Anti-unification and Generalization: A Survey

David M. Cerna[1*], **Temur Kutsia**[2]
[1]Czech Academy of Sciences Institute of Computer Science (CAS ICS), Prague, Czechia
[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 [Baader and Snyder, 2001]. AU was introduced by Plotkin [1970] and Reynolds [1970] and may be illustrated as follows:

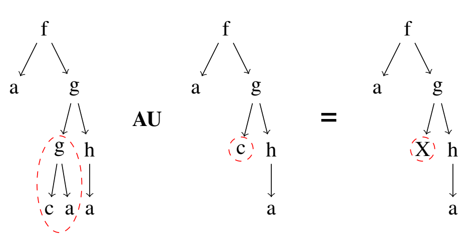

Figure 1: Illustration of anti-unification between two terms.

where $f(a, g(g(c, a), h(a)))$ 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)* [Cropper *et al.*, 2022] approaches exploited the relationship between generalizations to learn logic programs [Muggleton, 1995]. Modern ILP approaches, such as *Popper* [Cropper and Morel, 2021], use this mechanism to simplify the search iteratively. The *programming by example (pbe)* [Gulwani, 2016] paradigm integrates syntactic anti-unification methods to find the *least general* programs satisfying the input examples. Recent work concerning *library learning and compression* [Cao *et al.*, 2023] exploits *equational anti-unification* to find suitable programs efficiently and outperforms *Dreamcoder* [Ellis *et al.*, 2021], 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* [Barwell *et al.*, 2018] 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* [de Sousa *et al.*, 2021], *preventing misconfigurations* [Mehta *et al.*, 2020], and *detecting software clones* [Vanhoof and Yernaux, 2019].

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* [Cao *et al.*, 2023] developed an E-graph anti-unification algorithm motivated solely by the seminal work of Plotkin [1970]. Due to the fragmentary nature of the anti-unification literature, the authors missed relevant work on equational [Burghardt, 2005] and term-graph anti-unification [Baumgartner *et al.*, 2018] 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-

[*]Contact Author

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, $\dots$) in some formal language. Additionally, we consider a class of mappings $\mathcal{M}$ from $\mathcal{O}$ to $\mathcal{O}$. We say that $\mu(\mathsf{O})$ 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 $\mathsf{G} \in \mathcal{O}$ is a *generalization* of the object $\mathsf{O} \in \mathcal{O}$ *with respect to* $\mathcal{B}$ and a class of mappings $\mathcal{M}$ (briefly, $\mathcal{B}_\mathcal{M}$-generalization) if $\mathcal{B}(\mu(\mathsf{G}), \mathsf{O})$ 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 $\mathcal{P}(\mathsf{O}_1, \mathsf{O}_2)$ to indicate that the object O_1 *is preferred over* the object O_2. It induces an equivalence relation $\equiv_\mathcal{P}$: $\mathsf{O}_1 \equiv_\mathcal{P} \mathsf{O}_2$ iff $\mathcal{P}(\mathsf{O}_1, \mathsf{O}_2)$ and $\mathcal{P}(\mathsf{O}_2, \mathsf{O}_1)$.

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 $\mathcal{P}(\mathsf{G}_1, \mathsf{G}_2)$ holds for some G_2, then G_2 is also a $\mathcal{B}_\mathcal{M}$-generalization of O. In other words, if $\mathcal{B}(\mu_1(\mathsf{G}_1), \mathsf{O})$ for some $\mu_1 \in \mathcal{M}$ and $\mathcal{P}(\mathsf{G}_1, \mathsf{G}_2)$, then there should exist $\mu_2 \in \mathcal{M}$ such that $\mathcal{B}(\mu_2(\mathsf{G}_2), \mathsf{O})$.

$$
\begin{tikzcd}
\mathsf{G}_1 \arrow[rr, dashed, "\mathcal{P}"] \arrow[dr, "\mathcal{B}"', "\mu_1"] & & \mathsf{G}_2 \arrow[dl, "\mu_2"', "\mathcal{B}"] \\
& \mathsf{O} &
\end{tikzcd}
$$

Figure 2: Consistency

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* $\mathsf{O}_1, \dots, \mathsf{O}_n$, $n \geq 2$ 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 $\mathcal{P}(\mathsf{G}', \mathsf{G})$, then $\mathsf{G}' \equiv_\mathcal{P} \mathsf{G}$ (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:	Objects $\mathsf{O}_1, \dots, \mathsf{O}_n \in \mathcal{O}$, $n \geq 2$.
Find:	An object $\mathsf{G} \in \mathcal{O}$ that is a most $\mathcal{P}$-preferred common $\mathcal{B}_\mathcal{M}$-generalization of $\mathsf{O}_1, \dots, \mathsf{O}_n$.

This problem may have zero, one, or more solutions. There can be two reasons why it has zero solutions: either the objects $\mathsf{O}_1, \dots, \mathsf{O}_n$ have no common $\mathcal{B}_\mathcal{M}$-generalization at all (i.e, $\mathsf{O}_1, \dots, \mathsf{O}_n$ 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 $\mathsf{O}_1, \dots, \mathsf{O}_n$, $n \geq 2$, if the following properties are satisfied:

- **Soundness:** every $\mathsf{G} \in \mathcal{G}$ is a common $\mathcal{B}_\mathcal{M}$-generalization of $\mathsf{O}_1, \dots, \mathsf{O}_n$, and
- **Completeness:** for each common $\mathcal{B}_\mathcal{M}$-generalization G' of $\mathsf{O}_1, \dots, \mathsf{O}_n$ there exists $\mathsf{G} \in \mathcal{G}$ such that $\mathcal{P}(\mathsf{G}, \mathsf{G}')$.

The set $\mathcal{G}$ is called $\mathcal{P}$*-minimal complete set of common* $\mathcal{B}_\mathcal{M}$*-generalizations* of $\mathsf{O}_1, \dots, \mathsf{O}_n$ and is denoted by $\mathsf{mcsg}_{\mathcal{B}_\mathcal{M}, \mathcal{P}}(\mathsf{O}_1, \dots, \mathsf{O}_n)$ if, in addition, the following holds:

- **Minimality:** no distinct elements of $\mathcal{G}$ are $\mathcal{P}$-comparable: if $\mathsf{G}_1, \mathsf{G}_2 \in \mathsf{G}$ and $\mathcal{P}(\mathsf{G}_1, \mathsf{G}_2)$, then $\mathsf{G}_1 = \mathsf{G}_2$.

Note that the minimality property guarantees that if $\mathsf{G} \in \mathsf{mcsg}_{\mathcal{B}_\mathcal{M},\mathcal{P}}(\mathsf{O}_1, \ldots, \mathsf{O}_n)$, then no G', differing from G, in the $\equiv_\mathcal{P}$-equivalence class of G belongs to this set.

In the notation, we may skip $\mathcal{B}_\mathcal{M}$, $\mathcal{P}$, or both from $\mathsf{mcsg}_{\mathcal{B}_\mathcal{M},\mathcal{P}}$ when it is clear from the context.

Definition 5 (Generalization type)**.** We say that the *type of the* $(\mathcal{B}_\mathcal{M}, \mathcal{P})$*-generalization problem* between the generalizable objects $\mathsf{O}_1, \ldots, \mathsf{O}_n \in \mathcal{O}$ is

- **unitary** (**1**): $\mathsf{mcsg}_{\mathcal{B}_\mathcal{M},\mathcal{P}}(\mathsf{O}_1, \ldots, \mathsf{O}_n)$ is a singleton,
- **finitary** (ω): $\mathsf{mcsg}_{\mathcal{B}_\mathcal{M},\mathcal{P}}(\mathsf{O}_1, \ldots, \mathsf{O}_n)$ is finite and contains at least two elements,
- **infinitary** (∞): $\mathsf{mcsg}_{\mathcal{B}_\mathcal{M},\mathcal{P}}(\mathsf{O}_1, \ldots, \mathsf{O}_n)$ is infinite,
- **nullary** (**0**): $\mathsf{mcsg}_{\mathcal{B}_\mathcal{M},\mathcal{P}}(\mathsf{O}_1, \ldots, \mathsf{O}_n)$ does not exist (i.e., minimality and completeness contradict each other).

The *type of* $(\mathcal{B}_\mathcal{M}, \mathcal{P})$*-generalization over* $\mathcal{O}$ is

- **unitary** (**1**): each $(\mathcal{B}_\mathcal{M}, \mathcal{P})$-generalization problem between generalizable objects from $\mathcal{O}$ is unitary,
- **finitary** (ω): each $(\mathcal{B}_\mathcal{M}, \mathcal{P})$-generalization problem between generalizable objects from $\mathcal{O}$ is unitary or finitary, and there exists a finitary problem,
- **infinitary** (∞): each $(\mathcal{B}_\mathcal{M}, \mathcal{P})$-generalization problem between generalizable objects from $\mathcal{O}$ is unitary, finitary, or infinitary, and there exists an infinitary problem,
- **nullary** (**0**): there exists a nullary $(\mathcal{B}_\mathcal{M}, \mathcal{P})$-generalization problem between generalizable objects from $\mathcal{O}$.

The basic questions to be answered in this context are,

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

If the given objects $\mathsf{O}_1, \ldots, \mathsf{O}_n$ (resp. the desired object G) are restricted to belong to a subset $\mathcal{S} \subseteq \mathcal{O}$, then we talk about an $\mathcal{S}$**-fragment** (resp. $\mathcal{S}$**-variant**) of the generalization problem. It also makes sense to consider an $\mathcal{S}_1$-variant of an $\mathcal{S}_2$-fragment of the problem, where $\mathcal{S}_1$ and $\mathcal{S}_2$ are not necessarily the same.

When $\mathcal{O}$ 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 [Aït-Kaci and Sasaki, 2001; Armengol and Plaza, 2000], for term-graphs [Baumgartner *et al.*, 2018], nominal [Baumgartner *et al.*, 2015; Schmidt-Schauß and Nantes-Sobrinho, 2022] and approximate anti-unification [Aït-Kaci and Pasi, 2020; Kutsia and Pau, 2022; Pau, 2022]. These generalization problems all fit in our general framework.

Generic	Concrete (FOSG)
$\mathcal{O}$	The set of first-order terms
$\mathcal{M}$	First-order substitutions
$\mathcal{B}$	$\doteq$ (syntactic equality)
$\mathcal{P}$	$\succeq$ (more specific, less general): $s \succeq t$ iff $s \doteq t\sigma$ for some σ
$\equiv_\mathcal{P}$	Equi-generality: $\succeq$ and $\preceq$
Type	Unitary
Alg.	[Huet, 1976; Plotkin, 1970; Reynolds, 1970]

Table 1: First-order syntactic generalization.

3 Generalization in First-Order Theories

3.1 First-Order Syntactic Generalization (FOSG)

Plotkin [1970] and Reynolds [1970] introduced FOSG, 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 σ 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 σ. Table 1 specifies the concrete instances of the abstract parameters, and consistency follows from the transitivity of $\succeq$. The relation $\equiv_\mathcal{P}$ 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 $\succeq$-preferred $\doteq$-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*:* $t\sigma_i = s_i$, $i = 1, 2$*. 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* $\succeq$*-preferred ones.*

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

Given a set of axioms E, *the equational 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 equational theory, we say that s and t are equal modulo E and write $s \doteq_E t$.

Generic	Concrete (FOEG)
$\mathcal{O}$	The set of first-order terms
$\mathcal{M}$	First-order substitutions
$\mathcal{B}$	$\doteq_E$ (equality modulo E)
$\mathcal{P}$	$\succeq_E$ (more specific, less general modulo E) $s \succeq_E t$ iff $s \doteq_E t\sigma$ for some σ
$\equiv_{\mathcal{P}}$	Equi-generality modulo E: $\succeq_E$ and $\preceq_E$
Type	Depends on E, fragments, and variants
Alg.	Depends on E, fragments, and variants

Table 2: First-Order equational generalization.

In a theory, we may have several symbols that satisfy the same axiom. For instance, $\mathsf{C}(\{f, g\})$ denotes the theory where f and g are commutative; $\mathsf{AC}(\{f, g\})\mathsf{C}(\{h\})$ denotes the theory where f and g are associative-commutative and h is commutative; $\mathsf{U}(\{(f, e_f), (g, e_g)\})$ 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 $\mathsf{E}_1(S_1)\cdots\mathsf{E}_n(S_n)$, then $S_i \cap S_j = \emptyset$ for each $1 \le i \ne j \le n$.

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^1 stands for $\mathsf{E}(S)$, where the set S contains a single element; $\mathsf{E}^{>1}$ stands for $\mathsf{E}(S)$ where S contains finitely many, but at least two elements. When we write only E, we mean the equational theory $\mathsf{E}(S)$ with a finite set of symbols S that may contain one or more elements.

We can extend this notation to combinations: for instance, $(\mathsf{AU})^{>1}$ 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 $E = \mathsf{AC}(\{f\})$, $s = f(f(a, a), b)$, and $t = f(f(b, b), a)$, then $\mathsf{mcsg}_E(s, t) = \{f(f(x, x), y), f(f(x, a), b)\}$. If we had variables instead of a and b, e.g., if $s = f(f(z, z), v)$ and $t = f(f(v, v), z)$, then $\mathsf{mcsg}_E(s, t) = \{f(f(x, x), y)\}$, because $f(f(x, z), v)$ (the counterpart of $f(f(x, a), b)$) is more general (less preferred) than $f(f(x, x), y)$.

If $E = \mathsf{U}(\{(f, e)\})$, $s = g(f(a, c), a)$, $t = g(c, b)$, then $\mathsf{mcsg}_E(s, t) = \{g(f(x, c), f(y, x)),\ g(f(x, c), f(x, y))\}$. 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 $g(f(x, c), f(y, x))\sigma = g(f(a, c), f(e, a)) \doteq_{\mathsf{U}} s$ and $g(f(x, c), f(y, x))\vartheta = g(f(e, c), f(b, e)) \doteq_{\mathsf{U}} t$.

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

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

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

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 ω [Alpuente *et al.*, 2014];
- U^1, $(\mathsf{AU})^1$, $(\mathsf{CU})^1$, $(\mathsf{ACU})^1$: type ω. $\mathsf{U}^{>1}$, $(\mathsf{ACU})^{>1}$, $(\mathsf{CU})^{>1}$, $(\mathsf{AU})^{>1}$, $(\mathsf{AU})(\mathsf{CU})$: type **0** (but their linear variants have type ω) [Cerna and Kutsia, 2020c];
- I, AI, CI: type ∞ [Cerna and Kutsia, 2020b];
- $(\mathsf{UI})^{>1}$, $(\mathsf{AUI})^{>1}$, $(\mathsf{CUI})^{>1}$, $(\mathsf{ACUI})^{>1}$, 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 only 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 *θ-subsumption* [Plotkin, 1970]: It can be defined by treating disjunction as an ACUI symbol, but a more natural definition considers a clause $L_1 \vee \cdots \vee L_n$ as the set of literals $\{L_1, \ldots, L_n\}$. Then we say the clause C θ-subsumes the clause D, written $C \preceq D$, if there exists a substitution θ 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 $\mathcal{B}$ is $\subseteq$, generalization mappings in $\mathcal{M}$ are first-order substitutions, and the preference relation $\mathcal{P}$ is the inverse of θ-subsumption $\succeq$.

Clausal generalization problem is unitary: a finite set of clauses always possesses a unique lgg up to θ-subsumption equivalence $\equiv_{\mathcal{P}}$. 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 $C_1 \equiv_{\mathcal{P}} C_2$.*

Plotkin generalized the notion of θ-subsumption to *relative θ-subsumption*, taking into account background knowledge. Given knowledge $\mathcal{T}$ as a set of clauses and a clause C, we write $Res(C, \mathcal{T}) = R$ if there exists a resolution derivation of the clause R from $\mathcal{T}$ using C exactly once.[1] The notion of relative θ-subsumption can be formulated in the following way: A clause *C θ-subsumes a clause D relative to a theory $\mathcal{T}$*, denoted $C \preceq_{\mathcal{T}} D$, iff there exists a clause R such that $Res(C, \mathcal{T}) = R$ and $R \preceq D$. To accommodate this case within our framework, we modify mappings in $\mathcal{M}$ to be the composition of a resolution derivation and substitution application, and use $\mathcal{P} = \succeq_{\mathcal{T}}$. 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-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 θ-subsumption to generalization under a special form of implication. Unlike implication, it is decidable. Note that Plotkin introduced θ-subsumption as an incomplete approximation of implication. Idestam-Almquist [1997] lifted relative clausal generalization to T-implication.

Idestam-Almquist [2009] introduced a modification of relative clausal generalization, called asymmetric relative minimal generalization, implementing it in ProGolem. Kuželka *et al.* [2012] studied a bounded version of clausal generalization motivated by practical applications in clause learning. Yernaux and Vanhoof [2022a] consider different base relations for generalizing unordered logic program goals.

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 $\mathsf{mcsg}_{\mathcal{R}}$. Kutsia *et al.* [2014] describe an algorithm that computes this set. We use the lcs rigidity function below.

Generic	Concrete (UFOG)
$\mathcal{O}$	Unranked terms and hedges
$\mathcal{M}$	Substitutions (for terms and for hedges)
$\mathcal{B}$	$\doteq$ (syntactic equality)
$\mathcal{P}$	$\succeq$ (more specific, less general) $s \succeq t$ iff $s \doteq t\sigma$ for some σ.
$\equiv_{\mathcal{P}}$	Equi-generality: $\succeq$ and $\preceq$
Type	Finitary (also for the rigid variant)
Alg.	[Kutsia *et al.*, 2014]

Table 3: Unranked first-order generalization.

Example 5. *Consider singleton hedges $H_1 = g(f(a), f(a))$ and $H_2 = g(f(a), f)$. For the unrestricted generalization case, $\mathsf{mcsg}(H_1, H_2) = \{g(f(a), f(X)), g(f(X, Y), f(X)), g(f(X, Y), f(Y))\}$. To see why, e.g., $g(f(X, Y), f(Y))$ is a generalization of H_1 and H_2, consider substitutions $\sigma = \{X \mapsto (), Y \mapsto a\}$ and $\vartheta = \{X \mapsto a, Y \mapsto ()\}$. Then $g(f(X, Y), f(Y))\sigma = H_1$ and $g(f(X, Y), f(Y))\vartheta = H_2$.*

For a rigid variant, $\mathsf{mcsg}_{\mathcal{R}}(H_1, H_2) = \{g(f(a), f(X))\}$. The other two elements contained in the unrestricted mcsg are now dropped because they contain hedge variables next to each other, which is forbidden in rigid variants.

Example 6. *Let $H_1 = (f(a, a), b, f(c), g(f(a), f(a))$ and $H_2 = (f(b, b), g(f(a), f))$. Then $\mathsf{mcsg}_{\mathcal{R}}(H_1, H_2) = \{(f(x, x), X, g(f(a), f(Y))), (X, f(Y), g(f(a), f(Z))\}$.*

The elements of this set originate from two longest common subsequences of symbols at the top level: In both cases the lcs is f followed by g, but in the former we match the top symbols of $f(a, a)$ and $g(f(a), f(a))$ from H_1 to top symbols of H_2, while in the latter we match $f(c)$ and $g(f(a), f(a))$ from H_1.

For the rigidity function computing longest common substrings, $\mathsf{mcsg}_{\mathcal{R}}(H_1, H_2) = \{(X, f(Y), g(f(a), f(Z)))\}$.

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-

[1] In Plotkin's original definition, the derivation uses C at most once. Here we follow the exposition from [Idestam-Almquist, 1997].

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 $\mathcal{EL}$, $\mathcal{FLE}$, $\mathcal{ALE}$, and $\mathcal{ALEN}$, where $P \in N_C$ is a primitive concept, $r \in N_R$ is a role name, and $n \in \mathbb{N}$.

$$\begin{aligned}
\mathcal{EL}:&\quad C, D := P \mid \top \mid C \sqcap D \mid \exists r.C.\\
\mathcal{FLE}:&\quad C, D := P \mid \top \mid C \sqcap D \mid \exists r.C \mid \forall r.C.\\
\mathcal{ALE}:&\quad C, D := P \mid \top \mid C \sqcap D \mid \exists r.C \mid \forall r.C \mid \neg P \mid \bot.\\
\mathcal{ALEN}:&\quad C, D := P \mid \top \mid C \sqcap D \mid \exists r.C \mid \forall r.C \mid \neg P \mid \bot\\
&\quad \mid (\geq n\, r) \mid (\leq n\, r)
\end{aligned}$$

An interpretation $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ consists of a non-empty set $\Delta^\mathcal{I}$, called the interpretation domain, and a mapping $\cdot^\mathcal{I}$, called the extension mapping. It maps every concept name $P \in N_C$ to a set $P^\mathcal{I} \subseteq \Delta_\mathcal{I}$, and every role name $r \in N_R$ to a binary relation $r^\mathcal{I} \subseteq \Delta_\mathcal{I} \times \Delta_\mathcal{I}$. The other concept descriptions are defined as follows: $\top^\mathcal{I} = \Delta_\mathcal{I}$; $(C \sqcap D)^I = C^\mathcal{I} \cap D^\mathcal{I}$; $(\exists r.C)^\mathcal{I} = \{d \in \Delta_\mathcal{I} \mid \exists e.\, (d, e) \in r^\mathcal{I} \wedge e \in C^\mathcal{I}\}$; $(\forall r.C)^\mathcal{I} = \{d \in \Delta_\mathcal{I} \mid \forall e.\, (d, e) \in r^\mathcal{I} \Rightarrow e \in C^\mathcal{I}\}$; $(\mathbf{R}\, n\, r)^\mathcal{I} = \{d \in \Delta_\mathcal{I} \mid \#\{e \mid (d, e) \in r^\mathcal{I}\}\, \mathbf{R}\, n\}$ where $\mathbf{R} \in \{\geq, \leq\}$.

Like in FOCG, subsumption is important for defining the generalization problem. A concept description C *is subsumed by* D, written $C \sqsubseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$. (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) $C_1 \sqsubseteq D$ and $C_2 \sqsubseteq D$ and (ii) if there exists D' such that $C_1 \sqsubseteq D'$ and $C_2 \sqsubseteq D'$, then $D \sqsubseteq D'$.

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 $\mathcal{EL}$, $C = P \sqcap \exists r.(\exists r.(P \sqcap Q) \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.P)$, and $D = \exists r.(P \sqcap \exists r.P \sqcap \exists s.Q)$. Then $\exists r.(\exists r.P \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.\top)$ is the least common subsumer of C and D.*

Example 8 ([Küsters and Molitor, 2001])**.** *Assume the DL is $\mathcal{ALEN}$, $C = \exists r.(P \sqcap A_1) \sqcap \exists r.(P \sqcap A_2) \sqcap \exists r.(\neg P \sqcap A_1) \sqcap \exists r.(Q \sqcap A_3) \sqcap \exists r.(\neg Q \sqcap A_3) \sqcap (\leq 2\, r)$, 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 $\mathcal{EL}$, $\mathcal{FLE}$, $\mathcal{ALE}$, 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 λ-cube [Barendregt *et al.*, 2013] and related settings (see [Pfenning, 1991]).

Generic	Concrete (DL)
$\mathcal{O}$	Concept descriptions
$\mathcal{M}$	Contains only the identity mapping
$\mathcal{B}$	$\sqsupseteq$
$\mathcal{P}$	$\sqsubseteq$
$\equiv_\mathcal{P}$	$\equiv$: $\sqsubseteq$ and $\sqsupseteq$
Type	Unitary for all four DLs
Alg.	[Baader *et al.*, 1999] for $\mathcal{EL}$, $\mathcal{FLE}$, $\mathcal{ALE}$, [Küsters and Molitor, 2001] for $\mathcal{ALEN}$

Table 4: Generalization (least common subsumer) in DLs $\mathcal{EL}$, $\mathcal{FLE}$, $\mathcal{ALE}$, and $\mathcal{ALEN}$.

We consider lambda terms defined by the grammar $t ::= x \mid c \mid \lambda x.t \mid (t\, t)$, where x is a variable and c is a constant. A simple type τ is either a basic type δ or a function type $\tau \rightarrow \tau$. We use the standard notions of λ-calculus such as bound and free variables, subterms, α-conversion, β-reduction, η-long β-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 ($\mathbf{HOG}_{\alpha\beta\eta}$)

Syntactic anti-unification in simply-typed lambda calculus is generalization modulo α, β, η rules (i.e., the base relation is equality modulo $\alpha\beta\eta$, which we denote by $\approx$ in this section). Terms are assumed to be in η-long β-normal form. The preference relation is $\succsim$: $s \succsim t$ iff $s \approx t\sigma$ for a substitution σ. Its inverse is denoted by $\precsim$. 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 $\precsim$-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 $r\sigma \precsim r\sigma\sigma \precsim \ldots$ 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 (shallowness). 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*

Generic	Concrete (HOG$_{\alpha\beta\eta}$)
$\mathcal{O}$	The set of simply-typed λ terms
$\mathcal{M}$	Higher-order substitutions
$\mathcal{B}$	$\approx$ (equality modulo $\alpha\beta\eta$)
$\mathcal{P}$	$\succsim$ (more specific, less general modulo $\alpha\beta\eta$) $s \succsim t$ iff $s \approx t\sigma$ for a substitution σ.
$\equiv_{\mathcal{P}}$	Equi-general ($\succsim$ and $\precsim$) modulo $\alpha\beta\eta$
Type	$\mathbf{0}$, general [Cerna and Buran, 2022] $\mathbf{1}$, tms variant [Cerna and Kutsia, 2019]
Alg.	tms variant [Cerna and Kutsia, 2019], patterns [Baumgartner *et al.*, 2017]

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

Generic	Concrete (HOEG)
$\mathcal{O}$	The set of simply-typed λ terms
$\mathcal{M}$	Higher-order substitutions
$\mathcal{B}$	$\approx_E$ (equality modulo $\alpha\beta\eta$ and E)
$\mathcal{P}$	$\succsim_E$ (more specific, less general modulo $\alpha\beta\eta$ and E) $s \succsim_E t$ iff $s \approx_E t\sigma$ for a substitution σ.
$\equiv_{\mathcal{P}}$	Equi-general ($\succsim_E$ and $\precsim_E$) modulo $\alpha\beta\eta$
Type	Depends on E [Cerna and Kutsia, 2020a]
Alg.	Depends on E [Cerna and Kutsia, 2020a]

Table 6: Higher-order equational generalization.

lggs are ***pr-lgg:*** $\lambda x.f(X(h(g(g(x))), g(g(x))), X(h(g(x)), g(x)), X(a, h(a)))$, ***cs-lgg:*** $\lambda x.f(X(g(g(X(g(x)), Z(a)))$, ***rfc-lgg:*** $\lambda x.f(X(g(g(x))), X(g(x)), Z)$, ***fc-lgg:*** $\lambda x.f(X(g(x)), Y(g(x)), Z)$, *and* ***p-lgg:*** $\lambda x.f(X(x), Y(x), Z)$.

Table 5 relates HOG$_{\alpha\beta\eta}$ to the general framework.

The linear variant of $\alpha\beta\eta$-Generalization over higher-order patterns was used by [Pientka, 2009] to develop a higher-order term-indexing algorithm based on substitution trees. Insertion into the substitution requires computing the lgg of a given higher-order pattern and a higher-order pattern already in the substitution tree. Feng and Muggleton [1992] consider $\alpha\beta\delta_0\eta$-Generalization over a fragment of simply-typed lambda terms they refer to as λM, where δ_0 denotes an additional decomposition rule for constructions similar to *if-then-else*. Similar to top-maximal shallow lambda terms, λM allows constants within the arguments to generalization variables. Due to space constraints, we refrain from discussing in detail [Pfenning, 1991] work on anti-unification in the calculus of constructions, where he describes an algorithm for the pattern variant. This work fits our general framework by adjusting the parameters used for $\alpha\beta\eta$-Generalization over higher-order patterns.

4.2 Higher-Order Equational Generalization

Cerna and Kutsia [2020a] studied the pattern variant of higher-order equational generalization (HOEG) in simply-typed lambda calculus involving A, C, U axioms and their combinations (Table 6). In addition, they investigated fragments for which certain optimal generalizations may be computed fast. It was shown that pattern HOEG in A, C, AC theories is finitary. The same is true for the linear pattern variant of HOEG in A, C, U theories and their combinations.

The generalization problem for the considered fragments is unitary when only optimal solutions are considered. Optimality means that the solution should be at least as good as the $\alpha\beta\eta$-lgg. Fragments allowing fast computation of optimal solutions (in linear, quadratic, or cubic time) were identified.

4.3 Polymorphic Higher-Order Generalization

Lu *et al.* [2000] consider generalization within the *polymorphic lambda calculus*, typically referred to as $\lambda 2$ (Table 7). Unlike $\alpha\beta\eta$-Generalization presented above, terms are not required to be of the same type to be generalizable. While similar holds regarding [Pfenning, 1991], Lu *et al.* [2000] do not restrict themselves to the pattern variant but instead restrict the preference relation. They use a restricted form of the *application order*, i.e. $s \succsim t$ iff there exists terms and types $r_1, \ldots, r_n$ such that $sr_1 \cdots r_n \approx t$, in other words, $sr_1 \cdots r_n$ β-reduces to t. They restrict $r_1, \ldots, r_n$ to subterms of t and introduce *variable-freezing* to deal with bound variable order. Mappings are also based on β-reduction.

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) θ-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 θ-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. Krumnack *et al.* [2007] use a restricted

Generic	Concrete (PHOG)
$\mathcal{O}$	$\lambda 2$ terms
$\mathcal{M}$	Based on β-reduction
$\mathcal{B}$	$\approx$ (equality modulo $\alpha\beta\eta$)
$\mathcal{P}$	$\gtrsim_{SF}$ (application ordering restricted subterms and modulo *variable-freezing*)
$\equiv_{\mathcal{P}}$	$\gtrsim_{SF}$ and $\lesssim_{SF}$
Type	$\mathbf{1}$ [Lu *et al.*, 2000]
Alg.	[Lu *et al.*, 2000]

Table 7: Polymorphic higher-order generalization.

Generic	Concrete (SOCG)
$\mathcal{O}$	Combinators
$\mathcal{M}$	Substitutions
$\mathcal{B}$	$\approx$ (equality modulo α and combinator reduction)
$\mathcal{P}$	$\gtrsim$ (more specific modulo α and combinator reduction) $s \gtrsim t$ iff $s \approx t\sigma$.
$\equiv_{\mathcal{P}}$	$\gtrsim$ and $\lesssim$
Type	Monadic: ω, Cartesian: $\mathbf{0}$, Relevant: ω [Hasker, 1995]
Alg.	[Hasker, 1995]

Table 8: Second-order combinator generalization.

form of higher-order generalization to develop useful analogies. Weller and Schmid [2006] use the equational generalization method introduced by Burghardt [2005] to solve proportional analogies, i.e. "*A is to B as C is to ?*". Sotoudeh and Thakur [2020] discuss generalization as an tool for analogical reasoning about programs. Generalization is used in *case-based reasoning* [Ontañón and Plaza, 2007] literature as a method to encode coverage of cases by a prediction.

Related research concerning *concept blending* and *mathematical reasoning* builds upon *Heuristic-Driven Theory Projection*, using a form of higher-order anti-unification [Schwering *et al.*, 2009]. Example works in this area include [Guhe *et al.*, 2011] and [Martínez *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].

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 [Gulwani, 2011] and [Polozov and Gulwani, 2015]. Recent developments include [Dong *et al.*, 2022], where the authors specifically reference unranked methods for synthesis within *robotic process automation*, and [Feser *et al.*, 2015], where functional transformations of data structures are synthesized. An earlier tool on inductive synthesis of functional programs is IGOR II [Hofmann, 2010]. It is based on [Kitzelmann and Schmid, 2006] and exploits the basic syntactic generalization methods [Plotkin, 1970]. Johansson *et al.* [2011] use a form of generalization for conjecture synthesis.

Babble [Cao *et al.*, 2023] is a method for theory exploration and compression exploiting FOEG and term-graph representations to find functions that compress the representation of other functions in a library. Bowers *et al.* [2023] focus on the learning aspects of the problem. Singher and Itzhaky [2021] use generalizing templates as part of the synthesis process.

5.3 Analysis and Repair

Bulychev and Minea [2008] introduced anti-unification as a method to detect clones in software and repositories (see also [Bulychev *et al.*, 2009]). This research was further developed for specific use cases: Li and Thompson [2010] investigated clone detection in Erlang, and Yernaux and Vanhoof [2022b] studied clone detection in constraint logic programs.

Sinha [2008] and Arusoaie and Lucanu [2022], use anti-unification to implement efficient *symbolic execution*, a type of software analysis . Hasker [1995] used SOCG to develop a derivation replay approach to automate some aspects of programming through templating. Related to derivation replay is the work of Barwell *et al.* [2018] concerning *parallel recursion scheme detection*. Maude [Clavel *et al.*, 2002] is a declarative programming language useful for software verification tasks. It has been extended by *order-sorted*, *equational*, and *syntactic* anti-unification methods.

As discussed by Winter *et al.* [2022], recent investigations exploit anti-unification to provide program repair and bug detection capabilities. Sakkas *et al.* [2020] use variants of unranked hedge anti-unification as a templating mechanism for providing repairs based on type errors. This approach is also taken by the authors of *Getafix*, [Bader *et al.*, 2019], and REVISAR, [de Sousa *et al.*, 2021]. *Rex*, developed by Mehta *et al.* [2020], takes a similar approach for repairing misconfigured services, while [Siddiq *et al.*, 2021] uses unranked hedge anti-unification to detect and repair SQL injection vulnerabilities. Zhang *et al.* [2022] use generalization techniques to develop edit templates from edit sequences in repositories.

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

[Aït-Kaci and Pasi, 2020] Hassan Aït-Kaci and Gabriella Pasi. Fuzzy lattice operations on first-order terms over signatures with similar constructors: A constraint-based approach. *Fuzzy Sets and Systems*, 391:1–46, 2020.

[Aït-Kaci and Sasaki, 2001] Hassan Aït-Kaci and Yutaka Sasaki. An axiomatic approach to feature term generalization. In *EMCL*. Springer, 2001.

[Alpuente *et al.*, 2014] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014.

[Alpuente *et al.*, 2022] María Alpuente, Santiago Escobar, José Meseguer, and Julia Sapiña. Order-sorted equational generalization algorithm revisited. *Ann. Math. Artif. Intell.*, 90(5):499–522, 2022.

[Armengol and Plaza, 2000] Eva Armengol and Enric Plaza. Bottom-up induction of feature terms. *Mach. Learn.*, 41(3):259–294, 2000.

[Arusoaie and Lucanu, 2022] Andrei Arusoaie and Dorel Lucanu. Proof-carrying parameters in certified symbolic execution: The case study of antiunification. In *FROM*, EPTCS, 2022.

[Baader and Snyder, 2001] Franz Baader and Wayne Snyder. Unification theory. In *Handbook of Automated Reasoning*. Elsevier, 2001.

[Baader *et al.*, 1999] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *IJCAI*. Morgan Kaufmann, 1999.

[Baader *et al.*, 2007] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Appl. Log.*, 5(3):392–420, 2007.

[Baader, 1991] Franz Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In *RTA*, LNCS, 1991.

[Bader *et al.*, 2019] Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. Getafix: learning to fix bugs automatically. *Proc. ACM Program. Lang.*, 3(OOPSLA), 2019.

[Barendregt *et al.*, 2013] Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

[Barwell *et al.*, 2018] Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Gener. Comput. Syst.*, 79:669–686, 2018.

[Baumgartner and Kutsia, 2017] Alexander Baumgartner and Temur Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017.

[Baumgartner *et al.*, 2015] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal anti-unification. In *RTA*, LIPIcs, 2015.

[Baumgartner *et al.*, 2017] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reason.*, 58(2):293–310, 2017.

[Baumgartner *et al.*, 2018] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Term-graph anti-unification. In *FSCD*, LIPIcs, 2018.

[Biere, 1993] Armin Biere. Normalisation, unification and generalisation in free monoids. Master's thesis, University of Karlsruhe, 1993.

[Bowers *et al.*, 2023] Matthew Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-down synthesis for library learning. *Proc. ACM Program. Lang.*, 7(POPL), 2023.

[Bulychev and Minea, 2008] Peter Bulychev and Marius Minea. Duplicate code detection using anti-unification. In *SYRCOSE*, 2008.

[Bulychev *et al.*, 2009] Peter E. Bulychev, Egor V. Kostylev, and Vladimir A. Zakharov. Anti-unification algorithms and their applications in program analysis. In *PSI*, LNCS, 2009.

[Burghardt, 2005] Jochen Burghardt. E-generalization using grammars. *Artif. Intell.*, 165(1):1–35, 2005.

[Cao *et al.*, 2023] David Cao, Rose Kunkel, Chandrakana Nandi, Max Willsey, Zachary Tatlock, and Nadia Polikarpova. Babble: Learning better abstractions with e-graphs and anti-unification. *Proc. ACM Program. Lang.*, 7(POPL), 2023.

[Cerna and Buran, 2022] David M. Cerna and Michal Buran. One or nothing: Anti-unification over the simply-typed lambda calculus. *CoRR*, abs/2207.08918, 2022.

[Cerna and Cropper, 2023] David M. Cerna and Andrew Cropper. Generalisation through negation and predicate invention. *CoRR*, abs/2301.07629, 2023.

[Cerna and Kutsia, 2019] David M. Cerna and Temur Kutsia. A generic framework for higher-order generalizations. In *FSCD*, LIPIcs, 2019.

[Cerna and Kutsia, 2020a] David M. Cerna and Temur Kutsia. Higher-order pattern generalization modulo equational theories. *Math. Struct. Comput. Sci.*, 30(6):627–663, 2020.

[Cerna and Kutsia, 2020b] David M. Cerna and Temur Kutsia. Idempotent anti-unification. *ACM Trans. Comput. Log.*, 21(2):10:1–10:32, 2020.

[Cerna and Kutsia, 2020c] David M. Cerna and Temur Kutsia. Unital anti-unification: Type and algorithms. In *FSCD*, LIPIcs, 2020.

[Cerna, 2020] David M. Cerna. Anti-unification and the theory of semirings. *Theor. Comput. Sci.*, 848:133–139, 2020.

[Clavel *et al.*, 2002] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Jose F. Quesada. Maude: specification and programming in rewriting logic. *Theor. Comput. Sci.*, 285(2):187–243, 2002.

[Cohen and Hirsh, 1994] William W. Cohen and Haym Hirsh. Learning the classic description logic: Theoretical and experimental results. In *KR*. Morgan Kaufmann, 1994.

[Cropper and Morel, 2021] Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Mach. Learn.*, 110(4):801–856, 2021.

[Cropper *et al.*, 2022] Andrew Cropper, Sebastijan Dumancic, Richard Evans, and Stephen H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022.

[de Sousa *et al.*, 2021] Reudismam Rolim de Sousa, Gustavo Soares, Rohit Gheyi, Titus Barik, and Loris D'Antoni. Learning quick fixes from code repositories. In *SBES*. ACM, 2021.

[Dong *et al.*, 2022] Rui Dong, Zhicheng Huang, Ian Iong Lam, Yan Chen, and Xinyu Wang. Webrobot: Web robotic process automation using interactive programming-by-demonstration. In *PLDI*. ACM, 2022.

[Ellis *et al.*, 2021] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *PLDI*. ACM, 2021.

[Feng and Muggleton, 1992] Cao Feng and Stephen H. Muggleton. Towards inductive generalization in higher order logic. In *ML*. Morgan Kaufmann, 1992.

[Feser *et al.*, 2015] John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *PLDI*. ACM, 2015.

[Galitsky, 2019] Boris Galitsky. *Developing Enterprise Chatbots - Learning Linguistic Structures*. Springer, 2019.

[Guhe *et al.*, 2011] Markus Guhe, Alison Pease, Alan Smaill, Maricarmen Martinez, Martin Schmidt, Helmar Gust, Kai-Uwe Kühnberger, and Ulf Krumnack. A computational account of conceptual blending in basic mathematics. *Cognitive Systems Research*, 12(3):249–265, 2011.

[Gulwani, 2011] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*. ACM, 2011.

[Gulwani, 2016] Sumit Gulwani. Programming by examples - and its applications in data wrangling. In *NATO Science for Peace and Security Series*, volume 45. IOS Press, 2016.

[Hasker, 1995] Robert W. Hasker. *The Replay Of Program Derivations*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[Hofmann, 2010] Martin Hofmann. Igor2 - an analytical inductive functional programming system: Tool demo. In *PEPM*. ACM, 2010.

[Huet, 1976] Gerard Huet. Résolution d'Équations dans des langages d'ordre $1, 2, \ldots, \omega$. These d'État, Université de Paris VII, 1976.

[Idestam-Almquist, 1995] Peter Idestam-Almquist. Generalization of clauses under implication. *J. Artif. Int. Res.*, 3(1):467–489, 1995.

[Idestam-Almquist, 1997] Peter Idestam-Almquist. Generalization of clauses relative to a theory. *Mach. Learn.*, 26(2-3):213–226, 1997.

[Johansson *et al.*, 2011] Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *J. Autom. Reason.*, 47(3):251–289, 2011.

[Jung *et al.*, 2020] Jean Christoph Jung, Carsten Lutz, and Frank Wolter. Least general generalizations in description logic: Verification and existence. In *AAAI*. AAAI Press, 2020.

[Kitzelmann and Schmid, 2006] Emanuel Kitzelmann and Ute Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *J. Mach. Learn. Res.*, 7:429–454, 2006.

[Konev and Kutsia, 2016] Boris Konev and Temur Kutsia. Anti-unification of concepts in description logic EL. In *KR*. AAAI Press, 2016.

[Krumnack *et al.*, 2007] Ulf Krumnack, Angela Schwering, Helmar Gust, and Kai-Uwe Kühnberger. Restricted higher-order anti-unification for analogy making. In *AI*, LNCS, 2007.

[Küsters and Molitor, 2001] Ralf Küsters and Ralf Molitor. Computing least common subsumers in ALEN. In *IJCAI*. Morgan Kaufmann, 2001.

[Kutsia and Pau, 2022] Temur Kutsia and Cleo Pau. A framework for approximate generalization in quantitative theories. In *IJCAR*, LNCS, 2022.

[Kutsia *et al.*, 2014] Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reason.*, 52(2):155–190, 2014.

[Kuzelka *et al.*, 2012] Ondrej Kuzelka, Andrea Szabóová, and Filip Zelezný. Bounded least general generalization. In *ILP*, LNCS, 2012.

[Li and Thompson, 2010] Huiqing Li and Simon J. Thompson. Similar code detection and elimination for erlang programs. In *PADL*, LNCS, 2010.

[Libal and Miller, 2022] Tomer Libal and Dale Miller. Functions-as-constructors higher-order unification: extended pattern unification. *Ann. Math. Artif. Intell.*, 90(5):455–479, 2022.

[Lu *et al.*, 2000] Jianguo Lu, John Mylopoulos, Masateru Harao, and Masami Hagiyab. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

[Martínez *et al.*, 2017] Maricarmen Martínez, Ahmed M. H. Abdel-Fattah, Ulf Krumnack, Danny Gómez-Ramírez, Alan Smaill, Tarek Richard Besold, Alison Pease, Martin Schmidt, Markus Guhe, and Kai-Uwe Kühnberger. Theory blending: extended algorithmic aspects and examples. *Ann. Math. Artif. Intell.*, 80(1):65–89, 2017.

[Mehta *et al.*, 2020] Sonu Mehta, Ranjita Bhagwan, Rahul Kumar, Chetan Bansal, Chandra Shekhar Maddila, Balasubramanyan Ashok, Sumit Asthana, Christian Bird, and Aditya Kumar. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *USENIX*. USENIX Association, 2020.

[Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[Muggleton *et al.*, 2009] Stephen H. Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-Nezhad. ProGolem: A system based on relative minimal generalisation. In *ILP*, LNCS, 2009.

[Muggleton, 1995] Stephen H. Muggleton. Inverse entailment and Progol. *New Gener. Comput.*, 13(3&4):245–286, 1995.

[Ontañón and Plaza, 2007] Santiago Ontañón and Enric Plaza. Case-based learning from proactive communication. In *IJCAI*. ijcai.org, 2007.

[Pau, 2022] Cleo Pau. *Symbolic Techniques for Approximate Reasoning*. PhD thesis, RISC, Johannes Kepler University Linz, 2022.

[Pfenning, 1991] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *LICS*. IEEE Computer Society, 1991.

[Pientka, 2009] Brigitte Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Log.*, 11(1):6:1–6:40, 2009.

[Plotkin, 1970] Gordon D. Plotkin. A note on inductive generalization. *Machine Intell.*, 5(1):153–163, 1970.

[Polozov and Gulwani, 2015] Oleksandr Polozov and Sumit Gulwani. Flashmeta: A framework for inductive program synthesis. *SIGPLAN Not.*, 50(10):107–126, 2015.

[Purgal *et al.*, 2022] Stanislaw J. Purgal, David M. Cerna, and Cezary Kaliszyk. Learning higher-order logic programs from failures. In *IJCAI*. ijcai.org, 2022.

[Raza *et al.*, 2014] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. Programming by example using least general generalizations. In *AAAI*. AAAI Press, 2014.

[Reynolds, 1970] John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intell.*, 5(1):135–151, 1970.

[Sakkas *et al.*, 2020] Georgios Sakkas, Madeline Endres, Benjamin Cosman, Westley Weimer, and Ranjit Jhala. Type error feedback via analytic program repair. In *PLDI*. ACM, 2020.

[Schmidt-Schauß and Nantes-Sobrinho, 2022] Manfred Schmidt-Schauß and Daniele Nantes-Sobrinho. Nominal anti-unification with atom-variables. In *FSCD*, LIPIcs, 2022.

[Schwering *et al.*, 2009] Angela Schwering, Ulf Krumnack, Kai-Uwe Kühnberger, and Helmar Gust. Syntactic principles of heuristic-driven theory projection. *Cognitive Systems Research*, 10(3):251–269, 2009.

[Siddiq *et al.*, 2021] Mohammed Latif Siddiq, Md. Rezwanur Rahman Jahin, Mohammad Rafid Ul Islam, Rifat Shahriyar, and Anindya Iqbal. SQLIFIX: Learning based approach to fix SQL injection vulnerabilities in source code. In *SANER*. IEEE, 2021.

[Singher and Itzhaky, 2021] Eytan Singher and Shachar Itzhaky. Theory exploration powered by deductive synthesis. In *CAV*, LNCS, 2021.

[Sinha, 2008] Nishant Sinha. Symbolic program analysis using term rewriting and generalization. In *FMCAD*. IEEE, 2008.

[Sotoudeh and Thakur, 2020] Matthew Sotoudeh and Aditya V. Thakur. Analogy-making as a core primitive in the software engineering toolbox. In *Onward!* ACM, 2020.

[Srinivasan, 2001] Ashwin Srinivasan. The ALEPH manual. *Machine Learning at the Computing Laboratory, Oxford University*, 2001.

[Vanhoof and Yernaux, 2019] Wim Vanhoof and Gonzague Yernaux. Generalization-driven semantic clone detection in CLP. In *LOPSTR*, LNCS, 2019.

[Weller and Schmid, 2006] Stephan Weller and Ute Schmid. Solving proportional analogies by E -generalization. In *KI*, LNCS, 2006.

[Winter *et al.*, 2022] Emily Rowan Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Óskar Haraldsson, John R. Woodward, Serkan Kirbas, Etienne Windels, Olayori McBello, Abdurahman Atakishiyev, Kevin Kells, and Matthew W. Pagano. Towards developer-centered automatic program repair: findings from Bloomberg. In *ESEC/FSE*. ACM, 2022.

[Yamamoto *et al.*, 2001] Akihiro Yamamoto, Kimihito Ito, Akira Ishino, and Hiroki Arimura. Modelling semi-structured documents with hedges for deduction and induction. In *ILP*, LNCS, 2001.

[Yang and Deng, 2021] Kaiyu Yang and Jia Deng. Learning symbolic rules for reasoning in quasi-natural language. *CoRR*, abs/2111.12038, 2021.

[Yernaux and Vanhoof, 2022a] Gonzague Yernaux and Wim Vanhoof. Anti-unification of unordered goals. In *EACSL*, LIPIcs, 2022.

[Yernaux and Vanhoof, 2022b] Gonzague Yernaux and Wim Vanhoof. On detecting semantic clones in constraint logic programs. In *IWSC*. IEEE, 2022.

[Zhang *et al.*, 2022] Yuhao Zhang, Yasharth Bajpai, Priyanshu Gupta, Ameya Ketkar, Miltiadis Allamanis, Titus Barik, Sumit Gulwani, Arjun Radhakrishna, Mohammad Raza, Gustavo Soares, and Ashish Tiwari. Overwatch: Learning patterns in code edit sequences. *Proc. ACM Program. Lang.*, 6(OOPSLA), 2022.