A Variant of Higher-Order Anti-Unification

Alexander Baumgartner
Temur Kutsia
Jordi Levy
Mateu Villaret

Anti-Unification Problem

- Given two terms t_1, t_2.
- Find a generalization term t such that t_1, t_2 are instances of t.
- Interesting generalizations are the least general ones (lggs).

Input terms	$f(a, g(b), b)$ and $f(a, g(c), c)$
Generalization	$f(a, x, y)$
Lgg	$f(a, g(x), x)$

Goal / Setting

- The Setting:
  - Input: Simply-typed lambda terms t_1, t_2.
  - Output: Simply-typed higher-order pattern generalization of t_1, t_2.

Goal / Setting

- The Setting:
  - Input: Simply-typed lambda terms t_1, t_2.
  - Output: Simply-typed higher-order pattern generalization of t_1, t_2.
- Provide an anti-unification algorithm to compute lggs:
  - Design algorithm,
  - Prove correctness,
  - Complexity analysis,
  - Implementation.

Simply Typed Lambda Calculus

- Basic types: $\delta_1, \delta_2, \ldots$
- Type constructor: $\rightarrow$
- Types: $\tau ::= \delta \mid \tau \rightarrow \tau$
- Variables: $X, Y, x, y, \ldots$
- Constants: $c, f, g, \ldots$

Simply Typed Lambda Calculus

- λ-terms $(t, s, \ldots)$ are built using the grammar:

$$t ::= x \mid c \mid \lambda x.t \mid t_1 \ t_2$$

- Terms are assumed to be written in η-long β-normal form: $t = \lambda x_1, \ldots, x_n.h(t_1, \ldots, t_m)$ were $h(t_1, \ldots, t_m)$ has a basic type and h is a constant or variable.
- The head of t is defined as $\mathrm{Head}(t) = h$.

Substitution and Generalization

Definition (Substitution σ)

Finite set of pairs $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where X_i and t_i have the same type and the X's are pairwise distinct variables.

- $t\sigma$ for substitution application.
- $t \preceq s$ if there exists σ such that $t\sigma = s$.

Substitution and Generalization

Definition (Substitution σ)

Finite set of pairs $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where X_i and t_i have the same type and the X's are pairwise distinct variables.

- $t\sigma$ for substitution application.
- $t \preceq s$ if there exists σ such that $t\sigma = s$.

Definition (Generalization and least general generalization)

A term t is a generalization of t_1 and t_2, if $t \preceq t_1$ and $t \preceq t_2$. It is a lgg, if there is no generalization s which satisfies $t \prec s$.

Higher-Order Patterns

- In general, there is no unique higher-order lgg.

  Input terms: $f(g(a, b), c)$ and $f(c, h(a))$

  Higher-order lggs: $f(X, Y)$, $X(c, Y(a))$ and $X(Y(a), c)$
- Consider special classes to guarantee uniqueness.

Higher-Order Patterns

- In general, there is no unique higher-order lgg.

Input terms: $f(g(a,b),c)$ and $f(c,h(a))$
Higher-order lggs: $f(X,Y)$, $X(c,Y(a))$ and $X(Y(a),c)$

- Consider special classes to guarantee uniqueness.

Definition (Higher-order pattern)

Arguments of free variables are distinct bound variables.

- $\lambda x.f(X(x),Y)$, $f(c,\lambda x.x)$ and $\lambda x.\lambda y.X(\lambda z.x(z),y)$ are patterns.
- $\lambda x.f(X(X(x)),Y)$, $f(X(c),c)$ and $\lambda x.\lambda y.X(x,x)$ are not patterns.

Input terms: $f(g(a,b),c)$ and $f(c,h(a))$
Pattern-lgg: $f(X,Y)$

Input / Output

- Input: Higher-order terms t_1 and t_2 in η-long β-normal form.
- Output: Unique higher-order pattern generalization of t_1 and t_2.

Input terms	$\lambda x, y. f(g(x, x, y), g(x, y, y))$ $\lambda x, y. f(h(x, x, y), h(x, y, y))$
Pattern-lgg	$\lambda x, y. f(Y_1(x, y), Y_2(x, y))$
No pattern	$\lambda x, y. f(Z(x, x, y), Z(x, y, y))$

Anti-Unification Problem (AUP)

Definition (Anti-unification problem)

An anti-unification problem is a triple $X(\vec{x}) : t \triangleq s$ where

- $\lambda\vec{x}.X(\vec{x})$, $\lambda\vec{x}.t$, and $\lambda\vec{x}.s$ are terms of the same type,
- t and s are in η-long β-normal form,
- X does not occur in t and s.

Example: $X(x, y) : f(x, x, y) \triangleq g(x, x, y)$

Rule Based System $\mathfrak{P}$

- $\mathfrak{P}$ operates on a triple A; S; σ.
  - A is a set of AUPs like $\{X_1(\overrightarrow{x_1}) : t_1 \triangleq s_1, \ldots, X_n(\overrightarrow{x_n}) : t_n \triangleq s_n\}$.
  - S is a set of already solved AUPs (the store).
  - σ is a substitution which maps variables to patterns.
- Each generalization variable X_i occurs only once in $A \cup S$.

Compute Pattern Generalization

- Initialize $A; S; \sigma$ with $\{X : t \triangleq s\}; \emptyset; \varepsilon$ where X is fresh variable.
- Apply the rules of $\mathfrak{P}$ successively as long as possible.
- Final system has the form $\emptyset; S; \sigma$.
- Result $X\sigma$ is a pattern-lgg.
- Computed pattern-lgg is unique modulo α-equivalence.
- S contains all the differences between t and s.

The Rules of $\mathfrak{P}$

- Y's always denote fresh variables of the corresponding types.
- Dec: **Decomposition**
  $\{X(\vec{x}) : h(t_1, \ldots, t_m) \triangleq h(s_1, \ldots, s_m)\} \mathbin{\dot\cup} A;\ S;\ \sigma \Longrightarrow$
  $\{Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_m(\vec{x}) : t_m \triangleq s_m\} \cup A;\ S;$
  $\sigma\{X \mapsto \lambda\vec{x}.h(Y_1(\vec{x}), \ldots, Y_m(\vec{x}))\}$,
  where h is a constant or $h \in \vec{x}$.
- Abs: **Abstraction**
  $\{X(\vec{x}) : \lambda y.t \triangleq \lambda z.s\} \mathbin{\dot\cup} A;\ S;\ \sigma \Longrightarrow$
  $\{Y(\vec{x}, y) : t \triangleq s\{z \mapsto y\}\} \cup A;\ S;\ \sigma\{X \mapsto \lambda\vec{x}, y.Y(\vec{x}, y)\}$.
- Sol: **Solve**
  $\{X(\vec{x}) : t \triangleq s\} \mathbin{\dot\cup} A;\ S;\ \sigma \Longrightarrow A;\ \{Y(\vec{y}) : t \triangleq s\} \cup S;\ \sigma\{X \mapsto \lambda\vec{x}.Y(\vec{y})\}$,
  where t and s are of basic type. $\mathrm{Head}(t) \neq \mathrm{Head}(s)$ or $\mathrm{Head}(t) = Z \notin \vec{x}$.
  $\vec{y}$ is a subsequence of $\vec{x}$ consisting of the variables that appear freely in t or s.
- Rec: **Recover**
  $A;\ \{X(\vec{x}) : t_1 \triangleq t_2, Z(\vec{z}) : s_1 \triangleq s_2\} \mathbin{\dot\cup} S;\ \sigma \Longrightarrow$
  $A;\ \{X(\vec{x}) : t_1 \triangleq t_2\} \cup S;\ \sigma\{Z \mapsto \lambda\vec{z}.X(\vec{x}\pi)\}$,
  where $\pi : \{\vec{x}\} \mapsto \{\vec{z}\}$ is a bijection, extended as a substitution, such that $t_1\pi = s_1$ and $t_2\pi = s_2$.

Demonstration of $\mathfrak{P}$

- Let $t = \lambda x, y.f(x, y)$ and $s = \lambda x, y.f(y, x)$.

$$
\begin{aligned}
& \{X : \lambda x, y.f(x, y) \triangleq \lambda x, y.f(y, x)\}; \emptyset; \varepsilon \\
\Longrightarrow_{\mathsf{Abs}} \; & \{Y_1(x) : \lambda y.f(x, y) \triangleq \lambda y.f(y, x)\}; \emptyset; \{X \mapsto \lambda x.Y_1(x)\} \\
\Longrightarrow_{\mathsf{Abs}} \; & \{Y_2(x, y) : f(x, y) \triangleq f(y, x)\}; \emptyset; \{X \mapsto \lambda x, y.Y_2(x, y)\} \\
\Longrightarrow_{\mathsf{Dec}} \; & \{Y_3(x, y) : x \triangleq y, Y_4(x, y) : y \triangleq x\}; \emptyset; \\
& \{X \mapsto \lambda x, y.f(Y_3(x, y), Y_4(x, y))\} \\
\Longrightarrow_{\mathsf{Sol}} \; & \{Y_4(x, y) : y \triangleq x\}; \{Y_3(x, y) : x \triangleq y\}; \\
& \{X \mapsto \lambda x, y.f(Y_3(x, y), Y_4(x, y))\} \\
\Longrightarrow_{\mathsf{Sol}} \; & \emptyset; \{Y_3(x, y) : x \triangleq y, Y_4(x, y) : y \triangleq x\}; \\
& \{X \mapsto \lambda x, y.f(Y_3(x, y), Y_4(x, y))\} \\
\Longrightarrow_{\mathsf{Rec}} \; & \emptyset; \{Y_3(x, y) : x \triangleq y\}; \\
& \{X \mapsto \lambda x, y.f(Y_3(x, y), Y_3(y, x)), Y_4 \mapsto \lambda x, y.Y_3(y, x)\}.
\end{aligned}
$$

- The computed result $r = X\sigma$ is $\lambda x, y.f(Y_3(x, y), Y_3(y, x))$.
- It generalizes $t = r\{Y_3 \mapsto \lambda x, y.x\}$ and $s = r\{Y_3 \mapsto \lambda x, y.y\}$.

Matching Problem

- Rec: **Recover**
  $A;\ \{X(\vec{x}) : t_1 \triangleq t_2, Z(\vec{z}) : s_1 \triangleq s_2\} \,\dot\cup\, S;\ \sigma \Longrightarrow$
  $A;\ \{X(\vec{x}) : t_1 \triangleq t_2\} \cup S;\ \sigma\{Z \mapsto \lambda\vec{z}.X(\vec{x}\pi)\}$,
  where $\pi : \{\vec{x}\} \mapsto \{\vec{z}\}$ is a bijection, extended as a substitution, such that $t_1\pi = s_1$ and $t_2\pi = s_2$.

- Matching problem P, whose solution bijectively maps variables from a finite set D to a finite set R.
- The permuting matcher π is unique, if it exists.

Computing Permuted Matchers

- $\mathfrak{M}$ computes a permuting matcher π, if it exists.
- $\mathfrak{M}$ works on quintuples of the form D; R; P; ρ; π where
  - D is a set of domain variables,
  - R is a set of range variables,
  - P is a set of matching problems of the form $\{s_1 \rightrightarrows t_1, \ldots, s_m \rightrightarrows t_m\}$,
  - ρ is a substitution which keeps track of bound variable renamings,
  - π is a substitution which keeps track of the permutations.
- $\mathfrak{M}$ has two final states:
  - The failure state $\perp$.
  - The success state D; R; $\emptyset$; ρ; π.

Computing Permuted Matchers

- Create a variable renaming substitution ν to rename all the variables in D with fresh ones (domain/range separation).
- Take $D\nu$; R; $\{s_1\nu \rightrightarrows t_1, s_2\nu \rightrightarrows t_2\}$; ε; ε as the input of the algorithm and apply the rules exhaustively.
- If no rule applies to a system with $P \neq \emptyset$, then this system is transformed into $\perp$.
- If $\mathfrak{M}$ reaches the success state, then construct and return the permuting matcher $(\nu\pi)|_D$.

The rules of $\mathfrak{M}$

- Dec-M: **Decomposition**
  $D;\ R;\ \{h_1(t_1,\dots,t_m) \rightrightarrows h_2(s_1,\dots,s_m)\} \cupdot P;\ \rho;\ \pi \Longrightarrow$
  $D;\ R;\ \{t_1 \rightrightarrows s_1,\dots,t_m \rightrightarrows s_m\} \cup P;\ \rho;\ \pi,$
  where each of h_1 and h_2 is either a constant or a variable.
  $h_1\pi = h_2\rho$ and $h_1 \notin D$, or $h_1\pi = h_2\rho$ and $h_2 \notin R$.
- Abs-M: **Abstraction**
  $D;\ R;\ \{\lambda x.t \rightrightarrows \lambda y.s\} \cupdot P;\ \rho;\ \pi \Longrightarrow D;\ R;\ \{t \rightrightarrows s\} \cup P;\ \rho\{y \mapsto x\};\ \pi.$
- Per-M: **Permutation**
  $\{x\} \cupdot D;\ \{y\} \cupdot R;\ \{x(t_1,\dots,t_m) \rightrightarrows y(s_1,\dots,s_m)\} \cupdot P;\ \rho;\ \pi \Longrightarrow$
  $D;\ R;\ \{t_1 \rightrightarrows s_1,\dots,t_m \rightrightarrows s_m\} \cup P;\ \rho;\ \pi\{x \mapsto y\},$
  where x and y have the same type.

Demonstration of $\mathfrak{M}$

- Compute the permuting matcher of $\{x(y,z) \rightrightarrows x(z,y), X(y,\lambda u.u) \rightrightarrows X(z,\lambda v.v)\}$ from $\{x,y,z\}$ to $\{x,y,z\}$.

$$
\begin{aligned}
& \{x',y',z'\}; \{x,y,z\}; \{x'(y',z') \rightrightarrows x(z,y), X(y',\lambda u.u) \rightrightarrows X(z,\lambda v.v)\}; \varepsilon; \varepsilon \\
\Longrightarrow_{\text{Per-M}} \; & \{y',z'\}; \{y,z\}; \{y' \rightrightarrows z, z' \rightrightarrows y, X(y',\lambda u.u) \rightrightarrows X(z,\lambda v.v)\}; \varepsilon; \{x' \mapsto x\} \\
\Longrightarrow_{\text{Per-M}} \; & \{z'\}; \{y\}; \{z' \rightrightarrows y, X(y',\lambda u.u) \rightrightarrows X(z,\lambda v.v)\}; \varepsilon; \{x' \mapsto x, y' \mapsto z\} \\
\Longrightarrow_{\text{Per-M}} \; & \emptyset; \emptyset; \{X(y',\lambda u.u) \rightrightarrows X(z,\lambda v.v)\}; \varepsilon; \{x' \mapsto x, y' \mapsto z, z' \mapsto y\} \\
\Longrightarrow_{\text{Dec-M}} \; & \emptyset; \emptyset; \{y' \rightrightarrows z, \lambda u.u \rightrightarrows \lambda v.v\}; \varepsilon; \{x' \mapsto x, y' \mapsto z, z' \mapsto y\} \\
\Longrightarrow_{\text{Dec-M}} \; & \emptyset; \emptyset; \{\lambda u.u \rightrightarrows \lambda v.v\}; \varepsilon; \{x' \mapsto x, y' \mapsto z, z' \mapsto y\} \\
\Longrightarrow_{\text{Abs-M}} \; & \emptyset; \emptyset; \{v \rightrightarrows u\}; \{u \mapsto v\}; \{x' \mapsto x, y' \mapsto z, z' \mapsto y\} \\
\Longrightarrow_{\text{Dec-M}} \; & \emptyset; \emptyset; \emptyset; \{v \mapsto u\}; \{x' \mapsto x, y' \mapsto z, z' \mapsto y\}.
\end{aligned}
$$

- As result we obtain a substitution $\{x \mapsto x, y \mapsto z, z \mapsto y\}$.

Final remarks

- Proofs:
  - Soundness, completeness, and termination of $\mathfrak{M}$.
  - Soundness, completeness, and termination of $\mathfrak{P}$.
  - Computed result is a pattern-lgg and unique modulo α-equivalence.
- Complexity analysis:
  - $\mathfrak{M}$ has linear time and space complexity.
  - $\mathfrak{P}$ has cubic time and linear space complexity.
- Implementation:
  - http://www.risc.jku.at/projects/stout/software/hoau.php