Anti-Unification on Terms With Different Types

Gabriela de Souza Ferreira
Supervisor: Prof. Dr. Mauricio Ayala Ricón
Co-supervisor: Prof. Dr. Temur Kutsia

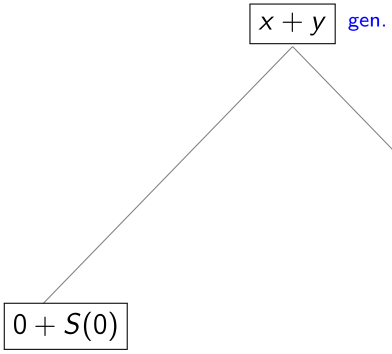
Graduate Program in Mathematics
Universidade de Brasília

This talk

- examples where the anti-unification problems are interesting,
- preliminary design of anti-unification rules,
- limitations of these rules,
- possible future work.

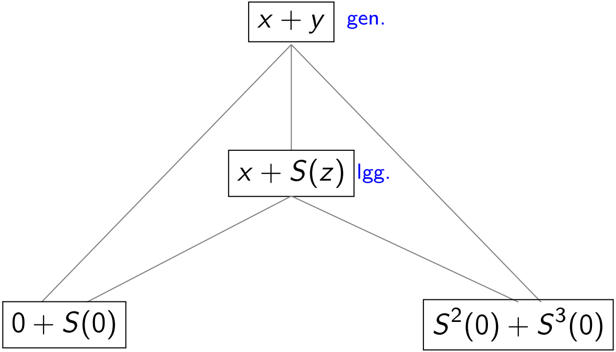
An intuitive example

$\mathbb{N} = \{0, S(0), S^2(0), \cdots\}$.
Compare $0 + S(0)$ and $S^2(0) + S^3(0)$



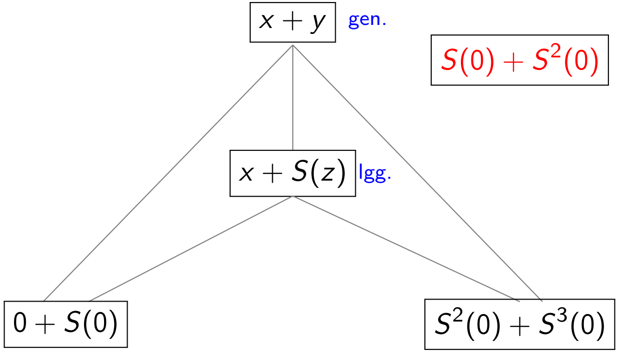
An intuitive example

$\mathbb{N} = \{0, S(0), S^2(0), \cdots\}$.
Compare $0 + S(0)$ and $S^2(0) + S^3(0)$



An intuitive example

$\mathbb{N} = \{0, S(0), S^2(0), \cdots\}$.
Compare $0 + S(0)$ and $S^2(0) + S^3(0)$



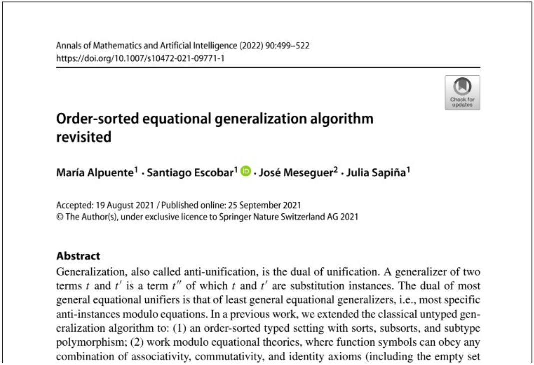
Anti-Unification

Definition

Given two terms s and t,
find the set of least general generalizations (lgg) of s and t.

Avoid some problems

$s : \tau$ $\times$ $t : \gamma$ No common generalization


Annals of Mathematics and Artificial Intelligence (2022) 90:499–522
https://doi.org/10.1007/s10472-021-09771-1

Check for updates

Order-sorted equational generalization algorithm revisited

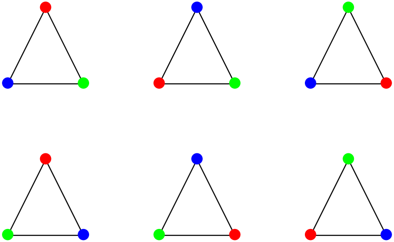
María Alpuente[1] · Santiago Escobar[1] · José Meseguer[2] · Julia Sapiña[1]

Accepted: 19 August 2021 / Published online: 25 September 2021

Abstract
Generalization, also called anti-unification, is the dual of unification. A generalizer of two terms t and t' is a term t'' of which t and t' are substitution instances. The dual of most general equational unifiers is that of least general equational generalizers, i.e., most specific anti-instances modulo equations. In a previous work, we extended the classical untyped generalization algorithm to: (1) an order-sorted typed setting with sorts, subsorts, and subtype polymorphism; (2) work modulo equational theories, where function symbols can obey any combination of associativity, commutativity, and identity axioms (including the empty set

Symmetric group S_3

Representation 1: $S_3 = \{\mathbb{1}, (12), (23), (13), (123), (132)\}$
Representation 2:



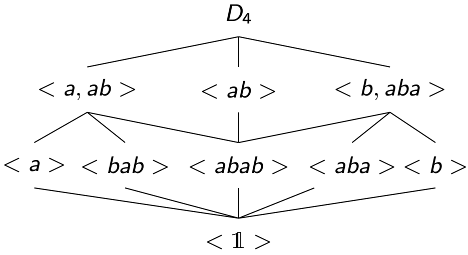
Different representations using different types!

Comparing groups

Find the maximal subgroups that are contained in both groups.

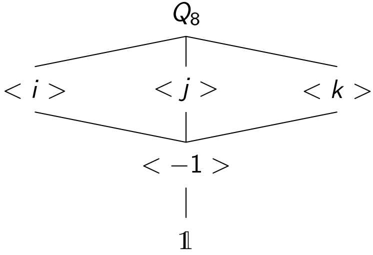
D_4 : **Dihedral group of order 8.**

$$a = (13), \ b = (14)(23)$$



Q_8 : **Quaternion group.**

$$Q_8 = < \mathbb{1}, i, j, k >$$

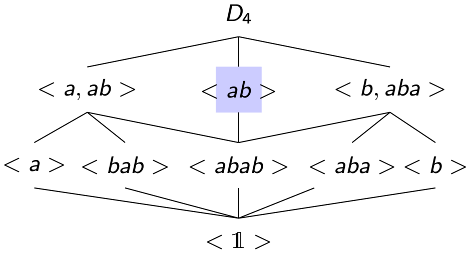


Comparing groups

Find the maximal subgroups that are contained in both groups.

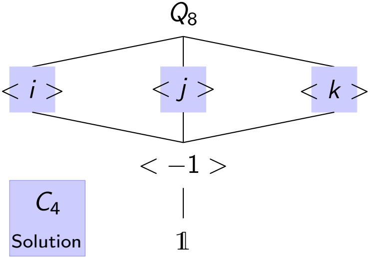
D_4 : **Dihedral group of order 8.**

$$a = (13),\ b = (14)(23)$$



Q_8 : **Quaternion group.**

$$Q_8 =< <1>, i, j, k >$$



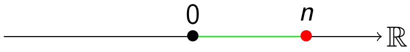
Let's talk about types

Function 1

Function modulo over real numbers

$$r(n) = \sqrt{n^2}$$

$$r : \mathbb{R} \to \mathbb{R}_+ \cup \{0\}$$

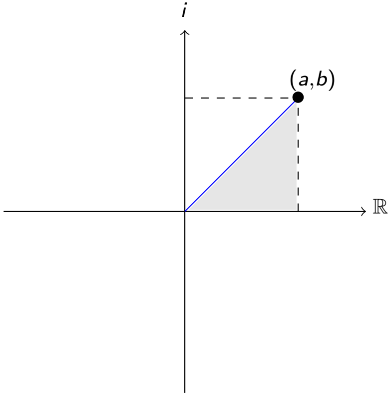


Function 2

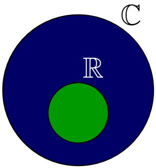
Function modulo over complex numbers

$$c(n) = c(a + bi) = \sqrt{a^2 + b^2}$$

$$c : \mathbb{C} \rightarrow \mathbb{R}_+ \cup \{0\}$$



Comparing



- Every real number n it's also a complex number $n = n + 0i$,
- if $r(n)$ is well defined, then $c(n)$ it's also well defined
- **Intuition:** c and r should have some structure in common!
- **Problem:** usual anti-unification problem solutions says that they have nothing in common because on they different types!

Comparing the structure:

$$g(a + xi) = \sqrt{a^2 + x^2}$$

$\{x \mapsto 0\}$ $\qquad$ $\{x \mapsto b\}$

$$r(a) = \sqrt{a^2} \qquad\qquad c(a + bi) = \sqrt{a^2 + b^2}$$

How to compare the types?

$$g(a + xi) = \sqrt{a^2 + x^2} : \mathbb{x} \to \mathbb{R}_+ \cup \{0\}, \text{ where } \mathbb{x} \text{ is a type variable}$$
$$r(a) = \sqrt{a^2} : \mathbb{R} \to \mathbb{R}_+ \cup \{0\}$$
$$c(a + bi) = \sqrt{a^2 + b^2} : \mathbb{C} \to \mathbb{R}_+ \cup \{0\}$$

Polymorphism

"The term polymorphism refers to a range of language mechanisms that allow a single part of a program to be used with different types in different contexts."

B. Pierce

Syntax

$$\lambda\text{-term t::= } x \mid c \mid \lambda x.t \mid t_1 t_2$$

- function symbols: c, f
- bind variables : $x, y, z,$
- free variables: $X, Y, Z,$
- substitutions: ϕ, ρ, θ,
- types: τ, π, ρ,
- type variables: $\mathbb{x}$, $\mathbb{y}$, $\mathbb{z}$.
- η-long, β-normal form,

Anti-Unification Problem:

Definition (Higher Order Pattern)

- η-long β-normal form,
- all free variables occurrences are applied to lists of pairwise distinct bound variables.

Examples

- $\lambda x, y.f(X(x), Z(y))$ and $\lambda x, y.f(X(x, y), Z(x, y))$ are pattern,
- and $\lambda x, y.f(X(x, x), Z)$ and $\lambda x, y.f(X(Y), Z)$ are not.

Anti-Unification

Definition (Anti-Unification)

Given: terms in η-long β-normal form, such that $s : \tau$ and $t : \pi$
To find: the set of least general pattern generalizations of s and t.

Inference Procedure

$$\underbrace{P; S; \sigma}_{\textit{terms}}; \underbrace{T_P; T_S; T_B}_{\textit{types}}$$

Input: $s : \tau \triangleq t : \pi$, where s and t are both in η-long β-normal form.

Initial Configuration	$\cdots \overset{}{\underset{\textbf{inf. rules}}{\Longrightarrow}}$	Final Configuration
$\{X : s^\tau \triangleq t^\pi\}; \emptyset; id; \emptyset; id; \emptyset$		$\emptyset; S; B; \emptyset; T_S; T_B$

Output: $X\sigma : \mathbb{x}\, T_B$

Preliminary Rules

Abstraction: ABS

$$\{X(\vec{x}) : \lambda x^{\tau_1}.s^{\tau_2} \triangleq \lambda y^{\pi_1}.t^{\pi_2}\} \uplus T; S; \sigma; \emptyset; T_S; T_B$$
$$\Longrightarrow \{Z(\vec{x}, z) : s^{\tau_2}[x \mapsto z] \triangleq t^{\tau_2}[y \mapsto z]\} \cup T; S; \sigma\{X \mapsto \lambda\vec{x}, z^{\mathbb{x}}.Z(\vec{x}, z)\}; \{\mathbb{x} : \tau_1 \triangleq \pi_1\}; T_B$$

Where Z, z are fresh variable, $\mathbb{x}$ is a fresh type variable.

Decomposition: DEC

$$\{X(\vec{x}) : f(s_1^{\tau_1}, \cdot, s_n^{\tau_n})^{\tau} \triangleq f(t_1^{\tau_1}, \cdots, t_n^{\tau_n})^{\pi}\} \uplus T, S; \sigma; \emptyset; T_S; T_B$$
$$\Longrightarrow \{X_1(\vec{x}) : s_1^{\tau_1} \triangleq t_1^{\tau_1}, \cdots X_n(\vec{x}) : s_n^{\tau_n} \triangleq t_n^{\tau_n}\} \cup T; S; \sigma\{X \mapsto \lambda\vec{x}.f^{\tau_1 \to \cdots \to \tau_n \to \tau}(X_1, \cdots, X_n)(\vec{x})\}$$
$$\emptyset; T_S; T_B$$

where f is a constant or $f \in \vec{x}$ such that $f : \tau_1 \to \cdots \to \tau_n \to \tau$, and $X_1, \cdots, X_n$ are fresh variables.

Preliminary Rules

Solve: SOL

$$\{X(\vec{x}) : s^{\tau} \triangleq t^{\pi}\} \uplus P; S; \sigma; \emptyset; T_S; T_B$$
$$\Longrightarrow P; \{Y(\vec{y}) : s^{\tau} \triangleq t^{\pi}\} \cup S; \sigma\{X \mapsto \lambda\vec{x}.Y^{\mathbb{x}}(y)\}; \{\mathbb{x} : \tau \triangleq \pi\}; T_S; T_B$$

where and Y is a fresh variable and

- τ is a basic type and π is not (or vise versa),or
- τ and π are basic type: $\mathtt{head}(s) \neq \mathtt{head}(t)$ or $\mathtt{head}(s) = \mathtt{head}(t) = Z \notin \vec{x}$, the sequence $\vec{y}$ is a subsequence of $\vec{x}$ consisting of the variables that appear freely in t or s.

Merge: MER

$$P; \{X(\vec{x}) : s_1^{\tau} \triangleq s_2^{\pi}, Y(\vec{y}) : t_1^{\tau} \triangleq t_2^{\pi}\} \uplus S; \sigma; T_P; T_S; T_B$$
$$\Longrightarrow P; \{X(\vec{x}) : s_1^{\tau} \triangleq s_2^{\pi}\} \cup S; \sigma\{Y \mapsto \lambda\vec{y}.X(\vec{x}\theta)\}; T_P; T_S; T_B$$

where $\theta : \{\vec{x}\} \to \{\vec{y}\}$ is a bijection, extended as a substitution, with $s_1\theta = t_1$ and $s_2\theta = t_2$.

Preliminary Rules

Type Decomposition 1: T-DEC-1

$$T; S; \sigma; \{\mathbb{x} : \tau_1 \to \tau_2 \triangleq \pi_1 \to \pi_2\} \uplus T_P; T_S; T_B$$
$$\Longrightarrow P; S; \sigma; \{\mathbb{x}_1 : \tau_1 \to \pi_1, \mathbb{x}_2 : \tau_2 \triangleq \pi_2\} \cup T_P; T_S; T_B\{\mathbb{x} \mapsto \mathbb{x}_1 \to \mathbb{x}_2\}$$

Type Decomposition 2: T-DEC-2

$$P; S; \sigma; \{\mathbb{x} : \tau \triangleq \tau\} \uplus T_P; T_S \uplus T_P; T_B$$
$$\Longrightarrow P; S; \sigma; T_P; T_S; T_B\{\mathbb{x} \mapsto \tau\}$$

where τ is basic.

Preliminary Rules

Type Solve T-SOL

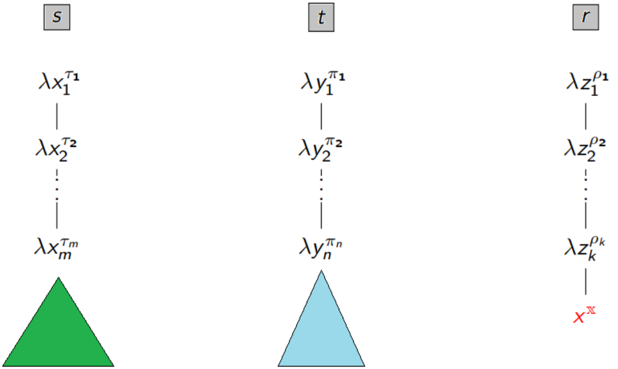
$$P; S; \sigma; \{\mathbb{x} : \tau \triangleq \pi\} \uplus T_P; T_S; T_B$$
$$\Longrightarrow P; S; \sigma; T_P; T_S \cup \{\mathbb{x} : \tau \triangleq \pi\}; T_B$$

where $\tau \neq \pi$ and "τ or π is basic".

Type Merge T-MER

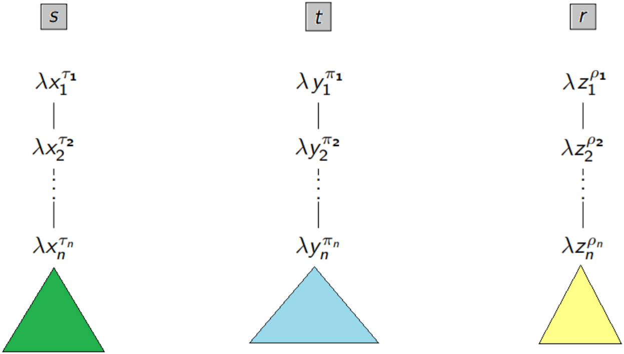
$$P; S; \sigma; \emptyset; \{\mathbb{x} : \tau \triangleq \pi, \mathbb{y} : \tau \triangleq \pi\} \uplus S; T_B$$
$$\Longrightarrow P; S; \sigma; \emptyset; \{\mathbb{x} : \tau \triangleq \pi\} \cup S; T_B\{\mathbb{y} \mapsto \mathbb{x}\}$$

What does the procedure calculates?



Where $k = \mathtt{min}(m, n)$

What does the procedure calculates?



Identify the type of the problem

- **Unitary type:** Singleton `mcsg`.
- **Finitary type:** Any anti-unification problem in the theory has an `mcsg` of finite cardinality, for at least one problem having greater than 1.
- **Infinitary type:** For any anti-unification problem in the theory there exists an `mcsg`, and for at least one problem this set is infinite.
- **Nullary type** (or type zero): There exists an anti-unification problem in the theory which does not have an `mcsg`, i.e., every complete set of generalizations for this problem contain two distinct element such that one is more general than the other.

Verify desirable properties

- Soundeness,
- Completeness,
- Complexity.

References

Annals of Mathematics and Artificial Intelligence (2022) 90:499–522
https://doi.org/10.1007/s10472-021-09771-1

Order-sorted equational generalization algorithm revisited

María Alpuente[1] · Santiago Escobar[1] · José Meseguer[2] · Julia Sapiña[1]

Accepted: 19 August 2021 / Published online: 25 September 2021

Abstract
Generalization, also called anti-unification, is the dual of unification. A generalizer of two terms t and t' is a term t'' of which t and t' are substitution instances. The dual of most general equational unifiers is that of least general equational generalizers, i.e., most specific anti-instances modulo equations. In a previous work, we extended the classical untyped generalization algorithm to: (1) an order-sorted typed setting with sorts, subsorts, and subtype polymorphism; (2) work modulo equational theories, where function symbols can obey any combination of associativity, commutativity, and identity axioms (including the empty set of such axioms); and (3) the combination of both, which results in a modular, order-sorted equational generalization algorithm. However, Cerna and Kutsia showed that our algorithm is generally incomplete for the case of identity axioms and a counterexample was given. Furthermore, they proved that, in theories with two identity elements or more, generalization with identity axioms is generally nullary, yet it is finitary for both the linear and one-unital fragments, i.e., either solutions with repeated variables are disregarded or the considered theories are restricted to having just one function symbol with an identity or unit element. In this work, we show how we can easily extend our original inference system to cope with the non-linear fragment and identify a more general class than one–unit theories where generalization with identity axioms is finitary.

Keywords Least general generalization · Rule–based languages · Equational reasoning · Order-Sorted · Associativity · Commutativity · Identity

Mathematics Subject Classification (2010) 68N17 · 68N18 · 68Q42 · 68Q60 · 68T20 · 68W30

1 Introduction

Computing generalizations is relevant in a wide spectrum of automated reasoning areas

J Autom Reasoning (2017) 58:293–310
DOI 10.1007/s10817-016-9383-3

CrossMark

Higher-Order Pattern Anti-Unification in Linear Time

Alexander Baumgartner[1] · Temur Kutsia[1] ·
Jordi Levy[2] · Mateu Villaret[3]

Received: 22 December 2015 / Accepted: 6 July 2016 / Published online: 27 July 2016

Abstract We present a rule-based Huet's style anti-unification algorithm for simply typed lambda-terms, which computes a least general higher-order pattern generalization. For a pair of arbitrary terms of the same type, such a generalization always exists and is unique modulo α-equivalence and variable renaming. With a minor modification, the algorithm works for untyped lambda-terms as well. The time complexity of both algorithms is linear.

Keywords Generalizations of lambda terms · Anti-unification · Higher-order patterns

This research has been partially supported by the Austrian Science Fund (FWF) project SToUT (P 24087-N18), the Upper Austrian Government strategic program "Innovatives OÖ 2010plus", the MINECO projects RASO (TIN2015-71799-C2-1-P) and HeLo (TIN2012-33042), the MINECO/FEDER UE project LoCoS (TIN2015-66293-R) and the UdG project MPCUdG2016/055.

✉ Temur Kutsia
kutsia@risc.jku.at

Alexander Baumgartner
abaumgar@risc.jku.at

Jordi Levy
levy@iiia.csic.es

References

A Generic Framework for Higher-Order Generalizations

David M. Cerna
FMV and RISC, Johannes Kepler University Linz, Austria
david.cerna@risc.jku.at

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

Abstract

We consider a generic framework for anti-unification of simply typed lambda terms. It helps to compute generalizations which contain maximally common top part of the input expressions, without nesting generalization variables. The rules of the corresponding anti-unification algorithm are formulated, and their soundness and termination are proved. The algorithm depends on a parameter which decides how to choose terms under generalization variables. Changing the particular values of the parameter, we obtained four new unitary variants of higher-order anti-unification and also showed how the already known pattern generalization fits into the schema.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Higher order logic; Theory of computation → Type theory

Keywords and phrases anti-unification, typed lambda calculus, least general generalization

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.10

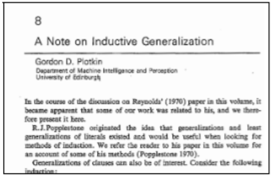
Funding Supported by the Austrian Science Fund (FWF) under the project P 28789-N32.

Acknowledgements We thank Tomer Libal for useful discussions on the early version of the paper.

1 Introduction

A term r is generalization of a term t, if t can be obtained from r by a variable substitution. The problem of finding common generalizations of two or more terms has been investigated quite intensively. The main idea is to compute least general generalizations (lggs) which maximally keep the similarities between the input terms and uniformly abstract over differences in them by new variables. For instance, if the input terms are $t = f(a, a)$ and $s = f(b, b)$, we are interested in their lgg $f(x, x)$. It gives more precise information about the nature of t and s than their other generalizations such as, e.g., $f(x, y)$ or just x. Namely, it shows that t and s not only have the same head f, but also each of them has its both arguments equal.

The technique of computing generalizations is called anti-unification. It was introduced in 1970s [17, 18] and saw a renewed interest in recent years (see, e.g., [3, 2, 11, 6, 1]), mostly motivated by various applications (see, e.g., [5, 13, 19, 20]).



8

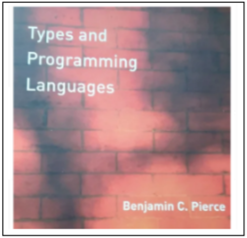
A Note on Inductive Generalization

Gordon D. Plotkin
Department of Machine Intelligence and Perception
University of Edinburgh

In the course of the discussion on Reynolds' (1970) paper in this volume, it became apparent that some of our work was related to his, and we therefore present it here.

R.J.Popplestone originated the idea that generalizations and least generalizations of literals existed and would be useful when looking for methods of induction. We refer the reader to his paper in this volume for an account of some of his methods (Popplestone 1970).

Generalizations of clauses can also be of interest. Consider the following induction:



Thank you!