur Kutsia

RISC, Johannes Kepler University of Linz, Austria
kutsia@risc.jku.at

Overview

Outline

Introduction

Preliminaries

Higher-Order Unification Procedure

Introduction

- In first order unification, we were not allowed to replace a variable with a function.
- However, it makes sense to ask to find, e.g., a function that when applied to an object gives again this object: Find an F such that $F(a) = a$.
- F: Higher-order variable, appears at functional position.
- Can be solved, e.g., with the identity function or with the constant function a.
- Higher-order equations.
- Solving method: Higher-order unification.

Introduction

- Higher-order unification is fundamental in automating higher-order reasoning.
- Used in logical frameworks, logic programming, program synthesis, program transformation, type inferencing, computational linguistics, etc.
- Much more complicated than first-order unification (undecidable, of type zero, nonterminating, ...).
- In this lecture: Introduction to higher-order unification.

Outline

Simply Typed λ-Calculus

- Simply type λ-calculus is our term language.
- In this section: Definitions and elementary properties.
  - Types
  - Terms
  - Substitutions
  - Reduction
  - Unification

Types

Types

Consider a finite set whose elements are called *atomic types* (or *base types*). Then:

- Atomic types are types,
- If T and U are types than $T \to U$ is a type.

The expression $T_1 \to T_2 \to \cdots \to T_n \to U$ is a notation for the type $T_1 \to (T_2 \to \cdots \to (T_n \to U) \ldots)$.

Types

Order of a Type

- $o(T) = 1$ if T is atomic.
- $o(T \to U) = max\{1 + o(T), o(U)\}$.

Example

Let T_1, T_2, T_3 be atomic types, then

- $o(T_1 \to T_2 \to T_3) = 2$.
- $o((T_1 \to T_2) \to T_3) = 3$.

Terms

Assumptions:

- Consider finite set of constants.
- To each constant a type is assigned.
- For each atomic type there is at least one constant.
- For each type there is an infinite set of variables.
- Two different types have disjoint sets of variables.

λ-Terms

- Constants are terms.
- Variables are terms.
- If t and s are terms then $(t\, s)$ is a term.
- If x is a variable and t is a term then $\lambda x.\, t$ is a term.

The expression $(t\, s_1\ \ldots\ s_n)$ is a notation for the term $(\ldots(t\, s_1)\ \ldots\ s_n)$

Terms

- $\lambda x.\, t$ is a function where λx is the λ-abstraction and t is the body. Intuitively, it is a function $x \mapsto t$.
- In $\lambda x.\, t$, λx is a binder for x in t. Occurrences of x in t are *bound*.
- $(t\, s)$ is an application where function t is applied to the argument s.

Terms

Type of a Term

A term t is said to have the type T if either

- t is a constant of type T,
- t is a variable of type T,
- $t = (r\, s)$, r has type $U \to T$ and s has type U for some U,
- $t = \lambda x.\, s$, the variable x has type U, the term s has type V and $T = U \to V$.

- A term t is said to be *well-typed* if there exists a type T such that t has type T.
- In this case T is unique and it is called *the type of* t.
- We consider only well-typed terms.

Order

Order of a Symbol, Language

- The order of a function symbol or a variable is the order of its type.
- A language of order n is one which allows function symbols of order at most $n+1$ and variables of order at most n.

Formalization of the conventions:

- First order term denotes an individual.
- Second order term denotes a function on individuals.
- etc.

Free Variables

- $vars(t)$: The set of variables occurring in the term t.
- An occurrence of a variable in a term is *free* if it is not bound.
- The set of variables that occur freely in t, denoted $fvars(t)$:
  - $fvars(c) = \emptyset$, where c is a constant.
  - $fvars(x) = \{x\}$.
  - $fvars((s\, r)) = fvars(s) \cup fvars(r)$.
  - $fvars(\lambda x.\, s) = fvars(s) \setminus \{x\}$.
- Closed term: A term without free variables.

Free Variables

Example

- $fvars(\lambda x.\, x) = \emptyset$.
  (Closed term)
- $fvars(\lambda x.\, y) = \{y\}$.
- $fvars(((\lambda x.\, x)\, x)) = \{x\}$.
  (x has a bound occurrence as well)

Substitution

- We reuse the definition of substitution as finite mapping from the previous lectures, but in addition require that it preserves types.
- Hence, if $x \mapsto t$ is a binding of a substitution, x and t have the same type.
- The definitions of composition, more general substitution, etc. will also be reused.

Replacement in a Term

Replacement in a Term

Let $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ be a substitution and t be a term, then the term $t\langle\sigma\rangle$ is defined as follows:

- $c\langle\sigma\rangle = c$.
- $x_i\langle\sigma\rangle = t_i$.
- $x\langle\sigma\rangle = x$, if $x \notin \{x_1, \ldots, x_n\}$.
- $(s\,r)\langle\sigma\rangle = (s\langle\sigma\rangle\, r\langle\sigma\rangle)$.
- $(\lambda x.\, s)\langle\sigma\rangle = (\lambda x.\, s\langle\sigma\rangle)$.

Example

- $(\lambda x.\, x)\langle\{x \mapsto y\}\rangle = \lambda x.\, y$.
- $(\lambda y.\, x)\langle\{x \mapsto y\}\rangle = \lambda y.\, y$ (variable capture).

α-Equivalence

α-Equivalence

- $c \equiv_\alpha c$.
- $x \equiv_\alpha x$.
- $(t\,s) \equiv_\alpha (t'\,s')$ if $t \equiv_\alpha t'$ and $s \equiv_\alpha s'$.
- $\lambda x.\,t \equiv_\alpha \lambda y.\,s$ if $t\langle\{x \mapsto z\}\rangle \equiv_\alpha s\langle\{y \mapsto z\}\rangle$ for some variable z different from x and y and occurring neither in t nor in s.

Example

- $\lambda x.\,x \equiv_\alpha \lambda y.\,y$.

- α-equivalence is an equivalence relation.
- Application and abstraction are compatible with α-equivalence.

Substitution in a Term

Substitution in a Term

Let $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ be a substitution and t be a term, then the term $t\sigma$ is defined as follows:

- $c\sigma = c$.
- $x_i\sigma = t_i$.
- $x\sigma = x$, if $x \notin \{x_1, \ldots, x_n\}$.
- $(s\, r)\sigma = (s\sigma\, r\sigma)$.
- $(\lambda x.\, s)\sigma = (\lambda y.\, s\{x \mapsto y\}\sigma)$, where y is a fresh variable of the same type as x.

Since the choice of fresh variable is arbitrary, the substitution operation is defined on α-equivalence classes.

Substitution in a Term

Example

- $(\lambda x.\, x)\{x \mapsto y\} = \lambda z.\, z$.
- $(\lambda y.\, x)\{x \mapsto y\} = \lambda z.\, y$ (no variable capture).
- $(x\, \lambda x.\, (x\, y))\{x \mapsto \lambda z.z\} = (\lambda z.z\ \lambda u.\, (u\, y))$.

Reduction

- Intuition: Function evaluation.
- For instance, evaluating function $f : x \mapsto x + 1$ at 2: $f(2) = 2 + 1$.
- As λ-terms: $((\lambda x. x + 1)\ 2) \rhd x + 1\{x \mapsto 2\} = 2 + 1$. ($\beta$-reduction)

Reduction

Formally:

$\beta\eta$-Reduction

- β-reduction: $((\lambda x.s)\, t) \rhd s\{x \mapsto t\}$.
- η-reduction: $(\lambda x.(t\, x)) \rhd t$, if $x \notin \mathit{fvars}(t)$.

Propagates into contexts:

- If $s \rhd s'$ then $(s\, t) \rhd (s'\, t)$.
- If $t \rhd t'$ then $(s\, t) \rhd (s\, t')$.
- If $t \rhd t'$ then $\lambda x.\, t \rhd \lambda x.\, t'$.

Reduction

$\rhd^*$ - reflexive-transitive closure of $\rhd$.

Facts:

- $\beta\eta$-Reduction preserves types.
- If $s \rhd^* t$ then $s\sigma \rhd^* t\sigma$.
- Each term has a unique $\beta\eta$-normal form modulo α-equivalence.

Reduction

Example

$$
\begin{aligned}
\lambda x.(f\ ((\lambda y.(y\,x))\ \lambda z.z)) &\rhd_\beta \lambda x.(f\ ((\lambda z.z)\ x)) \\
&\rhd_\beta \lambda x.(f\,x) \\
&\rhd_\eta f
\end{aligned}
$$

Long Normal Form

Long Normal Form

Assume

- $t = \lambda x_1 . \dots . \lambda x_m . (r\, s_1 \ \dots \ s_k)$ is in the $\beta\eta$-normal form,
- $T_1 \to \cdots \to T_n \to U$ is a type of t,
- U is atomic and $n \geq m$.

Then the long normal form of t is the term

$$t' = \lambda x_1 . \dots . \lambda x_m . \lambda x_{m+1} . \dots . \lambda x_n . (r\, s'_1 \ \dots s'_k\, x'_{m+1} \ \dots \ x'_n)$$

where

- s'_i is the long normal form of s_i.
- x'_i is the long normal form of x_i.

The long normal form of any term is that of its normal form.

Since t is in the normal form, r (called the *head* of t) is either a constant or a variable.

Long Normal Form

Example

Let the type of f be $T_1 \to T_2 \to U$, with U atomic.
Let t be $\lambda x.(f\;((\lambda y.(y\,x))\;\lambda z.z))$.

- The long normal form of t is $\lambda x.\lambda y.(f\,x\,y)$.
- $\lambda x.\lambda y.(f\,x\,y)$ is a long normal form of $\lambda x.(f\,x)$ as well, which is a β-normal form of t.
- In general, to compute long normal form, it is not necessary to perform η-reductions.

Long Normal Form

- In the rest, “normal form” stands for “long normal form”.
- Notation: We write

$$\lambda x_1. \dots \lambda x_n.\ r(t_1, \dots, t_m)$$

for

$$\lambda x_1. \dots \lambda x_n.\ (r\, t_1\ \dots\ t_m)$$

in normal form. r is either a constant or a variable.

Outline

Higher Order Unification

Higher-Order Unification Problem, Unifier

- Higher-Order Unification problem: a finite set of equations

$$\Gamma = \{s_1 \doteq^? t_1, \ldots, s_n \doteq^? t_n\},$$

  where s_i, t_i are λ-terms.
- Unifier of Γ: a substitution σ such that $s_i\sigma$ and $t_i\sigma$ have the same normal form for each $1 \leq i \leq n$.

We will use capital letters to denote free variables in unification problems.

Higher Order Unification

Example

- $\Gamma = \{F(f(a,b)) \doteq^? f(F(a),b)\}$.
- Unifier: $\sigma_1 = \{F \mapsto \lambda x.f(x,b)\}$.
- Justification:

$$F(f(a,b))\sigma_1 = ((\lambda x.f(x,b))\,f(a,b)) \rhd_\beta f(f(a,b),b).$$
$$f(F(a),b)\sigma_1 = f(((\lambda x.f(x,b))\,a),b) \rhd_\beta f(f(a,b),b).$$

Higher Order Unification

Example

- $\Gamma = \{F(f(a,b)) \doteq^? f(F(a),b)\}$.
- Another unifier: $\sigma_2 = \{F \mapsto \lambda x.f(f(x,b),b)\}$.
- Justification:

$$F(f(a,b))\sigma_2 = ((\lambda x.f(f(x,b),b))f(a,b)) \rhd_\beta f(f(f(a,b),b),b).$$
$$f(F(a),b)\sigma_2 = f(((\lambda x.f(f(x,b),b))\,a),b) \rhd_\beta f(f(f(a,b),b),b).$$

Higher Order Unification

Example

- $\Gamma = \{F(f(a,b)) \doteq^? f(F(a),b)\}$.
- Infinitely many unifiers, of the shape

$$\{F \mapsto \lambda x.\, f(\ldots f(x,b),\ldots b)\}.$$

- Incomparable wrt instantiation quasi-ordering.
- Minimal complete set of unifiers.
- There are problems for which this set does not exist!

Higher Order Unification Is of Type 0

- Unification problem: $\Gamma = \{F(\lambda x.\, G(x), a) \doteq^{?} F(\lambda x.\, G(x), b)\}$.
- Complete set of solutions (together with the instance terms):

$$\sigma = \{F \mapsto \lambda x.\lambda y.\, H(x)\} \qquad H(\lambda x.\, G(x))$$

$$\sigma_0 = \{F \mapsto \lambda x.\, x, G \mapsto \lambda x.\, Y\} \qquad Y$$

$$\sigma_1 = \{F \mapsto \lambda x.\lambda y.\, G_1(x, x(H_1^1(x,y))), G \mapsto \lambda x.\, Y\} \quad G_1(\lambda x.\, Y, Y)$$

$$\sigma_2 = \{F \mapsto \lambda x.\lambda y.\, G_2(x, x(H_1^2(x,y)), x(H_2^2(x,y))), G \mapsto \lambda x.\, Y\}$$

$$G_2(\lambda x.\, Y, Y, Y)$$

$$\dots$$

$$\sigma_n = \{F \mapsto \lambda x.\lambda y.\, G_n(x, x(H_1^n(x,y)), \dots, x(H_n^n(x,y))), G \mapsto \lambda x.\, Y\}$$

$$G_n(\lambda x.\, Y, Y, \dots, Y) \quad (n\ Y's)$$

Higher Order Unification Is of Type 0

- Unification problem: $\Gamma = \{F(\lambda x.\, G(x), a) \doteq^? F(\lambda x.\, G(x), b)\}$.
- Complete set of solutions:

$$
\begin{aligned}
\sigma &= \{F \mapsto \lambda x.\lambda y.\, H(x)\} \\
\sigma_0 &= \{F \mapsto \lambda x.\, x, G \mapsto \lambda x.\, Y\} \\
\sigma_n &= \{F \mapsto \lambda x.\lambda y.\, G_n(x, x(H_1^n(x,y)), \dots, x(H_n^n(x,y))), G \mapsto \lambda x.\, Y\}
\end{aligned}
$$

- No mcsu. For all $i, j > i$: $\sigma_i \not\lessdot^{\{F,G\}} \sigma_j$, $\sigma \not\lessdot^{\{F,G\}} \sigma_i$, $\sigma_i \not\lessdot^{\{F,G\}} \sigma$, and $\sigma_i \doteq^{\{F,G\}} \sigma_{i+1}\vartheta_i$ where

$$
\begin{aligned}
\vartheta_i = \{&G_{i+1} \mapsto \lambda x.\lambda y_1.\dots.\lambda y_{i+1}.G_i(x, y_1, \dots, y_i), \\
&H_1^{i+1} \mapsto H_1^i, \dots, H_i^{i+1} \mapsto H_i^i\}
\end{aligned}
$$

- Infinite descending chain: $\sigma_1 \gtrdot^{\{F,G\}} \sigma_2 \gtrdot^{\{F,G\}} \dots$

Higher Order Unification Is of Type 0

- Unification problem: $\Gamma = \{F(\lambda x. G(x), a) \doteq^? F(\lambda x. G(x), b)\}$.
- The problem is of third order.
- Higher-order unification of the order 3 and above is of type 0.
- Second order unification is infinitary.

Higher Order Unification Is Undecidable

- Idea: Reduce Hilbert's 10th problem to a higher-order unification problem.
- Hilbert's 10th problem is undecidable: There is no algorithm that takes as input two polynomials $P(X_1, \ldots, X_n)$ and $Q(X_1, \ldots, X_n)$ with natural coefficients and answers if there exist natural numbers $m_1, \ldots, m_n$ such that

$$P(m_1, \ldots, m_n) = Q(m_1, \ldots, m_n).$$

- Reduction requires to represent
  - natural numbers,
  - addition,
  - multiplication

  in terms of higher-order unification.

Higher Order Unification Is Undecidable

Representation (Goldfarb 1981):

- Natural number n represented as a λ-term denoted by $\overline{n}$:

  $$\lambda x.g(a, g(a, \ldots g(a, x) \ldots))$$

  with n occurrences of g and a. The type of g is $i \to i \to i$ and the type of a is i. Such terms are called Goldfarb numbers.

- Goldfarb numbers are exactly those that solve the unification problem

  $$\{g(a, X(a)) \doteq^? X(g(a, a))\}$$

  and have the type $i \to i$.

Higher Order Unification Is Undecidable

Representation:

- Addition is represented by the λ-term add:

$$\lambda n.\lambda m.\lambda x.\ n(m(x)).$$

- Multiplication is represented by the higher-order unification problem

$$\{Y(a,b,g(g(X_3(a),X_2(b)),a)) \doteq^? g(g(a,b),Y(X_1(a),g(a,b),a))$$
$$Y(b,a,g(g(X_3(b),X_2(a)),a)) \doteq^? g(g(b,a),Y(X_1(b),g(a,a),a))\}$$

that has a solution $\{X_1 \mapsto \overline{m_1}, X_2 \mapsto \overline{m_2}, X_3 \mapsto \overline{m_3}, Y \mapsto t\}$ for some t iff $m_1 \times m_2 = m_3$.

Higher Order Unification Is Undecidable

Reduction from Hilbert's 10th problem:

- Every equation $P(X_1, \ldots, X_n) = Q(X_1, \ldots, X_n)$ can be decomposed into a system of equations of the form:

  $X_i + X_j = X_k, \quad X_i \times X_j = X_k, \quad X_i = m.$
- With each such system associate a unification problem containing
  - for each X_i an equation $g(a, X_i(a)) \doteq^? X_i(g(a, a))$,
  - for each $X_i + X_j = X_k$ the equation $add(X_i, X_j) \doteq^? X_k$,
  - for each $X_i \times X_j = X_k$ the two equations used to define multiplication,
  - for each $X_i = m$ the equation $X_i \doteq^? \overline{m}$.

Second Order Unification Is Undecidable

- The reduction implies undecidability of higher-order unification.
- Since the reduction is actually to second-order unification, the result is sharper:

Theorem

Second-order unification is undecidable.

For the details of undecidability of second-order unification, see

W. D. Goldfarb
The undecidability of the second-order unification problem.
Theoretical Computer Science **13**, 225–230.

Higher-Order Unification Procedure

- Higher-order semi-decision procedure is easy to design:
  1. Enumerate all substitutions (in fact, it is enough to enumerate all closed substitutions).
  2. For a given unification problem, take the first untried substitution and check whether it is a solution.
  3. If yes, stop with success. If not, mark the substitution as tried and iterate.
- Checking is not hard: Apply the substitution to both sides of each equation, normalize, and compare the normal forms.
- If the problem is solvable, the procedure will detect it after finite steps.
- Then... why to bother with looking for another unification procedure?

Higher-Order Unification Procedure

Why to look for a “better” procedure?

- To find solutions faster.
- To report failure for many unsolvable cases.
- To reduce redundancy.
- etc.

Higher-Order Unification Procedure

- System: a pair $P; \sigma$, where P is a higher-order unification problem and σ is a substitution.
- Procedure is given by transformation rules on systems.
- The description essentially follows the paper

W. Snyder and J. Gallier.
Higher-Order Unification Revisited: Complete Sets of Transformations.
J. Symbolic Computation, **8**(1–2), 101–140, 1989.

Important Observation

- Flex-flex equation has a form

$$\lambda x_1. \ldots \lambda x_k. F(s_1, \ldots, s_n) \doteq^? \lambda x_1. \ldots \lambda x_k. G(t_1, \ldots, t_m).$$

  The head of both sides are free variables.
- Any flex-flex equation is solvable. Just take

$$\{F \mapsto \lambda y_1. \ldots \lambda y_n. c, \;\; G \mapsto \lambda y_1. \ldots \lambda y_m. c\}.$$

- The appropriate c always exists because for each type we have at least one constant of that type.
- Flex-flex equations may introduce infinite branching in the search tree (very undesirable property).
- Idea: Do not try to solve flex-flex equations. Assume them solved. Preunification.

Preunification

Preunifier

- Let $\cong$ be the least congruence relation on the set of λ-terms that contains the set of flex-flex pairs.
- A substitution σ is a preunifier for a unification problem $\{s_1 \doteq^? t_1, \ldots, s_n \doteq^? t_n\}$ iff

$$\mathit{normal\text{-}form}(s_i\sigma) \cong \mathit{normal\text{-}form}(t_i\sigma)$$

  for each $1 \leq i \leq n$.

Convention

- $\overline{x_n}$ abbreviates $x_1, \ldots, x_n$.
- $\lambda\overline{x_n}$ abbreviates $\lambda x_1. \ldots .\lambda x_n$.

One Technical Notion

Partial Binding

A partial binding of type $T_1 \to \cdots \to T_n \to U$ (U atomic) is a term of the form

$$\lambda\overline{x_n}.\ l(\lambda\overline{y^1_{m_1}}.H_1(\overline{x_n}, \overline{y^1_{m_1}}), \ldots, \lambda\overline{y^k_{m_k}}.H_k(\overline{x_n}, \overline{y^k_{m_k}}))$$

where l is a constant or a variable, and

- the type of x_i is T_i for $1 \leq i \leq n$,
- the type of l is $S_1 \to \cdots \to S_k \to U$, where S_i is $R^1_i \to \cdots \to R^i_{m_i} \to S'_i$ (S'_i atomic) for $1 \leq i \leq k$,
- the type of y^i_j is R^i_j for $1 \leq i \leq k$ and $1 \leq j \leq m_i$.
- the type of H_i is $T_1 \to \cdots \to T_n \to R^i_1 \to \cdots \to R^i_{m_i} \to S'_i$ for $1 \leq i \leq k$.

Partial Binding

$$\lambda\overline{x_n}.\ l(\lambda\overline{y^1_{m_1}}.H_1(\overline{x_n},\overline{y^1_{m_1}}),\ldots,\lambda\overline{y^k_{m_k}}.H_k(\overline{x_n},\overline{y^k_{m_k}}))$$

- Imitation binding: l is a constant or a free variable.
- (i^{th}) Projection binding: l is x_i.
- A partial binding t is appropriate to F if t and F have the same types.
- $F \mapsto t$: Appropriate partial (imitation, projection) binding if t is partial (imitation, projection) binding appropriate to F.

Higher-Order Preunification Procedure

- The inference system $\mathcal{U}_{HOP}$ consists of the rules:
  - **Trivial**
  - **Decomposition**
  - **Variable Elimination**
  - **Orient**
  - **Imitation**
  - **Projection**
- The rules transform systems: pairs $\Gamma; \sigma$, where Γ is a higher-order unification problem and σ is a substitution.
- A system $\Gamma; \sigma$ is in presolved form if Γ is either empty or consists of flex-flex equations only.

Higher-Order Preunification Procedure. Rules

Trivial: $\{t \doteq^? t\} \cup P'; \vartheta \Longrightarrow P'; \vartheta$

Decomposition:

$$\{\lambda\overline{x_k}.\ l(s_1, \ldots, s_n) \doteq^? \lambda\overline{x_k}.\ l(t_1, \ldots, t_n)\} \cup P'; \vartheta \Longrightarrow$$
$$\{\lambda\overline{x_k}.\ s_1 \doteq^? \lambda\overline{x_k}.\ t_1, \ldots, \lambda\overline{x_k}.\ s_n \doteq^? \lambda\overline{x_k}.\ t_n,\} \cup P'; \vartheta.$$

where l is either a constant or one of the bound variables $x_1, \ldots, x_k$.

Variable Elimination:

$$\{\lambda x_1. \ldots \lambda x_k.\ F(x_1, \ldots, x_k) \doteq^? t\} \cup P'; \vartheta \Longrightarrow P'\{F \mapsto t\}; \vartheta\{F \mapsto t\}.$$

If $F \notin fvars(t)$

Higher-Order Preunification Procedure. Rules

Orient:

$$\{\lambda\overline{x_k}.\, l(t_1,\ldots,t_m) \doteq^? \lambda\overline{x_k}.\, F(s_1,\ldots,s_n)\} \cup P'; \vartheta \Longrightarrow$$
$$\{\lambda\overline{x_k}.\, F(s_1,\ldots,s_n) \doteq^? \lambda\overline{x_k}.\, l(t_1,\ldots,t_m)\} \cup P'; \vartheta$$

where l is not a free variable.

Imitation:

$$\{\lambda\overline{x_k}.\, F(s_1,\ldots,s_n) \doteq^? \lambda\overline{x_k}.\, f(t_1,\ldots,t_m)\} \cup P'; \vartheta \Longrightarrow$$
$$\{\lambda\overline{x_k}.\, f(\lambda\overline{z^1_{r_1}}.\, H_1(s_1,\ldots,s_n,\overline{z^1_{r_1}}),\ldots,\lambda\overline{z^m_{r_m}}.\, H_m(s_1,\ldots,s_n,\overline{z^m_{r_m}}))\sigma$$
$$\doteq^? \lambda\overline{x_k}.\, f(t_1,\ldots,t_m)\sigma\} \cup P'\sigma; \vartheta\sigma$$

where

- $\sigma = \{F \mapsto \lambda\overline{y_n}.\, f(\lambda\overline{z^1_{r_1}}.\, H_1(\overline{y_n},\overline{z^1_{r_1}}),\ldots,\lambda\overline{z^m_{r_m}}.\, H_m(\overline{y_n},\overline{z^m_{r_m}}))\}$, appropriate imitation binding.
- $H_1,\ldots,H_m$ are fresh variables.

Higher-Order Preunification Procedure. Rules

Projection:

$$\{\lambda\overline{x_k}.\, F(s_1,\ldots,s_n) \doteq^? \lambda\overline{x_k}.\, l(t_1,\ldots,t_m)\} \cup P'; \vartheta \Longrightarrow$$
$$\{\lambda\overline{x_k}.\, s_i(\lambda\overline{z^1_{r_1}}.\, H_1(s_1,\ldots,s_n,\overline{z^1_{r_1}}),\ldots,\lambda\overline{z^m_{r_m}}.\, H_m(s_1,\ldots,s_n,\overline{z^m_{r_m}}))\sigma$$
$$\doteq^? \lambda\overline{x_k}.\, l(t_1,\ldots,t_m)\sigma\} \cup P'\sigma; \vartheta\sigma$$

where

- l is either a constant or one of the bound variables $x_1,\ldots,x_k$,
- $\sigma = \{F \mapsto \lambda\overline{y_n}.\, y_i(\lambda\overline{z^1_{r_1}}.\, H_1(\overline{y_n},\overline{z^1_{r_1}}),\ldots,\lambda\overline{z^m_{r_m}}.\, H_m(\overline{y_n},\overline{z^m_{r_m}}))\}$, appropriate projection binding.
- $H_1,\ldots,H_m$ are fresh variables.

Higher-Order Preunification Procedure. Control

In order to solve a higher-order unification problem Γ:

- Create an initial system $\Gamma; \varepsilon$.
- Apply successively rules from $\mathcal{U}_{HOP}$, building a complete (finitely branching, but potentially infinite) tree of derivations.
- If no rule can be applied to a node, and it contains at least one equation that is not flex-flex, then extend the branch with $\perp$, indicating failure.
- Successful leaves contain presolved systems.
- If $\Delta; \sigma$ is a successful leaf, σ is a solution of Γ computed by the higher-order preunification procedure.

Higher-Order Preunification. Major Results

Theorem (Soundness)

If $\Gamma; \varepsilon \Longrightarrow^* \Delta; \sigma$ *and* Δ *is in presolved form, then* $\sigma|_{\mathit{fvars}(\Gamma)}$ *is a preunifier of* Γ*.*

Theorem (Completeness)

If ϑ *is a preunifier of* Γ*, then there exists a sequence of transformations* $\Gamma; \varepsilon \Longrightarrow^* \Delta; \sigma$ *such that* Δ *is in presolved form, and* $\sigma \leq_{\beta}^{\mathit{fvars}(\Gamma)} \vartheta$*.*

Higher-Order Preunification. Optimization

- The procedure can be optimized by stripping off the binder λx when x does not occur in the body.
- For instance, Elimination rule does not apply to $\lambda x.\lambda y.\, P(x) \doteq^{?} \lambda x.\lambda y.\, f(a)$
- After removing λy from both sides, Elimination can be applied directly.

Higher-Order Preunification. Examples

Example

- Unification problem $\{F(f(a)) \doteq^? f(F(a))\}$.
- The preunification procedure enumerates the complete set of (pre)unifiers that is infinite.
- Here we show only two derivations.

$$\{F(f(a)) \doteq^? f(F(a))\}; \varepsilon$$

$$\Longrightarrow_{\mathit{Proj}} \{f(a) \doteq^? f(a)\}; \{F \mapsto \lambda x.\, x\}$$

$$\Longrightarrow_{\mathit{Tr}} \emptyset; \{F \mapsto \lambda x.\, x\}$$

$$\{F(f(a)) \doteq^? f(F(a))\}; \varepsilon$$

$$\Longrightarrow_{\mathit{Imit}} \{f(G(f(a))) \doteq^? f(f(G(a)))\}; \{F \mapsto \lambda x.\, f(G(x))\}$$

$$\Longrightarrow_{\mathit{Dec}} \{G(f(a)) \doteq^? f(G(a))\}; \{F \mapsto \lambda x.\, f(G(x))\}$$

$$\Longrightarrow_{\mathit{Proj}} \{f(a) \doteq^? f(a)\}; \{F \mapsto \lambda x.\, f(x), G \mapsto \lambda x.\, x\}$$

$$\Longrightarrow_{\mathit{Tr}} \emptyset; \{F \mapsto \lambda x.\, f(x), G \mapsto \lambda x.\, x\}$$

Higher-Order Preunification. Examples

Example

- Problem $\{\lambda x.\ F(f(x,G)) \doteq^? \lambda x.\ g(f(x,G_1),f(x,G_2))\}$.
- Here we show only the successful derivation.

$$\{\lambda x.\ F(f(x,G)) \doteq^? \lambda x.\ g(f(x,G_1),f(x,G_2))\};\varepsilon$$

$$\Longrightarrow_{\mathit{Imit}} \{\lambda x.\ g(H_1(f(x,G)),H_2(f(x,G))) \doteq^? \lambda x.\ g(f(x,G_1),f(x,G_2))\};$$
$$\{F \mapsto \lambda y.\ g(H_1(y),H_2(y))\}$$

$$\Longrightarrow_{\mathit{Dec},\mathit{Proj},\mathit{Proj}} \{\lambda x.\ f(x,G) \doteq^? \lambda x.\ f(x,G_1),\ \lambda x.\ f(x,G) \doteq^? \lambda x.\ f(x,G_2)\};$$
$$\{F \mapsto \lambda y.\ g(y,y), H_1 \mapsto \lambda y.\ y, H_2 \mapsto \lambda y.\ y\}$$

$$\Longrightarrow_{\mathit{Dec},\mathit{Tr},\mathit{Dec},\mathit{Tr}} \{\lambda x.\ G \doteq^? \lambda x.\ G_1,\ \lambda x.\ G \doteq^? \lambda x.\ G_2\};$$
$$\{F \mapsto \lambda y.\ g(y,y), H_1 \mapsto \lambda y.\ y, H_2 \mapsto \lambda y.\ y\}$$

$$\Longrightarrow^2_{\mathit{Elim}} \emptyset; \{F \mapsto \lambda y.\ g(y,y), H_1 \mapsto \lambda y.\ y, H_2 \mapsto \lambda y.\ y, G \mapsto G_2, G_1 \mapsto G_2\}$$

Higher-Order Preunification. Examples

Example

- Problem $\{\lambda x.\, F(x,a) \doteq^? \lambda x.\, f(G(a,x))\}$.
- One of the successful derivations.

$$\{\{\lambda x.\, F(x,a) \doteq^? \lambda x.\, f(G(a,x))\}; \varepsilon$$

$$\Longrightarrow_{Imit} \{\lambda x.\, f(H(x,a)) \doteq^? \lambda x.\, f(G(a,x))\}; \{F \mapsto \lambda y_1.\lambda y_2.\, f(H(y_1,y_2))\}$$

$$\Longrightarrow_{Dec} \{\lambda x.\, H(x,a) \doteq^? \lambda x.\, G(a,x)\}; \{F \mapsto \lambda y_1.\lambda y_2.\, f(H(y_1,y_2))\}$$

Flex-flex.

Decidable Subcases

Some decidable subcases of higher-order unification:

- Monadic second-order unification. Terms do not contain constants of arity greater than 1.
  Example: $\{\lambda x.f(F(x)) \doteq^? \lambda x.F(f(x))\}$.
- Second-order unification with linear occurrences of second-order variables.
- Unification with higher-order patterns. Pattern is a term t such that for every subterm of the form $F(s_1, \ldots, s_n)$, the s's are distinct variables bound in t.
  Example: $\{\lambda x.\lambda y.\ F(x) \doteq^? \lambda x.\lambda y.\ c(G(y, x))\}$.
- Higher-order matching. One side in the equations is a closed term.
  Example. $\{\lambda x.\ F(x, \lambda y.\ y) \doteq^? \lambda x.f(x, a)\}$.
- Stratified second-order unification.
- Bounded second-order unification.