Selected Topics from Unification Theory

Temur Kutsia

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

April 7, 2015

Overview

The Anti-Unification Problem

Early Algorithms of Anti-Unification

Applications

Nominal Anti-Unification

Outline

The Anti-Unification Problem

Early Algorithms of Anti-Unification

Applications

Nominal Anti-Unification

The Anti-Unification Problem

Given: Two terms t_1 and t_2.

Find: Their generalization, a term t such that both t_1 and t_2 are instances of t under some substitutions.

The Anti-Unification Problem

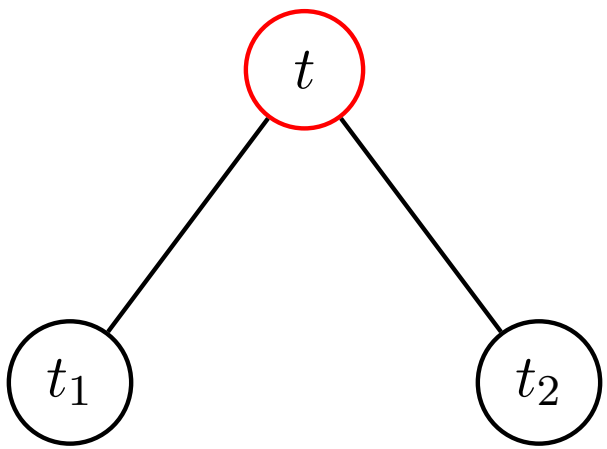
Given: Two terms t_1 and t_2.

Find: Their generalization, a term t such that both t_1 and t_2 are instances of t under some substitutions.

The Anti-Unification Problem

Given: Two terms t_1 and t_2.

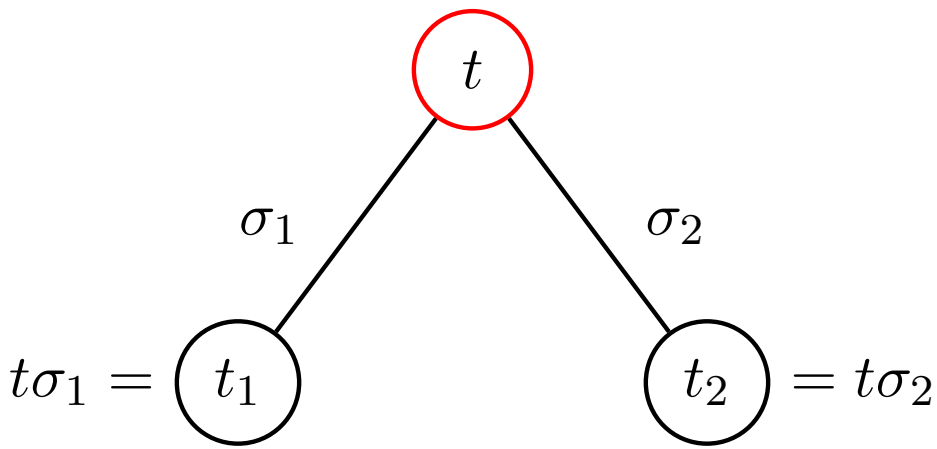
Find: Their generalization, a term t such that both t_1 and t_2 are instances of t under some substitutions.



The Anti-Unification Problem

Given: Two terms t_1 and t_2.

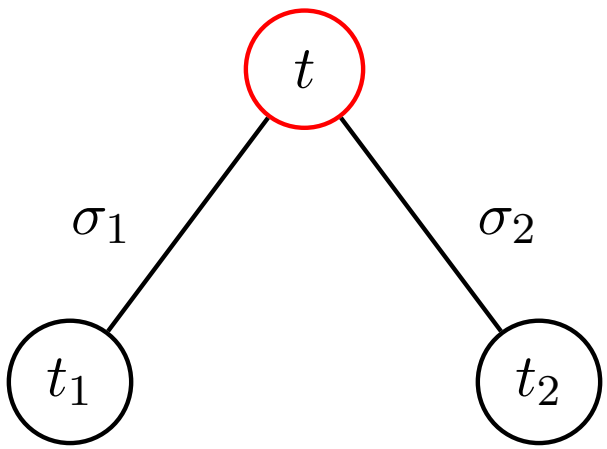
Find: Their generalization, a term t such that both t_1 and t_2 are instances of t under some substitutions.



The Anti-Unification Problem

Given: Two terms t_1 and t_2.

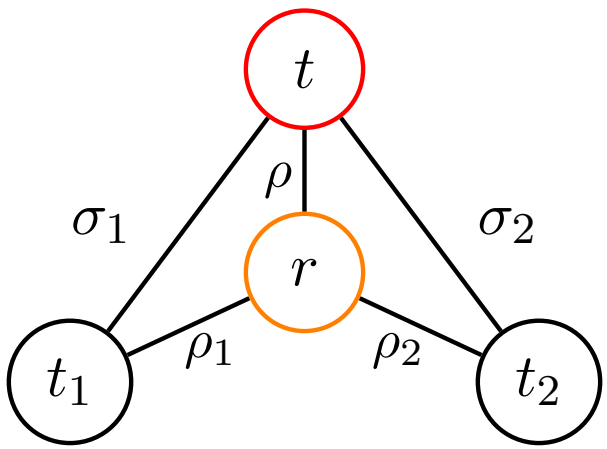
Find: Their least general generalization t.



The Anti-Unification Problem

Given: Two terms t_1 and t_2.

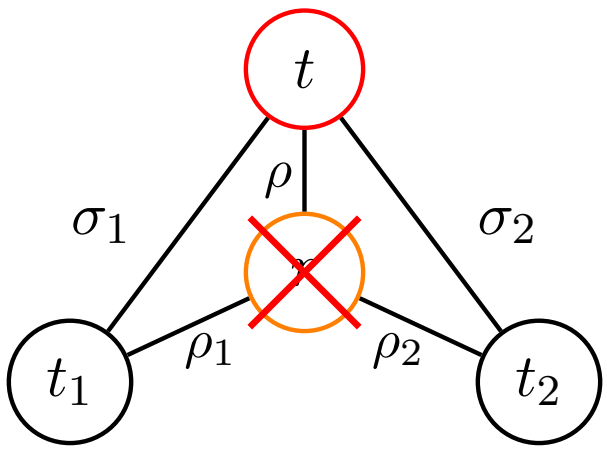
Find: Their least general generalization t.



The Anti-Unification Problem

Given: Two terms t_1 and t_2.

Find: Their least general generalization t.



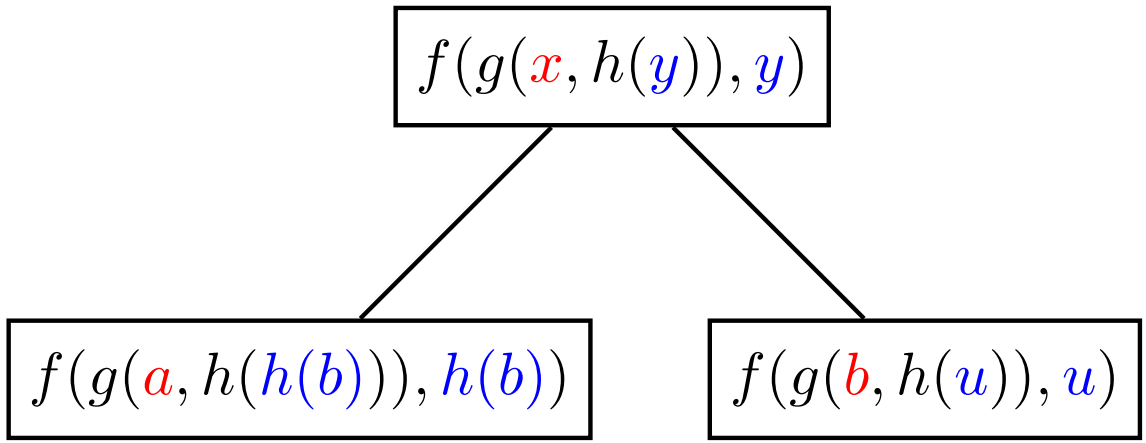
Anti-Unification: Example

$$\boxed{f(g(a, h(h(b))), h(b))} \qquad \boxed{f(g(b, h(u)), u)}$$

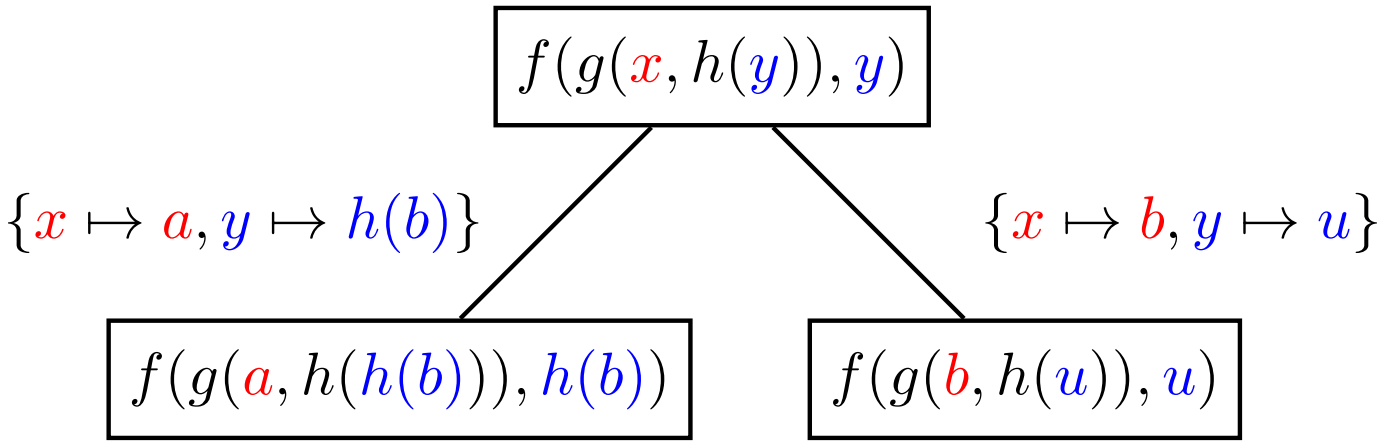
Anti-Unification: Example

$f(g(a, h(h(b))), h(b))$ $\qquad$ $f(g(b, h(u)), u)$

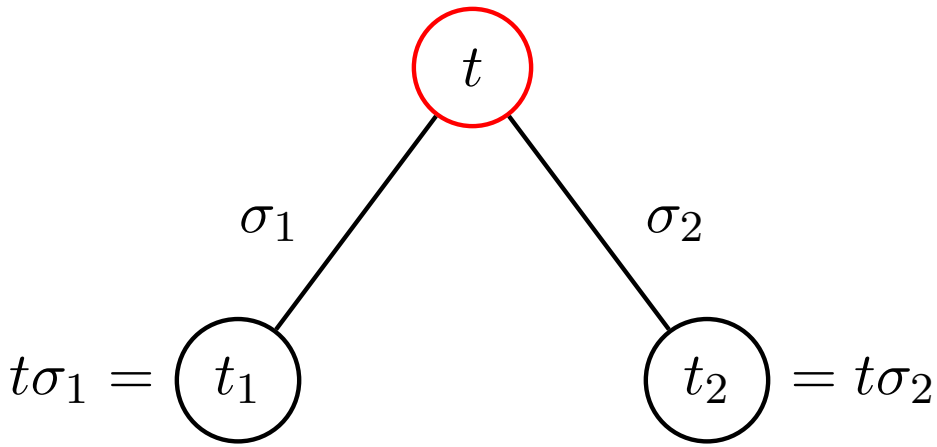
Anti-Unification: Example



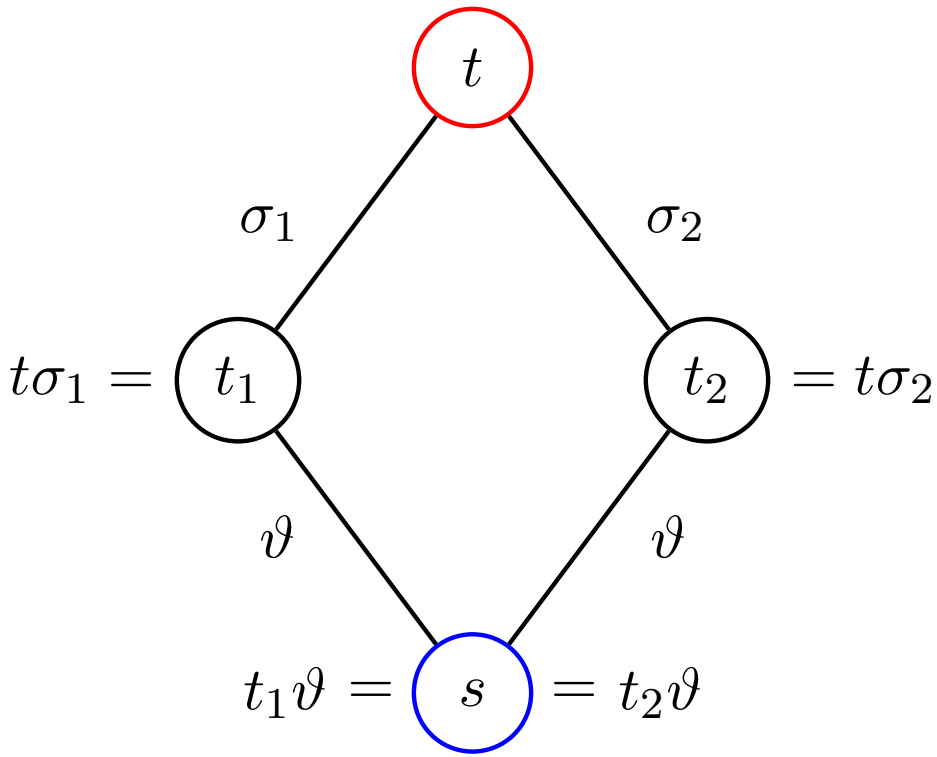
Anti-Unification: Example



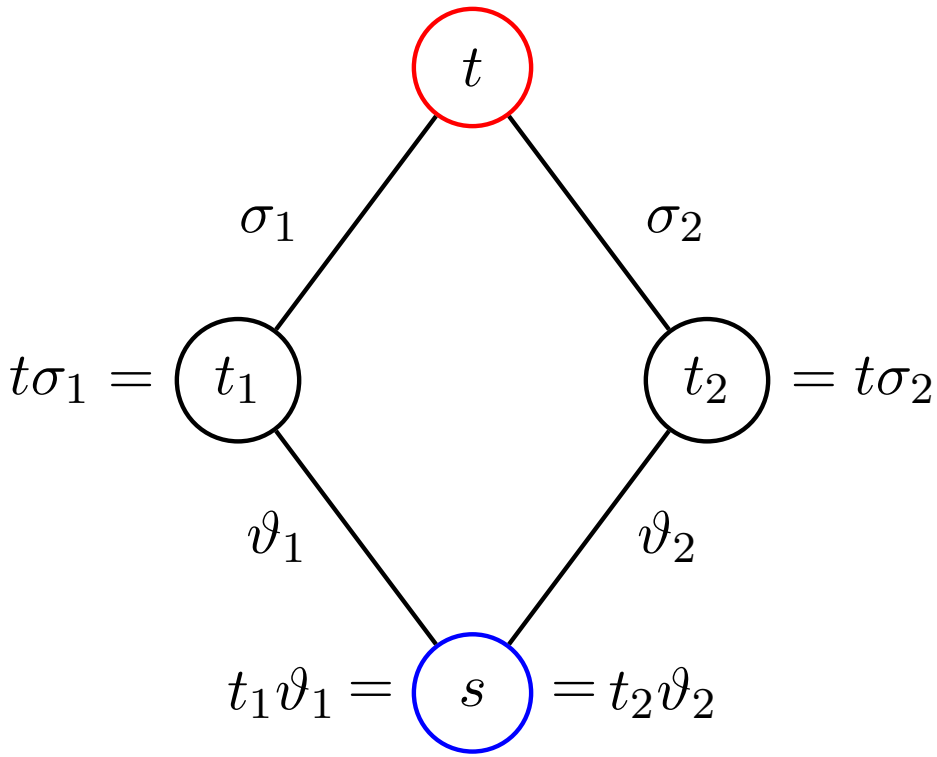
Anti-Unification and Unification



Anti-Unification and Unification



Anti-Unification and Weak Unification



Outline

The Anti-Unification Problem

Early Algorithms of Anti-Unification

Applications

Nominal Anti-Unification

Anti-Unification: Origins

- Anti-unification was introduced in two papers:

  Plotkin, G.D.: A note on inductive generalization. Mach. Intell. 5(1), 153–163 (1970)

  Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. Mach. Intell. 5(1), 135–151 (1970)

Anti-Unification: Origins

- Reynolds coined the term "anti-unification".
- Plotkin defined $C_1 \leq C_2$ for "a clause C_1 is more general than a clause C_2" iff there exists σ such that $C_1\sigma \subseteq C_2$.
- To justify this choice of notation, he writes:

  We have chosen to write $L_1 \leq L_2$ rather than $L_1 \geq L_2$ as Reynolds (1970) does, because in the case of clauses, '$\leq$' is almost the same as '$\subseteq$'...

Anti-Unification: Origins

- Huet in 1976 formulated an algorithm in terms of recursive equations:

  Let ϕ be a bijection from a pair of terms to variables.
  Define a function λ, which maps pairs of terms to terms:

  1. $\lambda(f(t_1, \ldots, t_n), f(s_1, \ldots, s_n)) = f(\lambda(t_1, s_1), \ldots, \lambda(t_n, s_n))$, for any f.
  2. $\lambda(t, s) = \phi(t, s)$ otherwise.

Outline

The Anti-Unification Problem

Early Algorithms of Anti-Unification

Applications

Nominal Anti-Unification

Anti-Unification: Applications

- The original motivation of introducing anti-unification was its application in automating induction.
- Since then, anti-unification has been used in reasoning by analogy, machine learning, inductive logic programming, software engineering, program synthesis, analysis, transformation, ...
- Algorithms suitable for those applications have been developed.

Software Code Clone Detection with Anti-Unification

- One of the interesting applications of anti-unification is in software code clone detection.
- Clones are similar pieces of software code.
- Obtained by reusing code fragments.
- Quite a typical practice.

Why Should Clones Be Detected?

In general, they are harmful:

- Additional maintenance effort.
- Additional work for enhancing and adapting.
- Inconsistencies presenting fault.

Why Should Clones Be Detected?

Extraction of similar code fragments may be required for

- program understanding
- code quality analysis
- plagiarism detection
- copyright infringement investigation
- software evolution analysis
- code compaction
- bug detection

Classification

Roy, Cordy and Koschke (2009) distinguish four types of clones:

Type 1: Identical code fragments except for variations in whitespace, layout, and comments.

Type 2: Syntactically identical fragments except for variations in identifiers, types, whitespace, layout, and comments.

Type 3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, types, whitespace, layout, and comments.

Type 4: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

1–3: Syntactic clones.

Examples of Syntactic Clone Types

```
if (a >= b) {
   c = d + b; // Comment1
   d = d + 1;}
else
   c = d - a; // Comment2
```

```
if (a >= b) {
   c = d + b; d = d + 1;
   }
else
   c = d - a
```

Type 1: Identical code fragments except for variations in whitespace, layout, and comments.

Examples of Syntactic Clone Types

```
if (a >= b) {
   c = d + b; // Comment1
   d = d + 1;}
else
   c = d - a; // Comment2
```

```
if (m >= n)
   { // Comment1'
    y = x + n;
    x = x + 5; //Comment3
   }
else
   y = x - m; //Comment2'
```

Type 2: Syntactically identical fragments except for variations in identifiers, types, whitespace, layout, and comments.

Examples of Syntactic Clone Types

```
if (a >= b) {
   c = d + b; // Comment1
   d = d + 1;}
else
   c = d - a; // Comment2
```

```
if (m >= n)
   { // Comment1'
    y = x + n;
    z = 1; // Added statement
    x = x + 5; //Comment3
   }
else
   y = x - m; //Comment2'
```

Type 3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, types, whitespace, layout, and comments.

Generic Clone Detection Process

From Roy, Cordy, and Koschke (2009):

1. Preprocessing: Remove uninteresting code, determine source and comparison units/granularities.
2. Transformation: Obtain an intermediate representation of the preprocessed code.
3. Detection: Find similar source units in the transformed code.
4. Formatting: Clone locations of the transformed code are mapped back to the original code.
5. Filtering: Clone extraction, visualization, and manual analysis to filter out false positives.

Clone Detection and Anti-Unification

1. Tree-based approach.
2. Anti-unification is used in the detection step.
3. Anti-unification based tools:
   - Breakaway (Cottrel at al, 2007)
   - CloneDigger (Bulychev et al. 2009).
   - Wrangler (Li and Thompson, 2010).
   - HaRe (Brown and Thompson, 2010).
4. Achieve high precision.
5. Detect primarily clones of type 1 and 2.

Clones and Their Refactoring

```
if (a >= b) {
   c = d + b; // Comment1
   d = d + 1;}
else
   c = d - a; // Comment2
```

```
if (m >= n)
   { // Comment1'
    y = x + n;
    x = x + 5; //Comment3
   }
else
   y = x - m; //Comment2'
```

```
proc(x1,x2,x3,x4,x5) =
     if (x1 >= x2) {
         x3 = x4 + x2;
         x4 = x4 + x5;}
     else
         x3 = x4 - x1;
```

```
proc(a,b,c,d,1)
```

```
proc(m,n,y,x,5)
```

Analogy Making and Anti-Unification

Example: Generalization of recursive program schemes from given structurally similar programs [Schmid, 2000].

Method: Restricted higher-order anti-unification.

Idea: Simple: abstract different heads of terms with a function variable if the arities coincide. Otherwise abstract with a term variable.

Example

Input:

- `fac(x) = if(eq0(x), 1, *(x, fac(p(x)))`
- `sqr(y) = if(eq0(y), 0, +(+(y,p(y)), sqr(p(y)))`

Generalization

- `X(z) = if(eq0(z), Y, Z(u, X(p(z)))`

Analogy Making and Anti-Unification

Example: Replay of program derivations [Hasker, 1995].

Given: Formal program specification together with a program fulfilling this specification, both connected by a derivation.

Assume: The specification has been slightly rewritten.

Goal: Instead of fully deriving a new program, alter the existing derivation and implementation along the changes of specification.

Method: Use higher-order anti-unification for combinator terms to detect changes and similarities between the old and the new specification, changes which can be propagated by adjusting the existing derivation.

Machine Learning and Anti-Unification

Example: An inductive learning method INDIE developed in [Armengol & Plaza, 2000].

Given: A training set of positive and negative examples, represented as feature terms.

Find: A description satisfied (subsumed) by all positive examples and no negative example.

Method: Feature term anti-unification (for positive examples).

Symbolic Computation and Anti-Unification

Example: Abstracting symbolic matrices [Almomen, Sexton, Sorge 2012]

Given: A concrete symbolic matrix.

Goal: Obtain a more compact representation employing ellipses in order to expose homogeneous regions present in the matrix.

Method: Use a version of first-order anti-unification with a special treatment of integer constants.

Symbolic Computation and Anti-Unification

$$\begin{bmatrix} a_1 & b & b & b & b & b & d_{11} & d_{12} & d_{13} & d_{14} \\ 0 & a_2 & b & b & b & b & d_{21} & d_{22} & d_{23} & d_{24} \\ 0 & 0 & a_3 & b & b & b & d_{31} & d_{32} & d_{33} & d_{34} \\ 0 & 0 & 0 & a_4 & b & b & d_{41} & d_{42} & d_{43} & d_{44} \\ 0 & 0 & 0 & 0 & a_5 & b & c_4 & c_3 & c_2 & c_1 \\ 0 & 0 & 0 & 0 & 0 & a_6 & c_4 & c_3 & c_2 & c_1 \end{bmatrix}$$

$$\begin{bmatrix} a_1 & b & \cdots & \cdots & b & d_{11} & \cdots & d_{1m} \\ 0 & \ddots & \ddots & . & \vdots & \vdots & . & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & d_{m1} & \cdots & d_{mm} \\ \vdots & . & \ddots & \ddots & b & c_m & \cdots & c_1 \\ 0 & \cdots & \cdots & 0 & a_n & c_m & \cdots & c_1 \end{bmatrix}$$

Program Analysis and Anti-Unification

Example: Invariant computation [Bulychev, Kostylev, Zakharov 2010]

Given: A program represented as a set assignment statements (with input and output points labeled by natural numbers), and a program point labeled by l.

Find: Most specific invariant at point l. An invariant at l is a (existentially closed equational) formula which holds for any run at point l.

Method: Based on anti-unification of substitutions. Compute an lgg of substitutions induced by sequences of variable assignments in runs.

Linguistics and Anti-Unification

Example: Modeling metaphoric expressions [Gust, Kühnberger, Schmid 2006]

Given: A metaphor as e.g., in "Electrons are the planets of the atom".

Find: Its formal representation.

Method: Using heuristic-driven theory projection, which is based on anti-unification.

More . . .

- Relative lgg [Plotkin 1971] taking into account background knowledge.
- Anti-unification in the Calculus of Constructions [Pfenning 1991] aiming at proof generalizations.
- Anti-unification for relaxed patterns [Feng and Muggleton 1992] for inductive logic programming.
- Generalization under implication (special forms) [Idestam-Almquist 1995, Nienhuys-Cheng & de Wolf 1996] for inductive logic programming.

More . . .

- Anti-unification in $\lambda 2$ [Lu et al. 2000] for reusing proofs about programs.
- Anti-unification for simple unranked hedges [Yamamoto et al 2001] for inductive reasoning about hedge logic programs.
- Second-order generalization [Chiba, Aoto, Toyama 2008] for automatic construction of program transformatione templates.
- Variations of restricted higher-order anti-unification [Bobere & Besold 2012] in analogy-making.
- Anti-unification for relational rules [de Souza Alcantara et al. 2012] for learning custom gestures.

More . . .

- Order-sorted feature term generalization [Aït-Kaci, Sasaki 1983]
- AC anti-unification [Pottier 1989].
- Anti-unification in commutative theories [Baader 1991].
- Variants of second order anti-unification [Hirata, Ogawa, Harao 2004].
- Word anti-unification [Biere 1993, Ciceckli & Ciceckli 2006].
- Constrained anti-unification [Page 1993].
- E-generalizations using regular tree grammars [Burghardt 2005].
- Equational and order-sorted anti-unification [Alpuente et al, 2008, 2009, 2013].

More . . .

- Anti-unification for unranked terms
  [Kutsia, Levy, Villaret 2011].
- Pattern anti-unification for simply-typed λ-calculus
  [Baumgartner et al. 2013].
- Restricted second-order unranked anti-unification
  [Baumgartner, Kutsia 2014].
- Nominal anti-unification
  [Baumgartner et al. 2014].
- Anti-Unification Library:
  `http://www.risc.jku.at/projects/stout/`

Outline

The Anti-Unification Problem

Early Algorithms of Anti-Unification

Applications

Nominal Anti-Unification

Nominal Anti-Unification: Syntax

- Nominal terms contain variables, atoms, and function symbols.
- Variables can be instantiated and atoms can be bound.
- A swapping $(a\,b)$ is a pair of atoms.
- A permutation π is a sequence of swappings.
- Nominal terms:

$$t ::= f(t_1, \ldots, t_n) \mid a \mid a.t \mid \pi \cdot X$$

Nominal Anti-Unification: Syntax

- Nominal terms contain variables, atoms, and function symbols.
- Variables can be instantiated and atoms can be bound.
- A swapping $(a\,b)$ is a pair of atoms.
- A permutation π is a sequence of swappings.
- Nominal terms:

$$t ::= f(t_1, \ldots, t_n) \mid a \mid a.t \mid \pi \cdot X$$

- Permutations apply to terms and cause swapping the names of atoms (permutation action).
- $(c\,b)(a\,b) \bullet f(c,\, b.g(a,b),\, X) = f(b,\, a.g(c,a),\, (c\,b)(a\,b) \cdot X)$.

Freshness Constraints

- Freshness constraint: $a\#X$.
- The instantiation of X cannot contain free occurrences of a.
- "a is fresh for X", "a is forbidden in X".
- Freshness context: a finite set of freshness constraints.

Substitutions

- Substitution: a mapping from variables to terms.
- Substitution application allows atom capture:
  $a.X\{X \mapsto a\} = a.a.$

Substitutions

- Substitution: a mapping from variables to terms.
- Substitution application allows atom capture: $a.X\{X \mapsto a\} = a.a$.
- A substitution σ respects a freshness context ∇ if for all X, free atoms in $X\sigma$ (except those in suspensions) are not forbidden in X by ∇.

Substitutions

- Substitution: a mapping from variables to terms.
- Substitution application allows atom capture: $a.X\{X \mapsto a\} = a.a$.
- A substitution σ respects a freshness context ∇ if for all X, free atoms in $X\sigma$ (except those in suspensions) are not forbidden in X by ∇.
- For instance, if $\nabla = \{a\#X,\ b\#X,\ b\#Y\}$, then
  - $\{X \mapsto f(c),\ Y \mapsto b.f(b)\}$ respects ∇.
  - $\{X \mapsto (a\,c)(b\,a) \cdot Y,\ Y \mapsto f(a)\}$ respects ∇.
  - $\{X \mapsto f(a)\}$ does not respect ∇.

$\approx$ and $\#$

The intended meanings of $\approx$ (the α-equivalence predicate) and $\#$ (the freshness predicate):

1. $\nabla \vdash t \approx u$ holds, if for every substitution σ such that $t\sigma$ and $u\sigma$ are ground terms and σ respects the freshness context ∇, $t\sigma$ and $u\sigma$ are α-equivalent.
2. $\nabla \vdash a\#t$ holds, if for every substitution σ such that $t\sigma$ is a ground term and σ respects the freshness context ∇, the atom a is not free in $t\sigma$.

Rules for $\approx$

$$\frac{}{\nabla \vdash a \approx a}\ (\approx\text{-atom}) \qquad \frac{\nabla \vdash t \approx t'}{\nabla \vdash a.t \approx a.t'}\ (\approx\text{-abs-1})$$

$$\frac{a \neq a' \quad \nabla \vdash t \approx (a\, a') \bullet t' \quad \nabla \vdash a \# t'}{\nabla \vdash a.t \approx a'.t'}\ (\approx\text{-abs-2})$$

$$\frac{a \# X \in \nabla \text{ for all } a \text{ such that } \pi \bullet a \neq \pi' \bullet a}{\nabla \vdash \pi \cdot X \approx \pi' \cdot X}\ (\approx\text{-susp.})$$

$$\frac{\nabla \vdash t_1 \approx t_1' \quad \cdots \quad \nabla \vdash t_n \approx t_n'}{\nabla \vdash f(t_1, \ldots t_n) \approx f(t_1', \ldots, t_n')}\ (\approx\text{-application})$$

Rules for

$$\frac{a \neq a'}{\nabla \vdash a\#a'} \text{ (\#-atom)} \qquad \frac{(\pi^{-1} \bullet a\#X) \in \nabla}{\nabla \vdash a\#\pi\cdot X} \text{ (\#-susp.)}$$

$$\frac{\nabla \vdash a\#t_1 \quad \cdots \quad \nabla \vdash a\#t_n}{\nabla \vdash a\#f(t_1, \ldots t_n)} \text{ (\#-application)}$$

$$\frac{}{\nabla \vdash a\#a.t} \text{ (\#-abst-1)} \qquad \frac{a \neq a' \quad \nabla \vdash a\#t}{\nabla \vdash a\#a'.t} \text{ (\#-abst-2)}$$

Instance of a Freshness Context

- Using the rules for $\approx$ bottom-up, one can solve the following problem:
  - Given: A set $\{a_1 \# t_1, \ldots, a_n \# t_n\}$.
  - Compute: A $\subseteq$-minimal freshness context ∇ such that $\nabla \vdash a_1 \# t_1, \ldots, \nabla \vdash a_n \# t_n$.
- Call this algorithm FC.

Instance of a Freshness Context

- Using the rules for $\approx$ bottom-up, one can solve the following problem:
  - Given: A set $\{a_1\#t_1, \ldots, a_n\#t_n\}$.
  - Compute: A $\subseteq$-minimal freshness context ∇ such that $\nabla \vdash a_1\#t_1, \ldots, \nabla \vdash a_n\#t_n$.
- Call this algorithm FC.
- An instance of a freshness context ∇ under a substitution σ:

$$\nabla\sigma := \mathsf{FC}(\{a\#X\sigma \mid a\#X \in \nabla\}).$$

Instance of a Freshness Context

- Using the rules for $\approx$ bottom-up, one can solve the following problem:
  - Given: A set $\{a_1\#t_1, \ldots, a_n\#t_n\}$.
  - Compute: A $\subseteq$-minimal freshness context ∇ such that $\nabla \vdash a_1\#t_1, \ldots, \nabla \vdash a_n\#t_n$.
- Call this algorithm FC.
- An instance of a freshness context ∇ under a substitution σ:

$$\nabla\sigma := \mathsf{FC}(\{a\#X\sigma \mid a\#X \in \nabla\}).$$

- Example:
  - $\nabla = \{a\#X,\ b\#X,\ b\#Y\}$.
  - $\sigma = \{X \mapsto (a\,c)(a\,b)\cdot Y,\ Y \mapsto f(a)\}$.
  - $\nabla\sigma = \mathsf{FC}(\{a\#(a\,c)(a\,b)\cdot Y,\ b\#(a\,c)(a\,b)\cdot Y,\ b\#f(a)\}) = \{(a\,b)(a\,c) \bullet a\#Y,\ (a\,b)(a\,c) \bullet b\#Y\} = \{c\#Y,\ a\#Y\}$.

Term-In-Context, Subsumption Order

- Term-in-context: a pair $\langle \nabla, t \rangle$ of a freshness context ∇ and a term t.
- Subsumption order defined on terms-in-context:

  $$\langle \nabla_1, t_1 \rangle \preceq \langle \nabla_2, t_2 \rangle$$

  if there exists a substitution σ such that
  - σ respects ∇_1,
  - $\nabla_1 \sigma \subseteq \nabla_2$, and
  - $\nabla_2 \vdash t_1 \sigma \approx t_2$.

Subsumption Order: Examples

- $\langle \emptyset, f(a) \rangle \preceq \langle \{a\#X\}, f(a) \rangle$ (with $\sigma = \varepsilon$).
- $\langle \{a\#X\}, f(a) \rangle \preceq \langle \emptyset, f(a) \rangle$ (with $\sigma = \{X \mapsto b\}$).
- $\langle \{a\#X\}, f(X) \rangle \not\preceq \langle \emptyset, f(X) \rangle$.
- $\langle \emptyset, f(X) \rangle \preceq \langle \{a\#Y\}, f(Y) \rangle$ with $\sigma = \{X \mapsto Y\}$.

Subsumption Order: Examples

- $\langle \{a\#X\}, f(X) \rangle \not\preceq \langle \{a\#X\}, f(a) \rangle$.
- $\langle \{b\#X\}, (a\,b)\cdot X \rangle \preceq \langle \{c\#X\}, (a\,c)\cdot X \rangle$ (with $\sigma = \{X \mapsto (a\,b)(a\,c)\cdot X\}$).
- $\langle \{c\#X\}, (a\,c)\cdot X \rangle \preceq \langle \{b\#X\}, (a\,b)\cdot X \rangle$ (with $\sigma = \{X \mapsto (a\,c)(a\,b)\cdot X\}$.

Generalization

- $\langle \Gamma, r \rangle$ is called a generalization of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ if $\langle \Gamma, r \rangle \preceq \langle \nabla_1, t \rangle$ and $\langle \Gamma, r \rangle \preceq \langle \nabla_2, s \rangle$.

Generalization

- $\langle \Gamma, r \rangle$ is called a generalization of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ if $\langle \Gamma, r \rangle \preceq \langle \nabla_1, t \rangle$ and $\langle \Gamma, r \rangle \preceq \langle \nabla_2, s \rangle$.
- $\langle \Gamma, r \rangle$ least general generalization (lgg) of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ if there is no generalization $\langle \Gamma', r' \rangle$ of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ which satisfies $\langle \Gamma, r \rangle \prec \langle \Gamma', r' \rangle$.

Generalization

- $\langle \Gamma, r \rangle$ is called a generalization of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ if $\langle \Gamma, r \rangle \preceq \langle \nabla_1, t \rangle$ and $\langle \Gamma, r \rangle \preceq \langle \nabla_2, s \rangle$.
- $\langle \Gamma, r \rangle$ least general generalization (lgg) of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ if there is no generalization $\langle \Gamma', r' \rangle$ of $\langle \nabla_1, t \rangle$ and $\langle \nabla_2, s \rangle$ which satisfies $\langle \Gamma, r \rangle \prec \langle \Gamma', r' \rangle$.
- Similar to unification, we can also talk about a minimal complete set of generalizations.

Bad News

Nominal anti-unification is of type 0.

Example

- Given terms-in-contexts: $\langle \emptyset, a_1 \rangle$ and $\langle \emptyset, a_2 \rangle$.
- In any complete set of generalizations of them there is an infinite chain

$$\langle \emptyset, X \rangle \prec \langle \{a_3 \# X\}, X \rangle \prec \langle \{a_3 \# X, a_4 \# X\}, X \rangle \prec \cdots ,$$

  where $\{a_1, a_2, a_3, \ldots\}$ is the set of all atoms of the language.
- Hence, $\langle \emptyset, a_1 \rangle$ and $\langle \emptyset, a_2 \rangle$ do not have a minimal complete set of generalizations.

Bad News

Nominal anti-unification is of type 0.

Example

- Given terms-in-contexts: $\langle \emptyset, a_1 \rangle$ and $\langle \emptyset, a_2 \rangle$.
- In any complete set of generalizations of them there is an infinite chain

$$\langle \emptyset, X \rangle \prec \langle \{a_3 \# X\}, X \rangle \prec \langle \{a_3 \# X, a_4 \# X\}, X \rangle \prec \cdots ,$$

  where $\{a_1, a_2, a_3, \ldots\}$ is the set of all atoms of the language.
- Hence, $\langle \emptyset, a_1 \rangle$ and $\langle \emptyset, a_2 \rangle$ do not have a minimal complete set of generalizations.

A way out: restrict the set of atoms permitted in generalizations.

Term-In-Context Based on an Atom Set

- A term-in-context $\langle \nabla, t \rangle$ is based on a set of atoms A, if all the atoms in t and ∇ are elements of A.
- For instance, $\langle \{b\#X\}, f(c.g(c), (a\,b) \cdot X) \rangle$ is based on $\{a, b, c\}$ and on $\{a, b, c, d\}$, but not on $\{a, b, d\}$.

Nominal Anti-Unification Problem

Given: Two nominal terms t_1 and t_2, a freshness context ∇, and a finite set of atoms A such that $\langle \nabla, t_1 \rangle$ and $\langle \nabla, t_2 \rangle$ are based on A.

Find: A term-in-context $\langle \Gamma, t \rangle$ which is also based on A, such that $\langle \Gamma, t \rangle$ is a least general generalization of $\langle \nabla, t_1 \rangle$ and $\langle \nabla, t_2 \rangle$.

Solving The Nominal Anti-Unification Problem

Notation:

- Anti-unification triple (AUT): $X : t \triangleq s$.
- X: generalization variable.

Solving The Nominal Anti-Unification Problem

The rule-based NAU algorithm works on tuples P; S; Γ; σ and two global parameters A and ∇, where

- P and S are sets of AUTs.
- If $X : t \triangleq s \in P \cup S$, then X is unique in $P \cup S$.
- P: AUTs to be solved (the problem).
- S: already solved AUTs (the store).
- A is a finite set of atoms.
- ∇ does not constrain generalization variables.
- Γ: freshness context (computed so far) which constrains generalization variables.
- σ: substitution (computed so far) mapping generalization variables to nominal terms.
- P, S, ∇, Γ: A-based.

Solving The Nominal Anti-Unification Problem

Example

To anti-unify two terms-in-context $\langle\{b\#Y\}, f(b,a)\rangle$ and $\langle\{b\#Y\}, f(Y,(a\,b)\cdot Y)\rangle$, based on the set of atoms $\{a,b\}$, we need to create the initial system P; S; Γ; σ where

- $P = \{X : f(b,a) \triangleq f(Y,(a\,b)\cdot Y)\}$,
- $S = \emptyset$,
- $\Gamma = \emptyset$,
- $\sigma = \varepsilon$,

and initialize the global parameters A and ∇ as

- $A = \{a,b\}$,
- $\nabla = \{b\#Y\}$.

Rules of The Nominal Anti-Unification Algorithm

Decomposition:

$$\{X : \mathsf{h}(t_1, \ldots, t_m) \triangleq \mathsf{h}(s_1, \ldots, s_m)\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow$$
$$\{Y_1 : t_1 \triangleq s_1, \ldots, Y_m : t_m \triangleq s_m\} \cup P;\ S;\ \Gamma;$$
$$\sigma\{X \mapsto \mathsf{h}(Y_1, \ldots, Y_m)\}$$

where h is a function symbol or an atom, $Y_1, \ldots, Y_m$ are fresh, and $m \geq 0$.

Rules of The Nominal Anti-Unification Algorithm

Abstraction:

$$\{X : a.t \triangleq b.s\} \cup P;\ S;\ \Gamma;\ \sigma \Longrightarrow$$
$$\{Y : (c\ a) \bullet t \triangleq (c\ b) \bullet s\} \cup P;\ S;\ \Gamma;\ \sigma\{X \mapsto c.Y\},$$

where Y is fresh, $c \in A$, $\nabla \vdash c\#a.t$ and $\nabla \vdash c\#b.s$.

Rules of The Nominal Anti-Unification Algorithm

Solving:

$$\{X : t \triangleq s\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow$$
$$P;\ S \cup \{X : t \triangleq s\};\ \Gamma \cup \Gamma';\ \sigma,$$

if none of the previous rules is applicable. The set Γ' is defined as $\Gamma' := \{a\#X \mid a \in A \ \wedge\ \nabla \vdash a\#t \ \wedge\ \nabla \vdash a\#s\}$.

Rules of The Nominal Anti-Unification Algorithm

Merging:

$$P;\ \{X : t_1 \triangleq s_1, Y : t_2 \triangleq s_2\} \uplus S;\ \Gamma;\ \sigma \Longrightarrow$$
$$P;\ \{X : t_1 \triangleq s_1\} \cup S;\ \Gamma\{Y \mapsto \boldsymbol{\pi}\cdot X\};\ \sigma\{Y \mapsto \boldsymbol{\pi}\cdot X\},$$

where $\boldsymbol{\pi}$ is an $Atoms(t_1, s_1, t_2, s_2)$-based permutation such that $\nabla \vdash \boldsymbol{\pi} \bullet t_1 \approx t_2$, and $\nabla \vdash \boldsymbol{\pi} \bullet s_1 \approx s_2$.

NAU Algorithm: Example

Let $t = f(a, b)$, $s = f(b, c)$, $\nabla = \emptyset$, and $A = \{a, b, c, d\}$.

The NAU algorithm run:

$$\{X : f(a,b) \stackrel{?}{=} f(b,c)\}; \emptyset; \emptyset; \varepsilon \Longrightarrow_{\mathsf{Dec}}$$

$$\{Y : a \stackrel{?}{=} b,\ Z : b \stackrel{?}{=} c\}; \emptyset; \emptyset; \{X \mapsto f(Y,Z)\} \Longrightarrow_{\mathsf{Sol}}$$

$$\{Z : b \stackrel{?}{=} c\}; \{Y : a \stackrel{?}{=} b\}; \{c\#Y, d\#Y\}; \{X \mapsto f(Y,Z)\} \Longrightarrow_{\mathsf{Sol}}$$

$$\emptyset; \{Y : a \stackrel{?}{=} b,\ Z : b \stackrel{?}{=} c\}; \{c\#Y, d\#Y, a\#Z, d\#Z\};$$
$$\{X \mapsto f(Y,Z)\} \Longrightarrow_{\mathsf{Mer}}$$

$$\emptyset; \{Y : a \stackrel{?}{=} b\}; \{c\#Y, d\#Y\}; \{X \mapsto f(Y, (a\,b)(b\,c)\cdot Y)\}$$

NAU Algorithm: Example

Let $t = f(b, a)$, $s = f(Y, (a\,b)\cdot Y)$, $\nabla = \{b\#Y\}$, and $A = \{a, b\}$.

The NAU algorithm computes $p = \langle \emptyset, f(Z, (a\,b)\cdot Z)\rangle$.

It generalizes the input pairs:

$p\{Z \mapsto b\} = \langle \emptyset, f(b, a)\rangle \preceq \langle \nabla, t\rangle$,

$p\{Z \mapsto Y\} = \langle \emptyset, f(Y, (a\,b)\cdot Y)\rangle \preceq \langle \nabla, s\rangle$.

NAU Algorithm: Example

Let $t = f(a.b, X)$, $s = f(b.a, Y)$, $\nabla = \{c\#X\}$, $A = \{a, b, c, d\}$.

The NAU algorithm computes $p = \langle \{c\#Z_1, d\#Z_1\}, f(c.Z_1, Z_2) \rangle$.

It generalizes the input pairs:

$p\{Z_1 \mapsto b, Z_2 \mapsto X\} = \langle \emptyset, f(c.b, X) \rangle \preceq \langle \nabla, t \rangle$,

$p\{Z_1 \mapsto a, Z_2 \mapsto Y\} = \langle \emptyset, f(c.a, Y) \rangle \preceq \langle \nabla, s \rangle$.

Properties of the NAU Algorithm

Informally:

- Soundness: The result computed by the algorithm is indeed an A-based gener- alization of the input terms-in-context.
- Completeness: For any A-based generalization of the input terms-in-context, the algorithm can compute one which is at most as general that the given generalization.
- Uniqueness: All generalizations computed by the algorithm via different AUT choices are the same modulo variable renaming and α-equivalence.

The Result Depends on the Choice of the Set of Atoms

Example

Let $A_1 = \{a, b\}$ and $A_2 = \{a, b, c\}$.

- A_1-based lgg of $\langle \emptyset, a.b \rangle$ and $\langle \emptyset, b.a \rangle$: $\langle \emptyset, X \rangle$.
- A_2-based lgg of $\langle \emptyset, a.b \rangle$ and $\langle \emptyset, b.a \rangle$: $\langle \{c \# X\}, c.X \rangle$.

A Pragmatic Question

Given t, s and ∇, how to choose a set of atoms A so that

(a) t, s, ∇ are A-based and

(b) in the A-based lgg $\langle \Gamma, r \rangle$ of $\langle \nabla, t \rangle$ and $\langle \nabla, s \rangle$, the term r generalizes s and t in the "best way", maximally preserving similarities and uniformly abstracting differences between s and t.

A Pragmatic Question

Given t, s and ∇, how to choose a set of atoms A so that

(a) t, s, ∇ are A-based and

(b) in the A-based lgg $\langle \Gamma, r \rangle$ of $\langle \nabla, t \rangle$ and $\langle \nabla, s \rangle$, the term r generalizes s and t in the "best way", maximally preserving similarities and uniformly abstracting differences between s and t.

Answer: Besides all the atoms occurring in t, s, or ∇, A should contain at least m more atoms, where $m = \min\{\|t\|_{\mathsf{Abs}}, \|s\|_{\mathsf{Abs}}\}$.

$\|t\|_{\mathsf{Abs}}$ stands for the number of abstraction occurrences in t.

Deciding Equivariance

Recall **Merging:**

$$P;\ \{X : t_1 \triangleq s_1, Y : t_2 \triangleq s_2\} \uplus S;\ \Gamma;\ \sigma \Longrightarrow$$
$$P;\ \{X : t_1 \triangleq s_1\} \uplus S;\ \Gamma\{Y \mapsto \pi \cdot X\};\ \sigma\{Y \mapsto \pi \cdot X\},$$

where π is an $Atoms(t_1, s_1, t_2, s_2)$-based permutation such that $\nabla \vdash \pi \bullet t_1 \approx t_2$, and $\nabla \vdash \pi \bullet s_1 \approx s_2$.

The condition requires an algorithm that constructively decides whether t and s are equivariant with respect to ∇:

Given: t, s, and ∇.

Find: An $Atoms(t, s)$-based permutation π such that $\nabla \vdash \pi \bullet t \approx s$.

Equvariance Decision Algorithm

A rule-based algorithm $\mathfrak{E}$, working on tuples E; ∇; A; π.

- E: A set of equivariance equations of the form $t \approx s$.
- ∇: A freshness context.
- A: A finite set of atoms which are available for computing π.
- π holds the permutation to be returned in case of success.

Equvariance Decision Algorithm

- First phase: simplification. Function applications, abstractions and suspensions are decomposed as long as possible.
- Second phase: permutation computation.

Equvariance Decision Algorithm: First Phase Rules

Decomposition in Equivariance

$$\{f(t_1, \ldots, t_m) \approx f(s_1, \ldots, s_m)\} \uplus E;\ \nabla;\ A;\ Id \Longrightarrow$$
$$\{t_1 \approx s_1, \ldots, t_m \approx s_m\} \cup E;\ \nabla;\ A;\ Id.$$

Alpha Equivalence

$$\{a.t \approx b.s\} \uplus E;\ \nabla;\ A;\ Id \Longrightarrow$$
$$\{(a\ \acute{c}) \bullet t \approx (b\ \acute{c}) \bullet s\} \cup E;\ \nabla;\ A;\ Id,$$

where $\acute{c}$ is a fresh atom of the same sort as a and b.

Suspension

$$\{\pi_1 \cdot X \approx \pi_2 \cdot X\} \uplus E;\ \nabla;\ A;\ Id \Longrightarrow$$
$$\{\pi_1 \bullet a \approx \pi_2 \bullet a \mid a \in A \wedge a\#X \notin \nabla\} \cup E;\ \nabla;\ A;\ Id.$$

Equvariance Decision Algorithm: Second Phase Rules

Remove

$$\{a \approx b\} \uplus E;\ \nabla;\ A;\ \pi \Longrightarrow E;\ \nabla;\ A \setminus \{b\};\ \pi,$$

if $\pi \bullet a = b$.

Solve in Equivariance

$$\{a \approx b\} \uplus E;\ \nabla;\ A;\ \pi \Longrightarrow E;\ \nabla;\ A \setminus \{b\};\ (\pi \bullet a\ b)\pi$$

if $\pi \bullet a \in A$, $b \in A$, and $\pi \bullet a \neq b$.

Equivariance Decision Algorithm: Example

Equivariance problem $E = \{a \approx a,\ a.(a\,b)(c\,d)\cdot X \approx b.X\}$ and $\nabla = \{a\#X\}$:

$$\{a \approx a,\ a.(a\,b)(c\,d)\cdot X \approx b.X\}; \{a\#X\};$$
$$\{a, b, c, d\}; Id \Longrightarrow \text{Alpa Equivalence}$$
$$\{a \approx a,\ (a\,é)(a\,b)(c\,d)\cdot X \approx (b\,é)\cdot X\}; \{a\#X\};$$
$$\{a, b, c, d\}; Id \Longrightarrow \text{Suspension}$$
$$\{a \approx a,\ é \approx é,\ d \approx c,\ c \approx d\}; \{a\#X\};$$
$$\{a, b, c, d\}; Id \Longrightarrow \text{Remove}$$
$$\{é \approx é,\ d \approx c,\ c \approx d\}; \{a\#X\};\ \{b, c, d\};\ Id \Longrightarrow \text{Remove}$$
$$\{d \approx c,\ c \approx d\}; \{a\#X\};\ \{b, c, d\};\ Id \Longrightarrow \text{Solve in Equivariance}$$
$$\{c \approx d\};\ \{a\#X\};\ \{b, d\};\ (d\,c) \Longrightarrow \text{Remove}$$
$$\emptyset;\ \{a\#X\};\ \{b\};\ (d\,c).$$

Properties of the Equivariance Algorithm

Informally:

- Soundness: The computed permutation shows that the input terms are equivariant.
- Completeness: For any permutation that shows that the input terms are equivariant, the algorithm computes one that is equal to the given permutation (on the set of free atoms of the corresponding term).

Complexity

- Given a set of equivariance equations E and a freshness context ∇, the equivariance algorithm has $O(n^2 + m)$ time complexity, where m be the size of ∇, and n is the size of E.

Complexity

- Given a set of equivariance equations E and a freshness context ∇, the equivariance algorithm has $O(n^2 + m)$ time complexity, where m be the size of ∇, and n is the size of E.
- The nominal anti-unification algorithm has $O(n^5)$ time complexity and $O(n^4)$ space complexity, where n is the input size.

Implementation

- `http://www.risc.jku.at/projects/stout/`
- Implemented in Java, open source.
- Both algorithms are accessible online and can be downloaded.
- The equivariance algorithm is also available separately.