Anti-unification and Generalization: A Survey

David M. Cerna\textsuperscript{1}\textsuperscript{*}, Temur Kutsia\textsuperscript{2}

\textsuperscript{1}Czech Academy of Sciences Institute of Computer Science (CAS ICS), Prague, Czechia
\textsuperscript{2}Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria
dcerna@cs.cas.cz, kutsia@risc.jku.at

Abstract

Anti-unification (AU) is a fundamental operation for generalization computation used for inductive inference. It is the dual operation to unification, an operation at the foundation of automated theorem proving. Interest in AU from the AI and related communities is growing, but without a systematic study of the concept nor surveys of existing work, investigations often resort to developing application-specific methods that existing approaches may cover. We provide the first survey of AU research and its applications and a general framework for categorizing existing and future developments.

1 Introduction

Anti-unification (AU), also known as generalization, is a fundamental operation used for inductive inference. It is abstractly defined as a process deriving from a set of symbolic expressions a new symbolic expression possessing certain commonalities shared between its members. It is the dual operation to unification, an operation at the foundation of modern automated reasoning and theorem proving \cite{Baader18}. AU was introduced by Plotkin \cite{Plotkin70} and Reynolds \cite{Reynolds70} and may be illustrated as follows:

\begin{center}
\begin{tikzpicture}
\node (a) at (0, 0) {a};
\node (b) at (1, 0) {g};
\node (c) at (0, -1) {c};
\node (d) at (1, -1) {h};
\node (e) at (0.5, 0) {f};
\node (f) at (0.5, -1) {X};
\node (g) at (1.5, 0) {a};
\node (h) at (2, -1) {a};
\node (i) at (1.5, -1) {h};
\node (j) at (2.5, 0) {f};
\node (k) at (2.5, -1) {a};
\node (l) at (3, -1) {g};
\node (m) at (2.5, -2) {a};
\node (n) at (3.5, 0) {f};
\node (o) at (3.5, -1) {a};
\node (p) at (4, -1) {g};
\node (q) at (3.5, -2) {a};
\node (r) at (4.5, 0) {f};
\node (s) at (4.5, -1) {a};
\node (t) at (5, -1) {g};
\node (u) at (4.5, -2) {a};
\node (v) at (5.5, 0) {f};
\node (w) at (5.5, -1) {a};
\node (x) at (6, -1) {g};
\node (y) at (5.5, -2) {a};
\node (z) at (6.5, 0) {f};
\node (aa) at (6.5, -1) {a};
\node (bb) at (7, -1) {g};
\node (cc) at (6.5, -2) {a};
\node (dd) at (7.5, 0) {f};
\node (ee) at (7.5, -1) {a};
\node (ff) at (8, -1) {g};
\node (gg) at (7.5, -2) {a};
\node (hh) at (8.5, 0) {f};
\node (ii) at (8.5, -1) {a};
\node (jj) at (9, -1) {g};
\node (kk) at (8.5, -2) {a};
\node (ll) at (9.5, 0) {f};
\node (mm) at (9.5, -1) {a};
\node (nn) at (10, -1) {g};
\node (oo) at (9.5, -2) {a};
\node (pp) at (10.5, 0) {f};\end{tikzpicture}
\end{center}

Figure 1: Illustration of anti-unification between two terms.

where \( f(a, g(c, a), h(a)) \) and \( f(a, g(c, h(a)) \) are two first-order terms we want to anti-unify and \( f(a, g(X, h(a)) \) is the resulting generalization; mismatched sub-terms are replaced by variables. Note that \( f(a, g(X, h(a)) \) captures the common structure, and through substitution, either input term is derivable. Additionally, \( f(a, g(X, h(a)) \) is commonly referred to as the least general generalization as there does not exist a more specific term capturing all common structure. The term \( f(a, X) \) is more general, partly covers common structure.

Early inductive logic programming (ILP) \cite{Cropper22} approaches exploited the relationship between generalizations to learn logic programs \cite{Muggleton95}. Modern ILP approaches, such as Popper \cite{Cropper21}, use this mechanism to simplify the search iteratively. The programming by example (pbe) \cite{Gulwani16} paradigm integrates syntactic anti-unification methods to find the least general programs satisfying the input examples. Recent work concerning library learning and compression \cite{Cao23} exploits equational anti-unification to find suitable programs efficiently and outperforms Dreamcoder \cite{Ellis21}, the previous state of the art approach.

Applications outside the area of inductive synthesis typically exploit the following observation: “Syntactic similarity often implies semantic similarity”. A notable example is automatic parallel recursion scheme detection \cite{Barwell18} where templates are developed, with the help of AU, allowing the replacement of non-parallelizable recursion by parallelized higher-order functions. Other uses are learning program repairs from repositories \cite{deSousa21}, preventing misconfigurations \cite{Mehta20}, and detecting software clones \cite{Vanhoof19}.

There is growing interest in anti-unification, yet much of the existing work is motivated by specific applications. The lack of a systematic investigation has led to, on occasion, reinvention of methods and algorithms. Illustratively, the authors of Babble \cite{Cao23} developed an E-graph anti-unification algorithm motivated solely by the seminal work of Plotkin \cite{Plotkin70}. Due to the fragmentary nature of the anti-unification literature, the authors missed relevant work on equational \cite{Burghardt05} and term-graph anti-unification \cite{Baumgartner18} among others. The discovery of these earlier papers could have probably sped up, improved, and/or simplified their investigation.

Unlike its dual unification, there are no comprehensive surveys, and little emphasis is put on developing a strong theoretical foundation. Instead, practically oriented topics dominate current research on anti-unification. This situation is unsurprising as generalization, in one form or another, is an essen-
tial ingredient within many applications: reasoning, learning, information extraction, knowledge representation, data compression, software development, and analysis, in addition to those already mentioned.

New applications pose new challenges. Some require studying generalization problems in a completely new theory, while others may be addressed adequately by improving existing algorithms. Classifying, analyzing, and surveying the known methods and their applications is of fundamental importance to shape the field and help researchers to navigate the current fragmented state-of-the-art on generalization.

In this survey, we provide (i) a general framework for the generalization problem, (ii) an overview of existing theoretical results, (iii) an overview of existing application domains, and (iv) an overview of some future directions of research.

2 Generalization Problems: an Abstract View

The definitions below are parameterized by a set of syntactic objects \( \mathcal{O} \), typically consisting of expressions (e.g., terms, formulas, \( \ldots \)) in some formal language. Additionally, we consider a class of mappings \( \mathcal{M} \) from \( \mathcal{O} \) to \( \mathcal{O} \). We say that \( \mu(O) \) is an instance of the object \( O \) with respect to \( \mu \in \mathcal{M} \).

In most cases, variable substitutions are instances of such mappings. We call elements of \( \mathcal{M} \) generalization mappings.

Our definition of the generalization problem requires two relations: The base relation defining what it means for an object to be a generalization of another, and the preference relation, defining a notion of rank between generalizations. These relations are defined abstractly, with minimal requirements. We provide the concrete instances of the base and preference relations and generalization mappings for each concrete generalization problem. One ought to consider the base relation as describing what we mean when we say an object is a generalization of another and the preference relation as describing the quality of generalizations with respect to one another. The mappings can be thought of as describing what the generalization of the objects means over the given base relation.

Definition 1. A base relation \( \mathcal{B} \) is a binary reflexive relation on \( \mathcal{O} \). An object \( G \in \mathcal{O} \) is a generalization of the object \( O \in \mathcal{O} \) with respect to \( \mathcal{B} \) and a class of mappings \( \mathcal{M} \) (briefly, \( \mathcal{B}_{\mathcal{M}} \)-generalization) if \( B(\mu(G), O) \) holds for some mapping \( \mu \in \mathcal{M} \). A preference relation \( \mathcal{P} \) is a binary reflexive, transitive relation (i.e., a preorder) on \( \mathcal{O} \). We write \( \mathcal{P}(O_1, O_2) \) to indicate that the object \( O_1 \) is preferred over the object \( O_2 \). It induces an equivalence relation \( \equiv_p \): \( O_1 \equiv_p O_2 \) iff \( \mathcal{P}(O_1, O_2) \) and \( \mathcal{P}(O_2, O_1) \).

We are interested in preference relations that relate to generalizations in the following way:

Definition 2 (Consistency). Let \( \mathcal{B} \) and \( \mathcal{P} \) be, respectively, base and preference relations defined on a set of objects \( \mathcal{O} \) and \( \mathcal{M} \) be a class of generalization mappings over \( \mathcal{O} \). We say that \( \mathcal{B} \) and \( \mathcal{P} \) are consistent on \( \mathcal{O} \) with respect to \( \mathcal{M} \) or, shortly, \( \mathcal{M} \)-consistent, if the following holds: If \( G_1 \) is a \( \mathcal{B}_{\mathcal{M}} \)-generalization of \( O \) and \( \mathcal{P}(G_1, G_2) \) holds for some \( G_2 \), then \( G_2 \) is also a \( \mathcal{B}_{\mathcal{M}} \)-generalization of \( O \). In other words, if \( B(\mu_1(G_1), O) \) for some \( \mu_1 \in \mathcal{M} \) and \( \mathcal{P}(G_1, G_2) \), then there should exist \( \mu_2 \in \mathcal{M} \) such that \( B(\mu_2(G_2), O) \).

Consistency is an important property since it relates otherwise unrelated base and preference relations in the context of generalizations. The intuition being that, for \( \mathcal{M} \)-consistent \( \mathcal{B} \) and \( \mathcal{P} \), \( G_1 \) is “better” than \( G_2 \) as a \( (\mathcal{B}_{\mathcal{M}}, \mathcal{P}) \)-generalization of \( O \) because it provides more information: not only \( G_1 \) is a generalization of \( O \), but also any object \( G_2 \) that is “dominated” by \( G_1 \) in the preference relation. From now on, we assume that our base and preference relations are consistent with respect to the considered set of generalization mappings.

We focus on characterizing common \( \mathcal{B}_{\mathcal{M}} \)-generalizations between multiple objects, selecting among them the “best” ones with respect to the preference relation \( \mathcal{P} \).

Definition 3 (Most preferred common generalizations). An object \( G \) is called a most \( \mathcal{P} \)-preferred common \( \mathcal{B}_{\mathcal{M}} \)-generalization of objects \( O_1, \ldots, O_n \), \( n \geq 2 \) if:

- \( G \) is a \( \mathcal{B}_{\mathcal{M}} \)-generalization of each \( O_i \), and

- for any \( G' \) that is also a \( \mathcal{B}_{\mathcal{M}} \)-generalization of each \( O_i \), if \( \mathcal{P}(G', G) \), then \( G' \equiv_p G \) (i.e., if \( G' \) is \( \mathcal{P} \)-preferred over \( G \), then they are \( \mathcal{P} \)-equivalent).

For \( \mathcal{O}, \mathcal{B}, \mathcal{M}, \) and \( \mathcal{P} \), the \( (\mathcal{B}_{\mathcal{M}}, \mathcal{P}) \)-generalization problem over \( \mathcal{O} \) is specified as follows:

Given: \( \text{Objects } O_1, \ldots, O_n \in \mathcal{O}, n \geq 2 \).

Find: An object \( G \in \mathcal{O} \) that is a most \( \mathcal{P} \)-preferred common \( \mathcal{B}_{\mathcal{M}} \)-generalization of \( O_1, \ldots, O_n \).

This problem may have zero, one, or more solutions. There can be two reasons why it has zero solutions: either the objects \( O_1, \ldots, O_n \) have no common \( \mathcal{B}_{\mathcal{M}} \)-generalization at all (i.e., \( O_1, \ldots, O_n \) are not generalizable, for an example see [Pfenning, 1991]), or they are generalizable but have no most \( \mathcal{P} \)-preferred common \( \mathcal{B}_{\mathcal{M}} \)-generalization.

To characterize “informative” sets of possible solutions, we introduce two notions: \( \mathcal{P} \)-complete and \( \mathcal{P} \)-minimal complete sets of common \( \mathcal{B}_{\mathcal{M}} \)-generalizations of multiple objects:

Definition 4. A set of objects \( \mathcal{G} \) is called a \( \mathcal{P} \)-complete set of common \( \mathcal{B}_{\mathcal{M}} \)-generalizations of the given objects \( O_1, \ldots, O_n \), \( n \geq 2 \), if the following properties are satisfied:

- Soundness: every \( G \in \mathcal{G} \) is a common \( \mathcal{B}_{\mathcal{M}} \)-generalization of \( O_1, \ldots, O_n \), and

- Completeness: for each common \( \mathcal{B}_{\mathcal{M}} \)-generalization \( G' \) of \( O_1, \ldots, O_n \) there exists \( G \in \mathcal{G} \) such that \( \mathcal{P}(G, G') \).

The set \( \mathcal{G} \) is called \( \mathcal{P} \)-minimal complete set of common \( \mathcal{B}_{\mathcal{M}} \)-generalizations of \( O_1, \ldots, O_n \) and is denoted by \( \text{mcsg}_{\mathcal{B}_{\mathcal{M}}, \mathcal{P}}(O_1, \ldots, O_n) \) if, in addition, the following holds:
Minimality: no distinct elements of $G$ are $P$-comparable:
if $G_1, G_2 \in G$ and $P(G_1, G_2)$, then $G_1 = G_2$.

Note that the minimality property guarantees that if $G \in mcsg_{B_M, P}(O_1, \ldots, O_n)$, then no $G'$, differing from $G$, in the $\equiv_P$-equivalence class of $G$ belongs to this set.

In the notation, we may skip $B_M, P$, or both from $mcsg_{B_M, P}$ when it is clear from the context.

**Definition 5 (Generalization type).** We say that the type of the $(B_M, P)$-generalization problem between the generalizable objects $O_1, \ldots, O_n \in O$ is

- **unitary** $(1)$: $mcsg_{B_M, P}(O_1, \ldots, O_n)$ is a singleton,
- **finitary** $(\omega)$: $mcsg_{B_M, P}(O_1, \ldots, O_n)$ is finite and contains at least two elements,
- **infinitary** $(\infty)$: $mcsg_{B_M, P}(O_1, \ldots, O_n)$ is infinite,
- **nullary** $(0)$: $mcsg_{B_M, P}(O_1, \ldots, O_n)$ does not exist (i.e., minimality and completeness contradict each other).

The type of $(B_M, P)$-generalization over $O$ is

- **unitary** $(1)$: each $(B_M, P)$-generalization problem between generalizable objects from $O$ is unitary,
- **finitary** $(\omega)$: each $(B_M, P)$-generalization problem between generalizable objects from $O$ is unitary or finitary, and there exists a finitary problem,
- **infinitary** $(\infty)$: each $(B_M, P)$-generalization problem between generalizable objects from $O$ is unitary or finitary, or infinitary, and there exists an infinitary problem,
- **nullary** $(0)$: there exists a nullary $(B_M, P)$-generalization problem between generalizable objects from $O$.

The basic questions to be answered in this context are,

- **Generalization type:** What is the $(B_M, P)$-generalization type over $O$?
- **Generalization algorithm/procedure:** How to compute (or enumerate) a complete set of generalizations (preferably, $mcsg_{B_M, P}$) for objects from $O$.

If the given objects $O_1, \ldots, O_n$ (resp. the desired object $G$) are restricted to belong to a subset $S \subseteq O$, then we talk about an $S$-fragment (resp. $S$-variant) of the generalization problem. It also makes sense to consider an $S_1$-variant of an $S_2$-fragment of the problem, where $S_1$ and $S_2$ are not necessarily the same.

When $O$ is a set of terms, the linear variant is often considered: the generalization terms do not contain multiple occurrences of the same variable.

The following sections show how some known generalization problems fit into this schema. For simplicity, when it does not affect generality, we consider generalization problems with only two given objects. Also, we skip the word “common” when discussing common generalizations.

Due to space constraints, we do not discuss anti-unification for feature terms [Ait-Kaci and Sasaki, 2001; Armengol and Plaza, 2000], for term-graphs [Baumgartner et al., 2018], nominal [Baumgartner et al., 2015; Schmidt-Schauf and Nantes-Sobrinho, 2022] and approximate anti-unification [Ait-Kaci and Pasi, 2020; Kutsia and Pau, 2022; Pau, 2022]. These generalization problems all fit in our general framework.

### 3 Generalization in First-Order Theories

#### 3.1 First-Order Syntactic Generalization (FOSG)

Plotkin [1970] and Reynolds [1970] introduced FOEG, the simplest and best-known among generalization problems in logic. The objects are first-order terms, and mappings are substitutions that map variables to terms such that all but finitely many variables are mapped to themselves. Application of a substitution $\sigma$ to a term $t$ is denoted by $t\sigma$, which is the term obtained from $t$ by replacing all variable occurrences by their images under $\sigma$. Table 1 specifies the concrete instances of the abstract parameters, and consistency follows from the transitivity of $\geq$. The relation $\equiv_P$ holds between terms that are obtained by variable renaming from each other (e.g., $f(x_1, g(x_1, y_1))$ and $f(x_2, g(x_2, y_2))$).

The most $\equiv_P$-preferred $\geq$-generalizations are called least general generalizations (lggs). Two terms always have an lgg unique up to variable renaming. Plotkin [1970], Reynolds [1970], and Huet [1976] introduced algorithms for computing lggs.

**Example 1.** Let $s_1 = f(a, g(a, b))$ and $s_2 = f(c, g(c, d))$. Their lgg is $t = f(x, g(x, y))$, which is unique up to variable names. Substitutions $\sigma_1 = \{x \mapsto a, y \mapsto b\}$ and $\sigma_2 = \{x \mapsto c, y \mapsto d\}$ give $s_1$ and $s_2$ from $t$. $\sigma t_i = s_i, i = 1, 2$. Note that $s_1$ and $s_2$ have other generalizations as well, e.g., $f(x, g(y, z))$ or $x$, but they are not the $\geq$-preferred ones.

<table>
<thead>
<tr>
<th>Type</th>
<th>Generic</th>
<th>Concrete (FOSG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}$</td>
<td>the set of first-order terms</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{M}$</td>
<td>first-order substitutions</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{B}$</td>
<td>$\equiv$ (syntactic equality)</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>$\geq$ (more specific, less general): $s \geq t$ if $s \equiv t \sigma$ for some $\sigma$</td>
<td></td>
</tr>
<tr>
<td>$\equiv_P$</td>
<td>equi-generality: $\geq$ and $\leq$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Alg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unitary</td>
<td>[Huet, 1976; Plotkin, 1970; Reynolds, 1970]</td>
</tr>
</tbody>
</table>

### 3.2 First-Order Equational Generalization (FOEG)

FOEG requires extending syntactic equality to equality modulo a given set of equations. Many algebraic theories are characterized by axiomatizing properties of function symbols via (implicitly universally quantified) equalities. Some well-known equational theories include

- commutativity, $\mathcal{C}(\{f\})$, $f(x, y) \approx f(y, x)$.
- associativity, $\mathcal{A}(\{f\})$, $f(f(x, y), z) \approx f(x, f(y, z))$.
- associativity and commutativity, $\mathcal{AC}(\{f\})$, above equalities for the same function symbol $f$.
- unital symbols, $\mathcal{U}(\{f, e\})$, $f(x, e) \approx x$ and $f(e, x) \approx x$ ($e$ is both left and right unit element for $f$).
- idempotency, $\mathcal{I}(\{f\})$, $f(x, x) \approx x$.

Given a set of axioms $E$, the equation theory induced by $E$ is the least congruence relation on terms containing $E$ and closed under substitution application. (Slightly abusing the notation, it is also usually denoted by $E$.) When a pair of terms $(s, t)$ belongs to such an equation theory, we say that $s$ and $t$ are equal modulo $E$ and write $s \equiv_E t$.  


### Table 2: First-Order equational generalization.

<table>
<thead>
<tr>
<th>Generic</th>
<th>Concrete (FOEG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>The set of first-order terms</td>
</tr>
<tr>
<td>$M$</td>
<td>First-order substitutions</td>
</tr>
<tr>
<td>$B$</td>
<td>$\equiv_E$ (equivalence modulo $E$)</td>
</tr>
<tr>
<td>$P$</td>
<td>$\geq_E$ (more specific, less general modulo $E$)</td>
</tr>
<tr>
<td>$\equiv_P$</td>
<td>Equi-genericity modulo $E$: $\equiv_P$ and $\geq_E$</td>
</tr>
<tr>
<td>Type</td>
<td>Depends on $E$, fragments, and variants</td>
</tr>
<tr>
<td>Alg.</td>
<td>Depends on $E$, fragments, and variants</td>
</tr>
</tbody>
</table>

In a theory, we may have several symbols that satisfy the same axiom. For instance, $C(\{f, g\})$ denotes the theory where $f$ and $g$ are commutative; $AC(\{f, g\})C(\{h\})$ denotes the theory where $f$ and $g$ are associative-commutative and $h$ is commutative; $U(\{f, e_f\}, \{g, e_g\})$ denotes the theory where $e_f$ and $e_g$ are the unit elements for $f$ and $g$, respectively. We follow the convention that if the equational theory is denoted by $E_1(S_1) \cdots E_n(S_n)$, then $S_1 \cap S_2 = \emptyset$ for each $1 \leq i \neq j \leq n$.

Some results depend on the number of symbols that satisfy the associated equational axioms. We use a special notation for that: For a theory $E$, the notation $E^S$ stands for $E(S)$, where the set $S$ contains a single element; $E^{>1}$ stands for $E(S)$ where $S$ contains finitely many, but at least two elements. When we write only $E$, we mean the equational theory $E(S)$ with a finite set of symbols $S$ that may contain one or more elements.

We can extend this notation to combinations: for instance, $(U)_{>1}$ stands for a theory that contains at least two function symbols, e.g., $f$ and $g$, that are associative and unital (with unit elements $e_f$ and $e_g$).

Table 2 shows how FOEG fits into our general framework.

**Example 2.** Consider a theory $E$ and terms $s$ and $t$.

If $E = AC(\{f\})$, $s = f(f(a, b), a)$, and $t = f(f(b, a), a)$, then $mcsg_E(s, t) = \{f((x, y), x), f((x, a), b))\}$. If we had variables instead of $a$ and $b$, e.g., $s = f(f(z, y), v)$ and $t = f(f(v, z), x)$, then $mcsg_E(s, t) = \{f((x, z), v), f((x, y), x))$. Because $f(x, z, v)$ (the counterpart of $f(x, a, b))$ is more general (less preferred) than $f(x, v, y)$.

If $E = U(\{f\})$, $s = g(f(a, c), a), t = g(c, b)$, then $mcsg_E(s, t) = \{g(f((x, y), x), f((x, c), f((x, y), x))\)$. To see why, e.g., $g(f(x, c), f(y, x))$ from this set is a $U$-generalization of $s$ and $t$, consider substitutions $\sigma = \{x \mapsto a, y \mapsto e\}$ and $\vartheta = \{x \mapsto e, y \mapsto b\}$. Then $g(f(x, c), f(y, x))\sigma = g(f(a, c), f(e, a))_{\vartheta} = g(f(e, c), f(b, e))_{\vartheta}$.

If $E = U(\{f, e_f\}, \{g, e_g\})$, $s = e_f$, and $t = e_g$, then $mcsg_E(s, t)$ does not exist: Any complete set of generalizations of $s$ and $t$ contains elements $g$ and $q$ such that $g \equiv_U g'$ (where $\equiv_U$ is the strict part of $\equiv_U$) [Cerna and Kutsia, 2020c].

If $E = h(\{h\})$, $s = h(a, b)$, $t = h(b, a)$, then $mcsg_E(s, t) = S_\infty$, where $S_\infty$ is the limit of the following construction:

$S_0 = \{h(h(x, b), h(a, y)), h(h(x, a), h(b, y))\}$
$S_k = \{h(s_1, s_2) \mid s_1, s_2 \in S_{k-1}, s_1 \neq s_2\} \cup S_{k-1}, k > 0$.

Alpuente et al. [2014] study anti-unification over $A$, $C$, and $AC$ theories in a more general, order-sorted setting and provide the corresponding algorithms. In [Alpuente et al., 2022], they also consider combining these theories with $U$ in a particular order-sorted signature that guarantees finitary type and completeness of the corresponding algorithms.

Burghardt [2005] proposed a grammar-based approach to the computation of equational generalizations: from a regular tree grammar that describes the congruence classes of the given terms $t_1$ and $t_2$, a regular tree grammar describing a complete set of $E$-generalizations of $t_1$ and $t_2$ is computed. This approach works for equational theories that lead to regular congruence classes. Otherwise, one can use some heuristics to approximate the answer, but completeness is not guaranteed.

Baader [1991] considers anti-unification over so-called commutative theories, a concept covering commutative monoids (ACU), commutative idempotent monoids (ACUI), and Abelian groups. The object set is restricted to terms built using variables and the algebraic operator. Anti-unification over commutative theories in this setting is always unitary.

Results for some theory types are summarized as follows:

- A, C, AC: type $\omega$ [Alpuente et al., 2014];
- $U^1, \{AU\}^1, \{CU\}^1, \{ACU\}^1$: type $\omega$.
- $U^{>1}, \{ACU\}^{>1}$, $\{CU\}^{>1}, \{AU\}^{>1}$, $\{AU\}$; type $0$ (but their linear variants have type $\omega$) [Cerna and Kutsia, 2020c];
- $I, AI, CI$: type $\infty$ [Cerna and Kutsia, 2020b];
- $\{UI\}^{>1}, \{AU\}^{>1}$, $\{CU\}^{>1}, \{ACU\}^{>1}$, semirings: type $0$ [Cerna, 2020];
- Commutative theories: type $1$ [Baader, 1991].

### 3.3 First-Order Clausal Generalization (FOCG)

Clauses are disjunctions of literals (atomic formulas or their negations). Generalization of first-order clauses can be seen as a special case of FOEG, with one ACUI symbol (disjunction) that appears always as the top symbol of the involved expressions.

It is one of the oldest theories for which generalization was studied (see, e.g., [Plotkin, 1970]). Clausal generalization (with various base relations) has been successfully used in relational learning. Newer work uses rigidity functions to construct generalizations and is used for clone detection in logic programs [Yernaux and Vanhoof, 2022b].

An important notion to characterize clausal generalization is $\theta$-subsumption [Plotkin, 1970]: It can be defined by treating disjunction as an $U$ symbol, but a more natural definition considers a clause $L_1 \lor \cdots \lor L_n$ as the set of literals $\{L_1, \ldots, L_n\}$. Then we say the clause $C$ $\theta$-subsumes the clause $D$, written $C \leq_D D$, if there exists a substitution $\theta$ such that $C \theta \subseteq D$ (where the notation $S \theta$ is defined as $\{S \theta \mid s \in S\}$ for a set $S$). The base relation $B$ is $\subseteq$, generalization mappings in $M$ are first-order substitutions, and the preference relation $P$ is the inverse of $\theta$-subsumption $\geq$.

Clausal generalization problem is unitary: a finite set of clauses always possesses a unique lggs up to $\theta$-subsumption equivalence $\equiv_P$. Its size can be exponential in the number of clauses it generalizes.

**Example 3.** [Idestam-Almquist, 1995] Let $D_1 = (p(a) \leftarrow q(a), q(b))$ and $D_2 = (p(b) \leftarrow q(b), q(x))$. Then both $C_1 = (p(y) \leftarrow q(y), q(b))$ and $C_2 = (p(y) \leftarrow q(y), q(b), q(z), q(w))$ are lggs of $D_1$ and $D_2$. It is easy to see that $C_1 \equiv_P C_2$.  

6566
Plotkin generalized the notion of \( \theta \)-subsumption to relative \( \theta \)-subsumption, taking into account background knowledge. Given knowledge \( T \) as a set of clauses and a clause \( C \), we write \( \text{Res}(C, T) = R \) if there exists a resolution derivation of the clause \( R \) from \( T \) using \( C \) exactly once.\(^1\) The notion of relative \( \theta \)-subsumption can be formulated in the following way: A clause \( C \) \( \theta \)-subsumes a clause \( D \) relative to a theory \( T \), denoted \( C \preceq_T D \), iff there exists a clause \( R \) such that \( \text{Res}(C, R) = R \) and \( R \preceq D \). To accommodate this case within our framework, we modify mappings in \( M \) to be the composition of a resolution derivation and substitution application, and use \( P \preceq = \geq_T \). The minimal complete set of relative generalizations of a finite set of clauses can be infinite.

Idestam-Almquist [1995] introduced another variant of clausal generalization, proposing a different base relation: 

\[ T-\text{implication} \Rightarrow T \]

Due to space limitations, we refrain from providing the exact definition. It is a reflexive non-transitive relation taking into account a set \( T \) of ground terms extending generalization under \( \theta \)-subsumption to generalization under a special form of implication. Unlike implication, it is decidable. Note that Plotkin introduced \( \theta \)-subsumption as an incomplete approximation of implication. Idestam-Almquist [1997] lifted relative clausal generalization to \( T \)-implication.


### 3.4 Unranked First-Order Generalization (UFOG)

In unranked languages, symbols do not have a fixed arity. They are often referred to as variadic, polyadic, flexary, or flexible arity symbols. To take advantage of such variadicity, unranked languages contain hedge variables (which stand for hedges: finite, possibly empty sequences of terms) together with individual variables (which stand for single terms). In this section, individual variables are denoted by \( x, y, z \), and hedge variables by \( X, Y, Z \). Terms of the form \( f() \) are written as \( f \). Hedges are usually put in parentheses, but a singleton hedge \( (t) \) is written as \( t \). Substitutions map individual variables to terms and hedge variables to hedges, flattening after application.

**Example 4.** Let \( H = (X, f(X), g(x, Y)) \) be a hedge and \( \sigma = \{ x \mapsto f(a, a), X \mapsto (), Y \mapsto (x, g(a, Z)) \} \) be a substitution where () is the empty hedge. Then \( H \sigma = (f, g(f(a, a), x, g(a, Z))) \).

We provide concrete values for UFOG with respect to the parameters of our general framework in Table 3.

Kutsia et al. [2014] studied unranked generalization and proposed the rigid variant forbidding neighboring hedge variables within generalizations. Moreover, an extra parameter called the rigidity function is used to select a set of common subsequences of top function symbols of hedges to be generalized. The most natural choice for the rigidity function computes the set of longest common subsequences (lcs’s) of its arguments. However, there are other interesting rigidity functions (e.g., lcs’s with the minimal length bound, a single lcs chosen by some criterion, longest common substrings, etc.). The rigid variant is also finitary. Its minimal complete set is denoted by \( \text{mcsg}_R \). Kutsia et al. [2014] describe an algorithm that computes this set. We use the lcs rigidity function below.

**Example 5.** Consider singleton hedges \( H_1 = g(f(a), f(a)) \) and \( H_2 = g(f(a), f(a)) \) for the unrestricted generalization case, \( \text{mcsg}(H_1, H_2) = \{g(f(a), f(a))\} \). Then \( g(f(X), Y) \) is a generalization of \( H_1 \) and \( H_2 \), consider substitutions \( \sigma = \{ X \mapsto (), Y \mapsto a \} \) and \( \sigma = \{ X \mapsto a, Y \mapsto () \} \). Then \( g(f(X), Y) \) is rigidly generalizable.

For a rigid variant, \( \text{mcsg}_R(H_1, H_2) = \{g(f(a), f(a))\} \).

The other two elements contained in the unrestricted \( \text{mcsg} \) are now dropped because they contain hedge variables next to each other, which is forbidden in rigid variants.

**Example 6.** Let \( H_1 = (f(a, a), b, g(f(a), f(a)), f) \) and \( H_2 = (f(b, b), g(f(a), f(a)), f) \). Then \( \text{mcsg}_R(H_1, H_2) = \{f(x, x), g(f(a), f(Y)), (X, f(Y), g(f(a), f(Z)))\} \).

Unranked terms and hedges can be used to model semi-structured documents, program code, execution traces, etc. Yamamoto et al. [2001] investigated unranked anti-unification in the context of inductive reasoning over hedge logic programs. They consider a special case (without individual variables), where hedges do not contain duplicate occurrences of the same hedge variable and any set of sibling arguments contains at most one hedge variable. Such hedges are called simple ones. Note that this problem as well as other related problems such as word generalization [Biere, 1993] or AU-generalization can be also solved by the algorithms from Kutsia et al., 2014.

Anti-unification for unranked first-order terms was generalized to unranked second-order terms [Baumgartner and Kutsia, 2017] and to unranked term-graphs [Baumgartner et al., 2018]. Both problems are finitary and fit into our general framework.

### 3.5 Description Logics

Description logics (DLs) are important formalisms for knowledge representation and reasoning. They are decidable frag-
ments of first-order logic. The basic syntactic building blocks in DLs are concept names (unary predicates), role names (binary predicates), and individual names (constants). Starting from these constructions, complex concepts and roles are built using constructors, which determine the expressive power of the DL. For DLs considered in this section, we show how concept descriptions (denoted by $C$ and $D$) are defined inductively over the sets of concept names $N_C$ and role names $N_R$. Below we provide definitions for the description logics $\mathcal{EL}$, $\mathcal{FEL}$, $\mathcal{ALE}$, and $\mathcal{ALEN}$, where $P \in N_C$ is a primitive concept, $r \in N_R$ is a role name, and $n \in \mathbb{N}$.

$$\mathcal{EL}: \quad C, D: = P \quad | \quad C \cap D \quad | \quad \exists r.C \quad | \quad \forall r.C.$$  

$$\mathcal{FEL}: \quad C, D: = P \quad | \quad C \cap D \quad | \quad \exists r.C \quad | \quad \forall r.C.$$  

$$\mathcal{ALE}: \quad C, D: = P \quad | \quad C \cap D \quad | \quad \exists r.C \quad | \quad \forall r.C \quad | \quad \lnot P \quad | \quad \bot.$$  

$$\mathcal{ALEN}: \quad C, D: = P \quad | \quad C \cap D \quad | \quad \exists r.C \quad | \quad \forall r.C \quad | \quad \lnot P \quad | \quad \bot.$$  

An interpretation $I = (\Delta_I, \cdot)$ consists of a non-empty set $\Delta_I$, called the interpretation domain, and a mapping $\cdot$, called the extension mapping. It maps every concept name $P \in N_C$ to a set $P^I \subseteq \Delta_I$, and every role name $r \in N_R$ to a binary relation $r^I \subseteq \Delta_I \times \Delta_I$. The other concept descriptions are defined as follows:

$$T^I = \Delta_I: = (C \cap D)^I = C^I \cap D^I;$$  

$$(\exists r.C)^I = \{d \in \Delta_I \mid \exists (e, d) \in r^I \wedge e \in C^I \};$$  

$$(\forall r.C)^I = \{d \in \Delta_I \mid \forall (e, d) \in r^I \Rightarrow e \in C^I \};$$  

$$(\lnot P)^I = \{d \in \Delta_I \mid \#\{e \mid (d, e) \in r^I\} = 0\}$$  

where $R \in \{\geq, \leq\}$.

Like in FOCG, subsumption is important for defining the generalization problem. A concept description $C$ is subsumed by $D$, written $C \subseteq D$, if $C^I \subseteq D^I$ holds for all interpretations $I$. (We write $C \equiv D$ if $C$ and $D$ subsume each other.) A concept description $D$ is called a least common subsumer of $C_1$ and $C_2$, if (i) $C_1 \subseteq D$ and $C_2 \subseteq D$ and (ii) if there exists $D'$ such that $C_1 \subseteq D'$ and $C_2 \subseteq D'$, then $D \subseteq D'$.

The problem of computing the least common subsumer of two or more concept descriptions can be seen as a version of the problem of computing generalizations in DLs. It has been studied, e.g., in [Cohen and Hirsh, 1994; Baader et al., 1999; Küsters and Molitor, 2001; Baader et al., 2007].

Example 7 ([Baader et al., 1999]). Assume the DL is $\mathcal{EL}$. $C = P \sqcap \exists r.(P \sqcap Q) \sqcap \exists r.(P \sqcap s.P)$, and $D = \exists r.(P \sqcap r \exists Q)$. Then $\exists r.(\exists r \exists Q) \sqcap \exists r.(\exists r \exists P, T)$ is the least common subsumer of $C$ and $D$.

Example 8 ([Küsters and Molitor, 2001]). Assume the DL is $\mathcal{ALEN}$. $C = \exists r.(P \sqcap A_1) \sqcap \exists r.(P \sqcap A_2) \sqcap \exists s.(\neg P \sqcap A_1) \sqcap \exists r.(Q \sqcap A_3) \sqcap \exists r.(-Q \sqcap A_3) \sqcap (\leq 2 r)$, and $D = (\geq 3 r) \sqcap \forall r.(A_1 \sqcap A_2 \sqcap A_3)$. Then $(\geq 2 r) \sqcap \forall r : (A_1 \sqcap A_3) \sqcap \exists r : (A_1 \sqcap A_2 \sqcap A_3)$ is the least common subsumer of $C$ and $D$.

Table 4 shows how results for $\mathcal{EL}$, $\mathcal{FEL}$, $\mathcal{ALE}$, and $\mathcal{ALEN}$ fit into our framework. Some other results about generalizations in DLs include the computation of the least common subsumer with respect to a background terminology [Baader et al., 2007], computation of the least common subsumer and the most specific concept with respect to a knowledge base [Jung et al., 2020], and anti-unification [Konev and Kutsia, 2016].

4 Higher-Order Generalization

Higher-Order generalization mainly concerns generalization in simply-typed lambda calculus, although it has been studied in other theories of Berendregt’s $\lambda$-cube [Barendregt et al., 2013] and related settings (see [Pfenning, 1991]).

We consider lambda terms defined by the grammar $t ::= x \mid c \mid \lambda x.t \mid \sigma(t)$, where $x$ is a variable and $c$ is a constant. A simple type $\tau$ is either a basic type $\delta$ or a function type $\tau \rightarrow \tau$. We use the standard notions of $\lambda$-calculus such as bound and free variables, subterms, $\alpha$-conversion, $\beta$-reduction, $\eta$-long $\beta$-normal form, etc. (see, e.g., [Barendregt et al., 2013]).

Substitutions are (type-preserving) mappings from variables to lambda terms. They form the set $\mathcal{M}$. In this section, $x, y, z$ are used from bound variables and $X, Y, Z$ for free ones.

4.1 Higher-Order $\alpha\beta\eta$-Generalization (HOG$\alpha\beta\eta$)

Syntactic anti-unification in simply-typed lambda calculus is generalization modulo $\alpha$, $\beta$, $\eta$ rules (i.e., the base relation is equality modulo $\alpha\beta\eta$, which we denote by $\equiv$ in this section). Terms are assumed to be in $\eta$-long $\beta$-normal form. The preference relation is $\preceq$: $t \preceq \tau$ if $s \equiv \tau$ for a substitution $\sigma$. Its inverse is denoted by $\succeq$. Cerna and Buran [2022] show that unrestricted generalization in this theory is nullary:

Example 9. Let $s = \lambda x \lambda y.f(x)$ and $t = \lambda x \lambda y.f(y)$. Then any complete set of generalizations of $s$ and $t$ contains $\succeq$-comparable elements. For instance, if such a set contains a generalization $r = \lambda x \lambda y.f(X(x, y))$, there exists an infinite chain of less and less general generalizations $\sigma \preceq r \sigma \preceq \ldots$ with $\sigma = \{X \mapsto \lambda x \lambda y.X(X(x, y), X(x, y))\}$.

Cerna and Kutsia [2019] proposed a generic framework that accommodates several special unitary variants of generalization in simply-typed lambda calculus. The framework is motivated by two desired properties of generalizations: to maximally keep the top-down common parts of the given terms (top-maximality) and to avoid the nesting of generalization variables (sh looseness). These constraints lead to the top-maximal shallow (TMS) generalization variants that allow some freedom in choosing the subterms occurring under generalization variables. Possible unitary variants are as follows: projective (pr: entire terms), common subterms (cs: maximal common subterms), other cs-variants where common subterms are not necessarily maximal but satisfy restrictions discussed in [Libal and Miller, 2022] such as (relaxed) functions-as-constructors (rfc,fc), and patterns (p). The time complexity of computing pr and p variants is linear in the size of input terms, while for the other cases, it is cubic.

Example 10. For terms $\lambda x.f(h(g(g(x)))), h(g(x)), a$ and $\lambda x.f(g(g(x)), g(x), h(a))$, various top-maximal shallow
4.4 Second-Order Combinator Generalization

Hasker [1995] considers an alternative representation of second-order logic using combinators instead of lambda abstractions (Table 8). Unlike lambda terms, where the application of one term to another is performed via substitution, combinators are special symbols, each associated with a precise axiomatic definition of their effect on the input terms. Note that here substitution concerns term normalization and not the generalization problem. The generalization problems and algorithms from [Hasker, 1995] still require second-order substitutions.

He considers monadic combinators, which take a single argument, and cartesian combinators, which generalize monadic combinators by allowing multiple arguments via a pairing function. Cartesian combinator generalization is nullary as the pairing function can act as a storage for irrelevant constructions. The author addresses this by introducing a concept of relevance resulting in finitary generalization problem.

5 Applications

Typical applications fall into one of the following areas: learning and reasoning, synthesis and exploration, and analysis and repair. Below we briefly discuss the state of the art in these areas and, when possible, the associated type of generalization.

5.1 Learning and Reasoning

Inductive logic programming systems based on inverse entailment, such as ProGolem [Muggleton et al., 2009], Aleph [Srinivasan, 2001] and Progol [Muggleton, 1995] used (relative) $\theta$-subsumption, or variants of it, to search for generalizations of the most specific clause entailing a single example (the bottom clause). The ILP system Popper, developed by Cropper and Morel [2021] uses $\theta$-subsumption-based constraints to iteratively simplify the search space. Recent extensions of this system, such as Hopper [Purgal et al., 2022], and NOPI [Cerna and Cropper, 2023] consider similar techniques over a more expressive hypothesis space.

Several authors have focused on generalization for analogical reasoning. Krummack et al. [2007] use a restricted

<table>
<thead>
<tr>
<th>Generic</th>
<th>Concrete (HOE$\alpha\beta\eta$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}$</td>
<td>The set of simply-typed $\lambda$ terms</td>
</tr>
<tr>
<td>$\mathcal{M}$</td>
<td>Higher-order substitutions</td>
</tr>
<tr>
<td>$B$</td>
<td>$\approx_E$ (equality modulo $\alpha\beta\eta$)</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>$\subseteq$ (more specific, less general modulo $\alpha\beta\eta$)</td>
</tr>
<tr>
<td>$\equiv_p$</td>
<td>Equi-general ($\equiv$ and $\subseteq$ modulo $\alpha\beta\eta$)</td>
</tr>
</tbody>
</table>

Type | Depends on $E$ |
Alg. | Depends on $E$ |

Table 5: Higher-order $\alpha\beta\eta$-generalization.

<table>
<thead>
<tr>
<th>Generic</th>
<th>Concrete (HOEG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}$</td>
<td>The set of simply-typed $\lambda$ terms</td>
</tr>
<tr>
<td>$\mathcal{M}$</td>
<td>Higher-order substitutions</td>
</tr>
<tr>
<td>$B$</td>
<td>$\approx_E$ (equality modulo $\alpha\beta\eta$ and $E$)</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>$\subseteq_E$ (more specific, less general modulo $\alpha\beta\eta$ and $E$)</td>
</tr>
<tr>
<td>$\equiv_p$</td>
<td>Equi-general ($\equiv_E$ and $\subseteq_E$) modulo $\alpha\beta\eta$</td>
</tr>
</tbody>
</table>

Type | Depends on $E$ |
Alg. | Depends on $E$ |

Table 6: Higher-order equational generalization.
5.2 Synthesis and Exploration

The programming by example (pbe) paradigm is an inductive synthesis method concerned with the generation of a program within a domain-specific language (dsl) that generalizes input-output examples [Raza et al., 2014]. Efficient search through the dsl exploits purpose-built generalization methods [Mitchell, 1982]. Foundational work in this area include [Guhe et al., 2011] and [Martinez et al., 2017]. Learning reasoning rules using anti-unification from quasi-natural language sentences is discussed in [Yang and Deng, 2021]. Learning via generalization of linguistic structures has found applications in the development of industrial chatbots [Galitsky, 2019].

6 Future Directions

Although research on anti-unification has a several decades-long history, most of the work in this area was driven by practical applications, and the theory of anti-unification is relatively less developed (in comparison to, e.g., its dual technique of unification). To address this shortcoming, we list some interesting future work directions which, in our opinion, can significantly contribute to improving the state-of-the-art.

- Characterization of anti-unification over equational theories based on the function symbols allowed in problems alongside variables (only equational symbols, equational symbols+free constants, or equational, etc.). This choice might influence e.g., generalization type.
- Developing methods for combining anti-unification algorithms for disjoint equational theories.
- Characterization of equational theories exhibiting similar behavior and properties for generalization problems.
- Studying the influence of the preference relation choice on the type and solution set of generalization problems.
- Studying computational complexity and optimizations.
Acknowledgments

Supported by Czech Science Foundation Grant No. 22-06414L, Austrian Science Fund project P 35530, and Cost Action CA20111 EuroProofNet.

References


