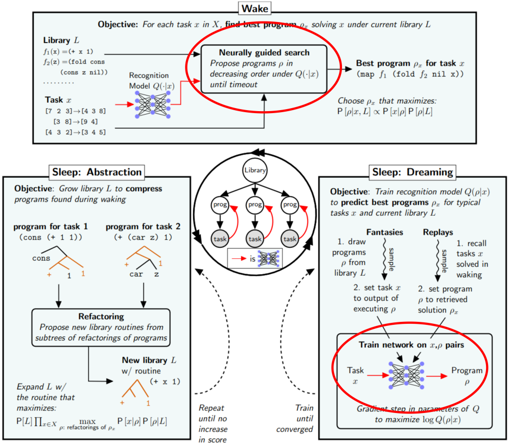
Anti-unification: Introduction, Applications, and Recent Results

David M. Cerna
Czech Academy of Sciences,
Institute of Computer Science

February 8^{th} 2024

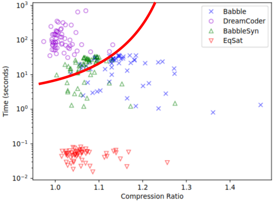
Czech Academy of Sciences

DreamCoder: library learning modulo theory

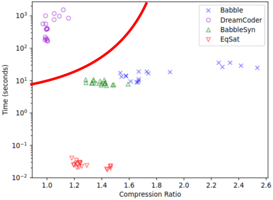


DreamCoder: Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning, 2021, Ellis *et al.*, PLDI

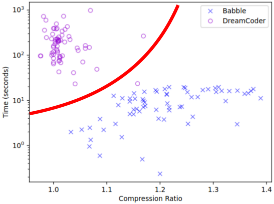
Babble: library learning modulo theory



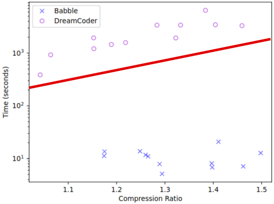
(a) List domain



(b) Physics domain



(c) Text domain

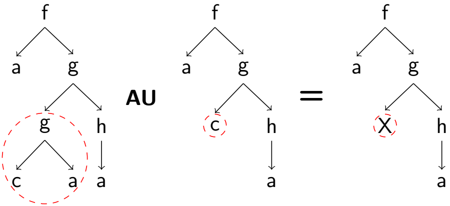


(d) Logo domain

Babble: Learning Better Abstractions with E-Graphs and Anti-Unification, Cao *et al.*, POPL

What is it?

- Unification: is a process by which two symbolic expressions may be identified through variable replacement.
- Anti-unification: A process that derives from a set of symbolic expressions a new symbolic expression possessing certain commonalities shared between its members.



- Independently introduced by Plotkin and Reynolds in 1970.
  - "*A note on inductive generalization*" by G. D. Plotkin
  - "*Transformational systems and the algebraic structure of atomic formulas*" by J.C. Reynolds

Anti-Unification: Basics

- Let Σ be signature, $\mathcal{V}$ a countable set of variables, and $\mathcal{T}(\Sigma, \mathcal{V})$ a term algebra.
- (Unification) For $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$:

  Does there exists a substitution σ s.t. $s\sigma = t\sigma$?
- (Anti-Unification) For $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$:

  Does there exists $g \in \mathcal{T}(\Sigma, \mathcal{V})$ and substitutions σ_s and σ_t s.t. $g\sigma_s = s$ and $g\sigma_t = t$?
- The term g is referred to as a generalization of s and t.
- While a substitution σ such that $s\sigma = t\sigma$ may not exists, $x \in \mathcal{V}$ always generalizes s and t (typically...):

$$\sigma_s = \{x \mapsto s\} \;,\; \sigma_t = \{x \mapsto t\}$$

- Let's look at an example.

Anti-Unification: Basics

$$f(g(x,a)) \stackrel{?}{=} f(y)$$

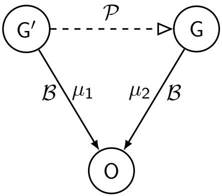
- $\{x \leftarrow a\ ,\ \ y \leftarrow g(a,a)\}$ is a unifier.
  But, $\{y \leftarrow g(x,a)\}$ is more general.

$$f(g(b,a)) \triangleq f(g(a,a))$$

- $f(y)$ is a generalization, $\{y \leftarrow g(b,a)\}$ and $\{y \leftarrow g(a,a)\}$.
  But, $f(g(y,a))$ is more specific, $\{y \leftarrow b\}$ and $\{y \leftarrow a\}$

- Dual of most general unifier, least general generalization.
- Let g_1 and g_2 be generalizers of t_1 and t_2, then $\underline{g_1 \text{ is less general then } g_2}$, $g_2 \prec g_1$ if there exists μ s.t. $g_2\mu = g_1$.
- g_1 is least general if for every $\underline{\text{comparable}}$ term g_2, $g_2 \prec g_1$.

A General Framework



Generic	Concrete
$\mathcal{O}$	$\mathcal{T}(\Sigma, \mathcal{V})$
$\mathcal{M}$	First-order substitutions
$\mathcal{B}$	$\doteq$ (syntactic equality)
$\mathcal{P}$	$\preceq$: $s \preceq t$ if $s\sigma \doteq t$ for some σ

- **Goal:** from $O_1, O_2 \in \mathcal{O}$ (symbolic expressions) derive $G \in \mathcal{O}$ possessing certain commonalities shared by O_1 and O_2.
- **Specification:** define (a) a class of mappings $\mathcal{M}$ from $\mathcal{O} \to \mathcal{O}$, (b) a base relation $\mathcal{B}$ consistent with $\mathcal{M}$, and (c) a preference relation $\mathcal{P}$ consistent with $\mathcal{B}$.
- **Result:** G is a $\mathcal{B}$-generalization of O_1 and O_2 and most $\mathcal{P}$-preferred ("better" than G').

A General Framework

- A set $\mathcal{G} \subset \mathcal{O}$ is called $\mathcal{P}$-complete set of $\mathcal{B}$-generalizations of $O_1, O_2 \in \mathcal{O}$ if:
  - **Soundness:** Every $G \in \mathcal{G}$ is a $\mathcal{B}$-generalization of O_1 and O_2.
  - **Completeness:** For each $\mathcal{B}$-generalization G' of O_1 and O_2, there exists $G \in \mathcal{G}$ such that $\mathcal{P}(G', G)$ (G is more preferred).
- Furthermore, $\mathcal{G}$ is minimal if:
  - **Minimality:** No distinct elements of $\mathcal{G}$ are $\mathcal{P}$-comparable: if $G_1, G_2 \in \mathcal{G}$ and $\mathcal{P}(G_1, G_2)$, then $G_1 = G_2$.
- Minimal Complete sets come in four Types:
  - **Unitary** ($\mathbf{1}$): $\mathcal{G}$ is a singleton,
  - **Finitary** (ω): $\mathcal{G}$ is finite and contains at least two elements,
  - **Infinitary** (∞): $\mathcal{G}$ is infinite,
  - **Nullary** ($\mathbf{0}$): $\mathcal{G}$ does not exist (minimality and completeness contradict each other).
- Types are extendable to generalization problems.

Complete sets of solutions

- Here are some examples for each category of complete sets:
  - **UNITARY:**
    - First-Order terms
    - High-Order patterns **(and friends)**
  - **FINITARY:**
    - FO terms, associative and/or commutative symbols
    - Unranked Terms and Hedges
    - FO terms, one symbol has a unit element
  - **INFINITARY:**
    - FO terms, idempotent symbols
    - FO terms, absorbing Yesterday's talk (A. F. G. Barragán)
  - **NULLARY:**
    - Semirings
    - FO terms, more than one symbol has a unit element
    - Simply typed lambda calculus
    - Cartesian Combinators

Rule-Based Algorithm

- $x : t \triangleq s$ is an anti-unification problem (AUP).
- A configuration is a triple $A; S; G$ where
  - A is a set of AUPs **(Active)**
  - S is a set of AUPs **(Solved)**
  - G is a set of AUPs **(Generalization)**
- The initial state for an AUP $x : t \triangleq s$ is $\{x : t \triangleq s\};\ \emptyset;\ x$.
- Inference rules transform configurations into configurations.
- A configurations is final when no rules may be applied.

Rule-Based AU: Examples

Dec: **Decomposition**

$\{\underline{x : f(\overline{t_m}) \triangleq f}(\overline{s_m})\} \uplus A;\ S;\ G \Longrightarrow$
$\{y_m : t_m \triangleq s_m\} \cup A; S;\ G\{x \mapsto f(\overline{y_m})\},$
where $y_1, \ldots, y_m$ are fresh variables

Sol: **Solve Rule**

$\{x : t \triangleq s\} \uplus A;\ S;\ G \Longrightarrow A;\ \{x : t \triangleq s\} \cup S;\ G,$
$head(t) \neq head(s)$ and y is a fresh variable.

Mer: **Merge Rule**

$A;\ \{x : t_1 \triangleq t_2, y : s_1 \triangleq s_2\} \uplus S;\ G \Longrightarrow$
$A;\ \{x : t_1 \triangleq t_2\} \cup S;\ G\,\{y \mapsto x\}\,,$

$t_1 = s_1$ and $t_2 = s_2$.

Rule-Based AU: Examples

$\{x : f(g(a,c), h(b,a,b)) \triangleq f(a, h(a,a,a))\}; \emptyset; x$

$\Longrightarrow_{\text{Dec}}$

$\{x_1 : g(a,c) \triangleq a, x_2 : h(b,a,b) \triangleq h(a,a,a)\}; \emptyset; f(x_1, x_2)$

$\Longrightarrow_{\text{Sol}}$

$\{x_2 : h(b,a,b) \triangleq h(a,a,a)\}; \{x_1 : g(a,c) \triangleq a\}; f(x_1, x_2)$

$\Longrightarrow_{\text{Dec}}$

$\{x_3 : b \triangleq a, x_4 : a \triangleq a, x_5 : b \triangleq a\}; \{x_1 : g(a,c) \triangleq a\}; f(x_1, h(x_3, x_4, x_5))$

$\Longrightarrow_{\text{Dec}}$

$\{x_3 : b \triangleq a, x_5 : b \triangleq a\}; \{x_1 : g(a,c) \triangleq a\}; f(x_1, h(x_3, a, x_5))$

$\Longrightarrow_{\text{Sol}^{\times 2}}$

$\emptyset; \{x_1 : g(a,c) \triangleq a, x_3 : b \triangleq a, x_5 : b \triangleq a\}; f(x_1, h(x_3, a, x_5))$

$\Longrightarrow_{\text{mer}}$

$\emptyset; \{x_1 : g(a,c) \triangleq a, x_3 : b \triangleq a\}; f(x_1, h(x_3, a, x_3))$

Applications of Anti-unification

- Many applications are covered in the following Survey:

  Anti-unification and Generalization: A Survey, D.M. Cerna and T. Kutsia, IJCAI 2023 doi.org/10.24963/ijcai.2023/736

- Anti-unification is often used to build templates.

  If objects match the template then they ought to behave similarly in a given situation.

- Investigations have used anti-unification and similar techniques for inductive synthesis.

Apps: Inductive Synthesis

- Second-order anti-unification for program Replay.

  The Replay of Program Derivations, R.W. Hasker, 1995, Thesis

- θ-subsumption for building bottom clauses.

  Inverse entailment and Progol, S. Muggleton, 1995, NGCO

- Lggs used for recursive functional program synthesis.

  IGOR II – an Analytical Inductive Functional Programming System, M. Hofmann, 2010, PEPM

- Anti-unification for templating the recursion step.

  Inductive Synthesis of Functional Programs: An Explanation Based Generalization Approach, E. Kitzelmann U. Schmid, 2006, JMLR

- Flash-fill in Microsoft Excel.

  Programming by Example using Least General Generalizations, By M. Raza, S. Gulwani, N. Milic-Frayling, 2014, AAAI

Applications:Bugs and Optimizations

- Extracting fixes from repository history.

  Learning Quick Fixes from Code Repositories by R. Sousa , *et al.*, 2021, SBES

- Templating bugs with corresponding fixes.

  Getafix: Learning to Fix Bugs Automatically By J. Bader, *et al.*, 2019, OOPSLA

- Templating configuration files to catagorize errors.

  Rex: Preventing Bugs and Misconfiguration in Large Services Using Correlated Change Analysis By Sonu Mehta, *et al.*, 2020, NSDI

- Optimization of recursion schemes for efficient parallelizability.

  Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification By A. D. Barwell, C. Brown, K. Hammond, 2017, FGCS

Applications: Theorem Proving

- Extraction of substitutions from substitution trees.

  Higher-order term indexing using substitution trees By B. Pientka, 2009, ACM TOCL

- Grammar compression and inductive theorem proving.

  Algorithmic Compression of Finite Tree Languages by Rigid Acyclic Grammars, By S. Eberhard, G. Ebner, S. Hetzl, 2017, ACM TOCL

- Generating SyGuS problems.

  Reinforcement Learning and Data-Generation for Syntax-Guided Synthesis, By J. Parsert and E. Polgreen, 2024, AAAI

Anti-unification over Lambda Terms

- Let $\mathcal{B}$ be a set of base types and Types is the set of types inductively constructed from δ and $\rightarrow$.
- The set Λ is constructed using the following grammar:

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

- A lambda term is a pattern if free variables only apply to distinct bound variables.
- $\lambda x.f(X(x), c)$ is a pattern, but $\lambda x.f(X(X(x)), c)$ and $\lambda x.f(X(x, x), c)$ are not.
- Anti-unification of an AUP $X(\vec{x}) : t \triangleq s$ often requires
  - t and s are of the same type ,
  - t and s are in η-long β-normal form,
  - and X does not occur in t and s.

Anti-unification over Lambda Terms

- Calculus of Constructions, pattern fragment.

  Unification and anti-unification in the calculus of construction By F. Pfenning, 1991, LICS

- Anti-unification in $\lambda 2$ ($\mathcal{P}$ based on β-reduction).

  Higher order generalization and its application in program verification, Lu *et al.*, 2000, AMAI

- Pattern Anti-unification in simply-typed λ-calculus.

  Higher-order pattern anti-unification in linear time, A. Baumgartner *et al.*, 2017, JAR

- Top-maximal shallow, simply-typed λ-calculus.

  A generic framework for higher-order generalization, D. Cerna and T. Kutsia, 2019, FSCD

- λ-calculus with recursive let expressions.

  Towards Fast Nominal Anti-unification of Letrec-Expressions, M. Schmidt-Schauß, D. Nantes-Sobrinho *et al.*, 2023, CADE

Rules: Pattern Anti-unification

Dec: **Decomposition**

$\{\underline{X(\vec{x}) : \mathrm{h}(\overline{t_m}) \triangleq} \, \mathrm{h}(\overline{s_m})\} \uplus A;\ S;\ \sigma \Longrightarrow$
$\{\overline{Y_m(\vec{x}) : t_m \triangleq s_m}\} \cup A; S;\ G\{X \mapsto \lambda\vec{x}.h(\overline{Y_m(\vec{x})})\}$,
where h is constant or $h \in \vec{x}$, and $\overline{Y_m}$ are fresh variables of the appropriate types.

Abs: **Abstraction Rule**

$\{X(\vec{x}) : \lambda y.t \triangleq \lambda z.s\} \uplus A;\ S;\ \sigma \Longrightarrow \{X'(\vec{x}, y) : t \triangleq$
$s\{z \mapsto y\}\} \cup A;\ S;\ G\,\{X \mapsto \lambda\vec{x}, y.X'(\vec{x}, y)\}$,
where X' is a fresh variable of the appropriate type.

Extensions: Lambda Terms

Sol: **Solve Rule**

$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ \sigma \Longrightarrow$
$A;\ \{Y(\vec{y}) : t \triangleq s\} \cup S;\ G\{X \mapsto \lambda \vec{x}.Y(\vec{y})\}$,

where t and s are of a base type, $head(t) \neq head(s)$ or $head(t) = head(s) = \mathrm{h} \notin \vec{x}$. The sequence $\vec{y}$ is a subsequence of $\vec{x}$ consisting of the variables that appear freely in t or in s, and Y is a fresh variable of the appropriate type.

Mer: **Merge Rule**

$A;\ \{X(\vec{x}) : t_1 \triangleq t_2, Y(\vec{y}) : s_1 \triangleq s_2\} \uplus S;\ \sigma \Longrightarrow A;\ \{X(\vec{x}) : t_1 \triangleq t_2\} \cup S;\ G\{Y \mapsto \lambda \vec{y}.X(\vec{x}\pi)\}$,

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

Pattern Anti-unification: Example

$\{X : \lambda x, y.f(u(g(x), y), u(g(y), x)) \triangleq \lambda x', y'.f(h(y', g(x')), h(x', g(y')))\};$
$\emptyset; X \Longrightarrow_{\mathsf{Abs}\times 2}$
$\{X'(x, y) : f(u(g(x), y), u(g(y), x)) \triangleq f(h(y, g(x)), h(x, g(y)))\};\ \emptyset;$
$\lambda x, y.X'(x, y) \Longrightarrow_{\mathsf{Dec}}$
$\{Y_1(x, y) : u(g(x), y) \triangleq h(y, g(x)), Y_2(x, y) : u(g(y), x) \triangleq h(x, g(y))\};\ \emptyset;$
$\lambda x, y.f(Y_1(x, y), Y_2(x, y)) \Longrightarrow_{\mathsf{Sol}}$
$\{Y_2(x, y) : u(g(y), x) \triangleq h(x, g(y))\}; \{Y_1(x, y) : u(g(x), y) \triangleq h(y, g(x))\};$
$\lambda x, y.f(Y_1(x, y), Y_2(x, y)) \Longrightarrow_{\mathsf{Sol}}$
$\emptyset;\ \{Y_1(x, y) : u(g(x), y) \triangleq h(y, g(x)), Y_2(x, y) : u(g(y), x) \triangleq h(x, g(y))\};$
$\lambda x, y.f(Y_1(x, y), Y_2(x, y)) \Longrightarrow_{\mathsf{Mer}}$
$\emptyset;\ \{Y_1(x, y) : u(g(x), y) \triangleq h(y, g(x))\};\ \lambda x, y.f(Y_1(x, y), Y_1(y, x))$

Friends of Patterns

- While useful, patterns are quite inexpressive.

  Functions-as-Constructors Higher-Order Unification, T. Libal and D. Miller, 2016, FSCD

  - Restricted terms occur as arguments to free variables.
  - Restricted terms are inductively constructed from bound variables and constant symbols with arity > 0.
  - Arguments cannot be subterms of each other.
    - $X(f(x), y)$ is ok, but not $X(f(x), x)$.
  - Arguments cannot be proper subterms of each other.
    - $g(X(f(x), y), Y(f(x), z))$ is ok, but not $g(X(f(x), y), Y(x))$.
- Unitary, but is Finitary without variable restrictions.
- Anti-unification is Unitary without most restrictions.

Friends of Patterns

- Rules construct Top-maximal Shallow Generalizations.
  - $\lambda x.f(X(x))$ is preferred to $\lambda x.X(f(x))$ when possible.
  - $\lambda x.f(X(X(x)))$ or $\lambda x.f(X(Y(x)))$ not allowed.
- Only the Solve rule changes:

Sol: **Solve**

$$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ r \Longrightarrow A;\ \{Y(y_1, \dots, y_n) : (C_t\, y_1 \cdots y_n) \triangleq (C_s\, y_1 \cdots y_n)\} \cup S;\ r\{X \mapsto \lambda\vec{x}.Y(q_1, \dots, q_n)\},$$

where t and s are of a basic type, $head(t) \neq head(s)$, $q_1, \dots, q_n$ are distinct subterms of t or s, C_t and C_s are terms such that $(C_t\, q_1 \cdots q_n) = t$ and $(C_s\, q_1 \cdots q_n) = s$, C_t and C_s do not contain any $x \in \vec{x}$, and Y, $y_1, \dots, y_n$ are distinct fresh variables of the appropriate type.

- Pattern if the $q_1, \dots, q_n \in \vec{x}$, and $C_t = \lambda\vec{x}.t$ and $C_s = \lambda\vec{x}.s$.

Anti-Unification beyond Patterns

- Not every choice of C_s and C_t will result in a Unitary variant.
- Inconsistent choices for C_s and C_t can result in the computation of non-lggs.
- In particular how the $q_i s$ are chosen matters:
  - $q_i s$ must match a selection condition.
  - $q_i s$ must occur in both terms.
  - $q_i s$ must not be positionally ordered within the terms.
- These conditions allowed us to define 4 Unitary variants.

Anti-Unification beyond Patterns

- Projection Anti-Unification:
  - $q_1 = t,\ q_2 = s,\ C_t = \lambda z_1, z_2.z_1,\ C_s = \lambda z_1, z_2.z_2.$
- Common Subterms Anti-Unification:
  - $q_i s$ position maximal common subterms.
  - $C_t = \lambda y_1, \ldots, y_n.\, t[p_1 \mapsto y_1] \cdots [p_m \mapsto y_n]$
  - $C_s = \lambda y_1, \ldots, y_n.\, s[l_1 \mapsto y_1] \cdots [l_m \mapsto y_n]$
- Restricted Function-as-constructor Anti-Unification:
  - $q_i s$ position maximal common subterms minus those which break the Local variable condition.
  - C_t and C_s are the same.
- Function-as-constructor Anti-Unification:
  - $q_i s$ position maximal common subterms minus those which break the Local/Global variable conditions.
  - C_t and C_s are the same.
- Other variants are definable (based on the selection condition).

Anti-Unification beyond Patterns: Example

$$\{X : \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x)))) \triangleq$$
$$\lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))\};\ \emptyset;\ X$$

$\Longrightarrow_{\mathsf{Abs}}$

$$\{X'(x) : f(h_1(g(g(x)), a, b), h_2(g(g(x)))) \triangleq$$
$$f(h_3(g(g(x)), g(x), a), h_4(g(g(x))))\}; \emptyset;\ \lambda x.X'(x)$$

$\Longrightarrow_{\mathsf{Dec}}$

$$\{Z_1(x) : h_1(g(g(x)), a, b) \triangleq h_3(g(g(x)), g(x), a),$$
$$Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x))\};\ \emptyset;$$
$$\lambda x.f(Z_1(x), Z_2(x))$$

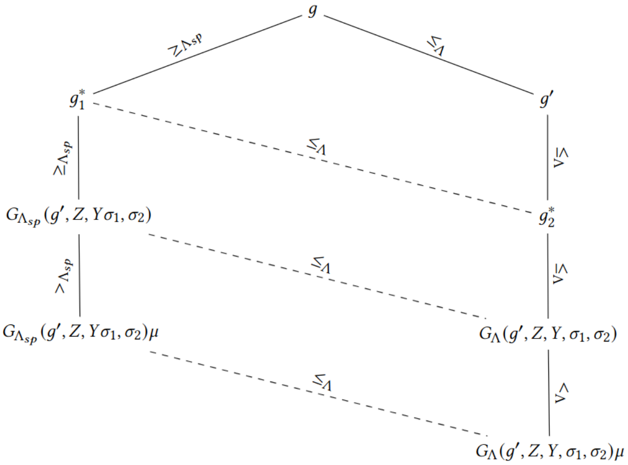
$\Longrightarrow_{\mathsf{Sol\text{-}RFC}}$

Anti-Unification beyond Patterns: Example

$$\{Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};$$
$$\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a)\};$$
$$\lambda x.f(Y_1(g(x)), Z_2(x))$$
$$\Longrightarrow_{\text{Sol-RFC}}$$
$$\emptyset;\ \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),$$
$$Y_2(y_2) : h_2(y_2) \triangleq h_4(y_2)\};$$
$$\lambda x.f(Y_1(g(x)), Y_2(g(g(x)))).$$

- Extending this idea to higher-type theories such as the calculus of constructions (COC) has yet to be considered?
- Beneficial for proof generalization.
- What happens when the terms are no longer shallow?

Deep Lambda Terms: Nullarity



- $\lambda x.\lambda y.f(x) \triangleq \lambda x.\lambda y.f(y)$ has no solution set.
- $\lambda x.\lambda y.f(Z(x,y)) < \lambda x.\lambda y.f(Z(Z(x,y),Z(x,y))) < \cdots$

Deep Lambda Terms: Nullarity

- Its pattern generalization is $g^p = \lambda x.\lambda y.f(Z(x,y))$.
- A generalization more specific g^p is pattern-derived

Definition

Let g be pattern-derived. Then g is *tight* if for all $W \in \mathcal{FV}(g)$:

1) $g\{W \mapsto \lambda\overline{b_k}.b_i\} \notin \mathcal{G}(s,t)$, if W has type $\overline{\gamma_k} \to \gamma_i$ and for $1 \leq i \leq k$ and $\gamma_i \in \mathcal{B}$, and
2) For $(\sigma_1, \sigma_2) \in \mathcal{GS}(s,t,g)$, $g\{W \mapsto t_1\}, g\{W \mapsto t_2\} \notin \mathcal{G}(s,t)$ where $t_1 = W\sigma_1$, $t_2 = W\sigma_2$.

Deep Lambda Terms: Nullarity

Definition

Let $g = \lambda x.\lambda y.f(Z(\overline{s_m}))$ be a tight generalization of $s \triangleq t$ where

1) Z has type $\overline{\delta_m} \to \alpha$ for $1 \leq i \leq m$, and s_i has type δ_i.
2) $(\sigma_1, \sigma_2) \in \mathcal{GS}(s, t, g)$ such that $Z\sigma_1 = r_1$ and $Z\sigma_2 = r_2$,
3) r_1 and r_2 are of type $\overline{\delta_m} \to \alpha$, and
4) Y such that $Y \notin \mathcal{FV}(g)$ and has type $\alpha \to \alpha \to \alpha$.

Then the *g-pseudo-pattern*, denoted $G(g, Z, Y, \sigma_1, \sigma_2)$, is

$$g\{Z \mapsto \lambda\overline{b_m}.Y(r_1(\overline{b_m}), r_2(\overline{b_m}))\} = \lambda x.\lambda y.f(Y(r_1(\overline{q_m}), r_2(\overline{q_m})))$$

where for all $1 \leq i \leq m$, $q_i = s_i\{Z \mapsto \lambda\overline{b_m}.Y(r_1(\overline{b_m}), r_2(\overline{b_m}))\}$.

- Essentially, we regularized the structure of the generalizations.

Deep Lambda Terms: Nullarity

Theorem

For anti-unification of simply-typed lambda terms is nullary.

Proof.

Let us assume that $C \subseteq \mathcal{G}(s,t)$ is minimal and complete. We know C contains a pattern-derived generalization g. Observe that g can be transformed into an tight generalization g' that is also pattern-derived. We can derive a pseudo-pattern generalization g'' of g'. Finally, $g^* = g''\{Y \mapsto \lambda w_1.\lambda w_2.Y(Y(w_1,w_2),Y(w_1,w_2))\}$ is strictly more specific than g''. This implies that $g <_{\mathcal{L}} g^*$, entailing that C is not minimal. □

- Result extendable to non-shallow fragments.

One or nothing: Anti-unification over the simply-typed lambda calculus, D. Cerna and M. Buran, 2022, Arxiv (under-review).

Equational Anti-unification

- Anti-unification over commutative theories.

  Unification, weak unification, upper bound, lower bound, and generalization problem, F. Baader, 1991, RTA

- Grammar for a **complete** set of E-generalizations:

  E-generalization using grammars, J. Burghardt, 2005, AI

- Minimal complete set of AC-generalizations.

  A modular order-sorted equational generalization algorithm, M. Alpuente *et al.*, 2014, Inf. Comput.

- Minimal complete set of I-generalizations.

  Idempotent anti-unification, D. Cerna and T. Kutsia, 2020, ACM TOCL

- Nullarity of U^2-generalization.

  Unital anti-unification: Type and algorithms, M. D. Cerna and T. Kutsia, 2020, FSCD

E-generalization: Important, but Explosive

- Many equational theories are not well behaved:

Problem	Theory	Type
$f(a,b) \triangleq f(b,a)$	$f(x,x) = x,$	∞
$g(\varepsilon_f, f(a, h(\varepsilon_f))) \triangleq g(f(h(\varepsilon_f), a), \varepsilon_f)$	$f(\varepsilon_f, x) = f(x, \varepsilon_f) = \varepsilon_f$	∞
$0 \triangleq 1$	Semirings	0
$a \triangleq b$	f(a)=a, f(b)=b	0

- Even when there are *least general generalizations*,
- are the majority of them useful? $f(f(f(\cdots f(x)\cdots)))$
- Though, not all theories behave badly....

Equantional Anti-unification: A and C

- AC-Anti-unification is finitary.
  - Though the minimal complete set may have an exponential number generalizations.
- Assuming that f is associative:

$$X : f(a, a, b, b) \triangleq f(a, b, b) \quad \text{(Flattened for Readability)}$$

- Note that there are many ways to decompose the problem:

$$
\begin{aligned}
&X_1 : a \triangleq a && X_2 : f(a, b, b) \triangleq f(b, b) && (1)\\
&X_1 : a \triangleq f(a, b) && X_2 : f(a, b, b) \triangleq b && (2)\\
&X_1 : f(a, a, b) \triangleq a && X_2 : b \triangleq f(b, b) && (3)\\
&X_1 : f(a, a) \triangleq a && X_2 : f(b, b) \triangleq f(b, b) && (4)
\end{aligned}
$$

Equantional Anti-unification: A and C

- If we continue this decomposition the lggs are:

$$g_1 = f(X_1, b, b) \quad g_2 = f(a, X_2, b)$$

- g_1 and g_2 are $\prec_A$-incomparable, and form the minimal complete set for the terms $f(a, a, b, b)$ and $f(a, b, b)$.
- To compute the minimal complete set modulo associativity we extend the syntactic algorithm by the following rules:

Equantional Anti-unification: A Rules

Dec-A-L: **Associative Decomposition Left**

$$\{X : f(t_1, \ldots, t_k, t_{k+1} \ldots, t_n) \triangleq f(s_1, s_2 \ldots, s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1 : f(t_1, \ldots, t_k) \triangleq s_1,\ Y_2 : f(t_{k+1} \ldots, t_n) \triangleq$$
$$f(s_2 \ldots, s_m)\} \cup A;\ S;\ G\{X \mapsto f(Y_1, Y_2)\},$$

where f is associative, $n, m \geq 2$, $1 \leq k \leq n-1$, and Y_1 and Y_2 are fresh variables.

Dec-A-R: **Associative Decomposition Right**

$$\{X : f(t_1, t_2 \ldots, t_n) \triangleq f(s_1, \ldots, s_k, s_{k+1} \ldots, s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1 : t_1 \triangleq f(s_1, \ldots, s_k),\ Y_2 : f(t_2 \ldots, t_n) \triangleq$$
$$f(s_{k+1} \ldots, s_m)\} \cup A;\ S;\ G\{X \mapsto f(Y_1, Y_2)\},$$

where f is associative, $n, m \geq 2$, $1 \leq k \leq m-1$, and Y_1 and Y_2 are fresh variables

Equantional Anti-unification: A and C

- Similarly one can define Commutative anti-unification.
- We assume that f is commutative:

$$X : f(a, f(a, b)) \triangleq f(b, f(b, a))$$

- There are only two ways to decompose:

$$X_1 : a \triangleq b \qquad X_2 : f(a, b) \triangleq f(b, a) \tag{5}$$

$$X_1 : a \triangleq f(b, a) \qquad X_2 : f(a, b) \triangleq b \tag{6}$$

- Furthermore, one of the possible decompositions is syntactic.

Equantional Anti-unification: A and C

- Continuing this decomposition we get two lggs:

$$g_1 = f(x, f(a, b)) \quad g_2 = f(x, f(x, y))$$

- Observe, g_1 and g_2 are $\prec_C$-incomparable and form the minimal complete set.
- To computing the minimal complete set modulo commutatively we extend the syntactic algorithm by the following rule:

Equantional Anti-unification: A and C

Dec-C: **Commutative Decomposition**

$\{X : f(t_1, t_2) \triangleq f(s_1, s_2)\} \uplus A;\ S;\ \sigma \Longrightarrow \{Y_1 : t_1 \triangleq s_i,\ \ Y_2(\vec{x}) : t_2 \triangleq s_{(i \mod 2)+1}\} \cup A;\ S;\ G\{X \mapsto f(Y_1, Y_2)\},$

where f is commutative, $i \in \{1, 2\}$, and Y_1 and Y_2 are fresh variables

- We can combine the A and C inference rules and construct an even more flexible anti-unification algorithm.
- This combined anti-unification problem is still Finitary.
- $f(a, a, b) \triangleq f(a, b, b)$ has solutions $f(a, b, x)$ and $f(x, x, y)$.

Selection Heuristics

- How to deal with the explosion?
  - Alignment and Rigidity functions
  - Skeletons
  - beam search
  - Syntactic restriction
- **Recent Direction:**
  - Should the preference and base relations be Crisp?
  - Are most lggs too distant from the generalized terms to be generalizations?
- Is **similarity** and **quantitative** anti-unification a fix?

A Framework for Approximate Generalization in Quantitative Theories, T. Kutsia and C. Pau, 2022, FSCD

Future Work

- Investigating the above questions
- New applications for anti-unification
- Developing methods for combining anti-unification algorithms for disjoint equational theories
- Characterization of classes of equational theories that exhibit similar behavior and properties
- Studying computational complexity and optimizations.