J Autom Reasoning (2014) 52:155–190
DOI 10.1007/s10817-013-9285-6

Anti-unification for Unranked Terms and Hedges

Temur Kutsia · Jordi Levy · Mateu Villaret

Received: 26 March 2012 / Accepted: 2 April 2013 / Published online: 19 April 2013

Abstract We study anti-unification for unranked terms and hedges that may contain term and hedge variables. The anti-unification problem of two hedges $\tilde{s}_1$ and $\tilde{s}_2$ is concerned with finding their generalization, a hedge $\tilde{q}$ such that both $\tilde{s}_1$ and $\tilde{s}_2$ are instances of $\tilde{q}$ under some substitutions. Hedge variables help to fill in gaps in generalizations, while term variables abstract single (sub)terms with different top function symbols. First, we design a complete and minimal algorithm to compute least general generalizations. Then, we improve the efficiency of the algorithm by restricting possible alternatives permitted in the generalizations. The restrictions are imposed with the help of a rigidity function, which is a parameter in the improved algorithm and selects certain common subsequences from the hedges to be generalized. The obtained *rigid anti-unification* algorithm is further made more precise by permitting combination of hedge and term variables in generalizations. Finally, we indicate a possible application of the algorithm in software engineering.

This research has been partially supported by the Spanish Ministerio de Economa y Competitividad under the projects HeLo (TIN2012-33042) and TASSAT (TIN2010-20967-C04-01), by the EC FP6 for Integrated Infrastructures Initiatives under the project SCIEnce (contract No. 026133), by the Austrian Science Fund (FWF) under the project SToUT (P 24087-N18), and by the Generalitat de Catalunya under the grant AGAUR 2009-SGR-1434.

T. Kutsia (✉)
Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria
e-mail: kutsia@risc.jku.at

J. Levy
Artificial Intelligence Research Institute (IIIA), Spanish Council for Scientific Research (CSIC), Barcelona, Spain
e-mail: levy@iiia.csic.es

M. Villaret
Departament d'Informàtica i Matemàtica Aplicada (IMA), Universitat de Girona (UdG), Girona, Spain
e-mail: villaret@ima.udg.edu

Keywords Anti-unification · Generalization · Unranked terms · Hedges · Software clones

1 Introduction

The anti-unification problem of two terms t_1 and t_2 is concerned with finding their generalization, a term t such that both t_1 and t_2 are instances of t under some substitutions. The problem has a trivial solution, a fresh variable, which is the most general generalization of the given terms. Interesting generalizations are the least general ones. The purpose of anti-unification algorithms is to compute such least general generalizations. Plotkin [31] and Reynolds [32] pioneered research on anti-unification, designing generalization algorithms for ranked terms (where function symbols have a fixed arity) in the syntactic case. Since then, a number of algorithms and their modifications have been developed, addressing the problem in various theories (e.g., [2, 3, 5, 10, 16, 29]) and from different application points of view (e.g., [4, 9, 13, 19, 27, 28, 35]). Applications come from the areas such as reasoning by analogy, machine learning, inductive logic programming, software engineering, program synthesis, analysis, transformation, and verification, just to name a few.

Unranked terms differ from the ranked ones by not having fixed arity for function symbols. Hedges are finite sequences of such terms. They are flexible structures, popular in representing semistructured data. To take the advantage of variadicity, unranked terms and hedges use two kinds of variables: term variables, which stand for a single term, and hedge variables, which stand for hedges. Solving techniques over unranked terms and hedges mostly address unification and matching problems, see, e.g., [12, 20–25]. Anti-unification for these structures practically has not been studied. The only exception, to the best of our knowledge, is Yamamoto et al. [37], which addresses a special case of so called simple hedges. Related, but essentially different problems of anti-unification for feature terms have been studied by Plaza [30], Armengol and Plaza [4], Aït-Kaci and Sasaki [1].

We address this shortcoming, presenting algorithms to compute least general generalizations (lggs, in short) for unranked terms/hedges. Hedge variables help to fill in gaps in generalizations, while term variables abstract single (sub)terms with different top function symbols. Unlike ranked anti-unification, where there is always one least general generalization, in the unranked case there can be finitely many lggs. First, we design an algorithm that computes a minimal complete set of unranked generalizations. But this set can be exponentially large compared to the size of the input. Therefore, it is interesting to find a smaller, yet practically useful set of lggs. We go ahead to specify such generalizations and give algorithms to compute them. These are the problems considered in this paper, whose main contributions can be summarized as follows:

- An algorithm that computes a complete set of lggs for the given hedges is developed in Section 3.
- Efficiency of the algorithm is improved by restricting it to compute a minimal and complete set of only certain generalizations. The restrictions are imposed with the help of a rigidity function, which is a parameter in the improved algorithm. It gives also the name to those special generalizations: they are called rigid generalizations. A rigidity function works on two strings composed of top function

symbols of terms in the input hedges, returning their common subsequence that satisfies an additional property specified by the function. With the help of this resulting string, at each step, the rigidity function dictates the algorithm which subsequence of the given hedges is to be (structurally) retained in the generalization. These are the terms whose top symbols appear in the string returned by the rigidity function. In this way we get a more efficient, yet pretty general algorithm to compute a complete set of rigid lggs. It is described in Section 4.

- Rigid generalizations use only hedge variables to abstract differences between hedges. It can be made more precise if we permit (sequences of) term variables to abstract differences between sequences of terms of the same length, retaining hedge variables for abstracting differences between hedges of distinct length. This refined algorithm is described in Section 5.
- Some known anti-unification problems can be modeled as a special case of unranked anti-unification. Several such examples are considered in Section 6.
- Termination, soundness, and completeness theorems are proved for all the above mentioned generalization algorithms.
- Our results open a possibility to address, by means of unranked anti-unification, the problems of detecting duplicate pieces (clones) of software code, refactoring, clustering XML documents, detecting similarities and differences in them, etc. Using rigid anti-unification for these tasks has several advantages, including appealing combination of fast textual and precise structural techniques. We discuss a possible application in software clone detection in Section 7.

This paper is an extended and improved version of Kutsia et al. [26].

1.1 Related Work

A special case of anti-unification for unranked terms has been studied by Yamamoto et al. [37] in the context of applying inductive reasoning to semi-structured documents. This special case deals with a restricted class of so called simple hedges. They are characterized as being linear (no duplicated occurrences of hedge variables) and for each subterm $f(s_1, \ldots, s_n)$ occurring in the hedge at any depth, there is at most one hedge variable among $s_1, \ldots, s_n$. Term variables are not permitted. The proposed anti-unification algorithm accepts two simple hedges as input and produces one of the least general simple generalizations in cubic time. In Section 4 we show how simple anti-unification can be modeled as a special case of rigid anti-unification, choosing an appropriate rigidity function.

Word anti-unification is a special case of hedge anti-unification, when all the terms in the hedge are constants. Unrestricted search for word generalizations suffers from the same problem as the minimal and complete hedge generalization algorithm: Too much non-determinism and, consequently, too large search space. Therefore, there were attempts to compute only special generalizations. Cicekli and Cicekli [11] are interested only in generalizations that contain the longest common subsequence of the original words provided that this longest common subsequence contains all the letters that the given words have in common. Biere [7] considers only ε-free generalizations. They are characterized by the following property: If the word w_g is a generalization of the word w such that $\sigma(w_g) = w$ for a substitution σ, then σ does not map any variable to the empty word ε. That is, the size of the generalization

is never strictly larger than the size of the words it generalizes. In Section 4 we show how to model these algorithms in our framework.

The rules for associative anti-unification with the unit element (AU anti-unification) considered in Alpuente et al. [2] can be directly simulated in the minimal and complete unranked anti-unification algorithm. Conversely, to simulate unranked anti-unification rules in AU anti-unification, one needs to introduce sorts to distinguish between term variables and hedge variables (and their instantiations). Rigid anti-unification and AU anti-unification are not comparable.

One can not compare feature anti-unification [1, 4, 30] and unranked anti-unification either. The only relationship between these problems is that in both of them, the arity of symbols is not fixed. Otherwise, feature terms do not require fixed ordering of arguments (which is the case for unranked expressions) and its functors represent ordered sorts (in contrast to unsorted setting for unranked terms). The usage of variables in unranked generalization does not have its counterpart in feature anti-unification.

2 Preliminaries

Given pairwise disjoint countable sets of unranked function symbols F (symbols without fixed arity), term variables $\mathscr{V}_\mathsf{T}$, and hedge variables $\mathscr{V}_\mathsf{H}$, we define *unranked terms* (terms in short) and *hedges* (sequences of terms or hedge variables) over F and $\mathsf{V} = \mathscr{V}_\mathsf{T} \cup \mathscr{V}_\mathsf{H}$ by the following grammar:

$$
\begin{array}{lll}
t ::= x \mid f(\tilde{s}) & & \text{(terms)} \\
s ::= t \mid X & & \text{(terms or hedge variables)} \\
\tilde{s} ::= s_1, \ldots, s_n & & \text{(hedges)}
\end{array}
$$

where $x \in \mathscr{V}_\mathsf{T}$, $f \in \mathsf{F}$, $X \in \mathscr{V}_\mathsf{H}$, and $n \geq 0$. With this definition, terms are singleton hedges. Not all singleton hedges are terms: some may be hedge variables.

The set of terms (respectively, hedges) over F and V is denoted by $\mathsf{T}(\mathsf{F}, \mathscr{V}_\mathsf{T}, \mathscr{V}_\mathsf{H})$ (respectively, $\mathsf{H}(\mathsf{F}, \mathscr{V}_\mathsf{T}, \mathscr{V}_\mathsf{H})$). We use the letters f, g, h, a, b, c, and d for function symbols, x, y, and z for term variables, X, Y, Z, U, and V for hedge variables, χ for a term variable or a hedge variable, t, l, and r for terms, s and q for a hedge variable or a term, and $\tilde{s}$ and $\tilde{q}$ for hedges. The empty hedge is denoted by ε. The terms of the form $a(\varepsilon)$ are written as just a.

Note that ε is just a notation for the empty hedge introduced for readability. It does not appear next to other hedges because concatenation of a hedge $\tilde{s}$ with the empty hedge is $\tilde{s}$ itself. Hence, we assume the implicit removal of the symbol ε in such cases. We write it explicitly only when we want to talk about the empty hedge.

If $\tilde{s} = s_1, \ldots, s_n$ and $\tilde{s}' = s'_1, \ldots, s'_m$, then we write $\tilde{s}, \tilde{s}'$ for $s_1, \ldots, s_n, s'_1, \ldots, s'_m$. We denote by $\tilde{s}|_i$ the ith element of the sequence $\tilde{s}$. We denote by $\tilde{s}|_i^j$, where $i < j$, the subsequence between positions i and j, which does not include any of them, i.e., the subsequence $\tilde{s}|_{i+1}, \ldots, \tilde{s}|_{j-1}$. The length of a sequence $\tilde{s}$, denoted $|\tilde{s}|$, is the number of elements in it.

The *size* of a term t is the number of occurrences of symbols (from $\mathsf{F} \cup \mathsf{V}$) in it and is denoted by $size(t)$. We denote by $var(t)$ the *set of variables* of a term t. These

definitions are generalized for any syntactic object. Note that the size of the empty hedge is 0.

A *substitution* is a mapping from hedge variables to hedges and from term variables to terms, which is the identity almost everywhere. We will use the traditional finite set representation of substitutions, writing, e.g., $\{x \mapsto f(a), X \mapsto \varepsilon, Y \mapsto x, g(a, Z)\}$ for the substitution that maps every variable to itself except x, X, and Y that are mapped respectively to $f(a)$, ε, and $x, g(a, Z)$. The lower case Greek letters are used to denote substitutions, with the exception of the identity substitution for which we write *Id*.

Substitutions can be applied to terms and hedges using the congruences

$$\sigma(f(s_1, \ldots, s_n)) = f(\sigma(s_1), \ldots, \sigma(s_n)), \qquad \sigma(s_1, \ldots, s_n) = \sigma(s_1), \ldots, \sigma(s_n).$$

We call $\sigma(s)$ and $\sigma(\tilde{s})$ the *instances* of respectively s and $\tilde{s}$ and use postfix notation to denote them, writing $s\sigma$ and $\tilde{s}\sigma$. (Note that if σ maps some variable X to ε, then X is simply removed from the instance.) We also say that $\tilde{s}$ is *more general* than $\tilde{q}$ if $\tilde{q}$ is an instance of $\tilde{s}$ and denote this fact by $\tilde{s} \preceq \tilde{q}$. If $\tilde{s} \preceq \tilde{q}$ and $\tilde{q} \preceq \tilde{s}$, then we write $\tilde{s} \simeq \tilde{q}$. If $\tilde{s} \preceq \tilde{q}$ and $\tilde{s} \not\simeq \tilde{q}$, then we say that $\tilde{s}$ is *strictly more general* than $\tilde{q}$ and write $\tilde{s} \prec \tilde{q}$. The set $dom(\sigma) = \{\chi \in \mathsf{V} \mid \chi\sigma \neq \chi\}$ is called the *domain* of σ.

The *composition* of two substitutions σ and ϑ, written as $\sigma\vartheta$, is defined as the composition of two mappings: We have $s(\sigma\vartheta) = (s\sigma)\vartheta$ for all s. A substitution σ_1 is *more general* than σ_2 with respect to a set of variables $\mathscr{X} \subseteq \mathsf{V}$, written $\sigma_1 \preceq_{\mathscr{X}} \sigma_2$, if there exists ϑ such that $\chi\sigma_1\vartheta = \chi\sigma_2$, for each $\chi \in \mathscr{X}$. The relations $\simeq$ and $\prec$ are extended to substitutions: $\sigma_1 \simeq_{\mathscr{X}} \sigma_2$ means $\sigma_1 \preceq_{\mathscr{X}} \sigma_2$ and $\sigma_2 \preceq_{\mathscr{X}} \sigma_1$, and $\sigma_1 \prec_{\mathscr{X}} \sigma_2$ means $\sigma_1 \preceq_{\mathscr{X}} \sigma_2$ and $\sigma_1 \not\simeq_{\mathscr{X}} \sigma_2$.

The *top symbol* of a term is defined as $top(x) = x$ for any variable x, and $top(f(\tilde{s})) = f$ for any term $f(\tilde{s})$. We extend this notion to hedges, defining it as the sequence of symbols as follows: $top(X, \tilde{s}) = X top(\tilde{s})$ and $top(t, \tilde{s}) = top(t)top(\tilde{s})$ for any hedge variable X, term t, and hedge $\tilde{s}$. We write these sequences as words, e.g., $top(f(a), a, X, x) = faXx$. $top(\varepsilon)$ is just the empty word which we also refer to by ε. The letter w will be used for the words of top symbols.

A hedge $\tilde{s}$ is called a *generalization* or an *anti-instance* of two hedges $\tilde{s}_1$ and $\tilde{s}_2$ if $\tilde{s} \preceq \tilde{s}_1$ and $\tilde{s} \preceq \tilde{s}_2$. That means, there exist substitutions σ_1 and σ_2 such that $\tilde{s}_1 = \tilde{s}\sigma_1$ and $\tilde{s}_2 = \tilde{s}\sigma_2$. We say that a hedge $\tilde{s}$ is a *least general generalization* (lgg in short), aka a *most specific anti-instance*, of $\tilde{s}_1$ and $\tilde{s}_2$ if $\tilde{s}$ is a generalization of $\tilde{s}_1$ and $\tilde{s}_2$ and there is no generalization $\tilde{q}$ of $\tilde{s}_1$ and $\tilde{s}_2$ that satisfies $\tilde{s} \prec \tilde{q}$. That means, there are no generalizations of $\tilde{s}_1$ and $\tilde{s}_2$ that are strictly less general than their least general generalization.

An *anti-unification problem* (or equation), AUP in short, is a triple $\chi : \tilde{s}_1 \triangleq \tilde{s}_2$, where χ does not occur in $\tilde{s}_1$ and $\tilde{s}_2$. Intuitively, χ is a variable that stands for the most general generalization of $\tilde{s}_1$ and $\tilde{s}_2$. An *anti-unifier* of $\chi : \tilde{s}_1 \triangleq \tilde{s}_2$ is a substitution σ such that $dom(\sigma) \subseteq \{\chi\}$ and $\chi\sigma$ is a generalization of both $\tilde{s}_1$ and $\tilde{s}_2$. An anti-unifier σ of an AUP $\chi : \tilde{s}_1 \triangleq \tilde{s}_2$ is *least general (or most specific)* if there is no anti-unifier ϑ of the same problem that satisfies $\sigma \prec_{\mathscr{X}} \vartheta$. Obviously, if σ is a least general anti-unifier of an AUP $\chi : \tilde{s}_1 \triangleq \tilde{s}_2$, then $\chi\sigma$ is a least general generalization of $\tilde{s}_1$ and $\tilde{s}_2$.

A *complete set of generalizations* of two hedges $\tilde{s}_1$ and $\tilde{s}_2$ is a set G of hedges that satisfies the properties:

Soundness: Each $\tilde{q} \in G$ is a generalization of both $\tilde{s}_1$ and $\tilde{s}_2$.

Completeness: For each generalization $\tilde{s}$ of $\tilde{s}_1$ and $\tilde{s}_2$, there exists $\tilde{q} \in G$ such that $\tilde{s} \preceq \tilde{q}$.

G is a *minimal complete set of generalizations* of $\tilde{s}_1$ and $\tilde{s}_2$ if it, in addition to soundness and completeness, satisfies also the following property:

Minimality: For each $\tilde{q}_1, \tilde{q}_2 \in G$, if $\tilde{q}_1 \preceq \tilde{q}_2$ then $\tilde{q}_1 = \tilde{q}_2$.

Lemma 2.1 *For any three hedges $\tilde{s}_1$, $\tilde{s}_2$ and $\tilde{q}$, and any pair of substitutions σ_1 and σ_2 satisfying $\tilde{s}_1 = \tilde{q}\sigma_1$ and $\tilde{s}_2 = \tilde{q}\sigma_2$, if $size(\tilde{q}) > size(\tilde{s}_1) + size(\tilde{s}_2)$ then there exists at least one hedge variable X occurring in $\tilde{q}$ such that $X\sigma_1 = X\sigma_2 = \varepsilon$.*

Proof Since substitutions cannot remove function symbols, the hedge $\tilde{q}$ can not contain more function symbols than $\tilde{s}_1$ and $\tilde{s}_2$ do. Since they cannot substitute term variables by empty terms, $\tilde{q}$ also can not contain more term variables than there are subterms in $\tilde{s}_1$ and $\tilde{s}_2$. Violation of any of these conditions would forbid $\tilde{s}_1$ or $\tilde{s}_2$ to be an instance of $\tilde{q}$. Therefore, if $size(\tilde{q}) > size(\tilde{s}_1) + size(\tilde{s}_2)$, then $\tilde{q}$ must contain some hedge variable that is mapped to ε by both σ_1 and σ_2. □

Lemma 2.2 *For any hedges $\tilde{s}_1$ and $\tilde{s}_2$ there exists their minimal complete set of generalizations. This set is finite and unique modulo $\simeq$.*

Proof For classical first-order anti-unification this property is trivial, because instantiation does not decrease the size of terms. This means that anti-unifiers of two terms are smaller than each of those terms, hence finite modulo variable renaming. For hedges the property is not so simple to prove because instantiating a hedge variable by ε, the size of a term may decrease. However, by Lemma 2.1 we have that for any anti-unifier $\tilde{q}$ of $\tilde{s}_1$ and $\tilde{s}_2$ with $size(\tilde{q}) > size(\tilde{s}_1) + size(\tilde{s}_2)$ there exists another anti-unifier less general than $\tilde{q}$ (that we can obtain by replacing those extra hedge variables in $\tilde{q}$ by ε). The set of anti-unifiers smaller than the sum of the sizes of both hedges is a complete set of anti-unifiers. This set contains all least general anti-unifiers, contains finitely many distinct elements modulo $\simeq$. If we remove from this set all non least general anti-unifiers, we get a minimal complete set of generalizations that is unique modulo $\simeq$. □

We denote the minimal complete set of generalizations of $\tilde{s}_1$ and $\tilde{s}_2$ by $mcg(\tilde{s}_1 \triangleq \tilde{s}_2)$. Its elements are lggs of $\tilde{s}_1$ and $\tilde{s}_2$.

Like unification problems, anti-unification problems may be classified as unitary (if minimal complete sets of generalizations always exist and are singletons), finitary (if they always exist, are finite, and the problem is not unitary), infinitary (if they always exist and may be infinite), and nullary (if they may not exist). Hence, Lemma 2.2 implies that hedge anti-unification is finitary.

An anti-unification problem always has an anti-unifier. The empty substitution is a trivial example that represents the most general generalization. Our goal is to compute less general generalizations. In the next section, we design an algorithm that computes (the set of anti-unifiers that represents) the *mcg* of a given AUP. It requires some care to properly address the issues that arise because of hedge variables in generalizations.

Quiz 1: *Given two hedges* $\tilde{s} = f(a),\ f(a)$ *and* $\tilde{q} = f(a),\ f$, *what is the set* $mcg(\tilde{s} \triangleq \tilde{q})$? *Hint: There are three elements in this set.*

Without loss of generality, below we assume that the hedges to be generalized are variable disjoint.

3 Complete and Minimal Algorithm

We present our anti-unification algorithm as a rule-based algorithm that works on triples $A; S; \sigma$. Here A is a set of AUPs of the form $\{X_1 : \tilde{s}_1 \triangleq \tilde{q}_1, \ldots, X_n : \tilde{s}_n \triangleq \tilde{q}_n\}$ where each X_i occurs in the problem only once, S is a set of already solved anti-unification equations (the store), and σ is a substitution (computed so far).[1] We call such a triple a *system*. The rules transform systems into systems:

T-H: **Trivial Hedge**
$\{X : \varepsilon \triangleq \varepsilon\} \cup A;\ S;\ \sigma \Longrightarrow A;\ S;\ \sigma\{X \mapsto \varepsilon\}$.

Dec-T: **Decomposition for Terms**
$\{X : f(\tilde{s}) \triangleq f(\tilde{q})\} \cup A;\ S;\ \sigma \Longrightarrow \{Y : \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma\{X \mapsto f(Y)\}$
where Y is a fresh variable.

Dec1-H: **Decomposition 1 for Hedges**
$\{X : s, \tilde{s} \triangleq q, \tilde{q}\} \cup A;\ S;\ \sigma \Longrightarrow \{Y : s \triangleq q,\ Z : \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma\{X \mapsto Y, Z\}$,
where $U : s, \tilde{s} \triangleq q, \tilde{q} \notin S$ for all $U \in \mathcal{V}_{\mathsf{H}}$, the variables Y and Z are fresh, and $\tilde{s} \neq \varepsilon$ or $\tilde{q} \neq \varepsilon$.

Dec2-H: **Decomposition 2 for Hedges**
$\{X : s, \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma \Longrightarrow \{Y : s \triangleq \varepsilon,\ Z : \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma\{X \mapsto Y, Z\}$,
where $\chi : s, \tilde{s} \triangleq \tilde{q} \notin S$ for all $\chi \in \mathsf{V}$, the variables Y and Z are fresh, and $\tilde{s} \neq \varepsilon$ or $\tilde{q} \neq \varepsilon$.

Dec3-H: **Decomposition 3 for Hedges**
$\{X : \tilde{s} \triangleq q, \tilde{q}\} \cup A;\ S;\ \sigma \Longrightarrow \{Y : \varepsilon \triangleq q,\ Z : \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma\{X \mapsto Y, Z\}$,
where $\chi : \tilde{s} \triangleq q, \tilde{q} \notin S$ for all $\chi \in \mathsf{V}$, the variables Y and Z are fresh, and $\tilde{s} \neq \varepsilon$ or $\tilde{q} \neq \varepsilon$.

Sol1-H: **Solve 1 for Hedges**
$\{X : s \triangleq \varepsilon\} \cup A;\ S;\ \sigma \Longrightarrow A;\ \{X : s \triangleq \varepsilon\} \cup S;\ \sigma$, if $Y : s \triangleq \varepsilon \notin S$ for all Y.

Sol2-H: **Solve 2 for Hedges**
$\{X : \varepsilon \triangleq q\} \cup A;\ S;\ \sigma \Longrightarrow A;\ \{X : \varepsilon \triangleq q\} \cup S;\ \sigma$,
if $Y : \varepsilon \triangleq q \notin S$ for all Y.

Sol3-H: **Solve 3 for Hedges**
$\{X : s \triangleq q\} \cup A;\ S;\ \sigma \Longrightarrow A;\ \{X : s \triangleq q\} \cup S;\ \sigma$,
if $s \neq q, s \in \mathcal{V}_{\mathsf{H}}$ or $q \in \mathcal{V}_{\mathsf{H}}$, and $Y : s \triangleq q \notin S$ for all Y.

[1] Such a representation was first proposed in Alpuente et al. [2] for equational anti-unification.

Sol-T: **Solve for Terms**
$\{X : l \triangleq r\} \cup A;\ S;\ \sigma \Longrightarrow A;\ \{y : l \triangleq r\} \cup S;\ \sigma\{X \mapsto y\}$,
if $top(l) \neq top(r)$, $\chi : l \triangleq r \notin S$ for all χ, and y is fresh.

Rec: **Recover**
$\{X : \tilde{s} \triangleq \tilde{q}\} \cup A;\ \{\chi : \tilde{s} \triangleq \tilde{q}\} \cup S;\ \sigma \Longrightarrow A;\ \{\chi : \tilde{s} \triangleq \tilde{q}\} \cup S;\ \sigma\{X \mapsto \chi\}$.

The idea of the store is to keep track of already solved AUPs in order to generalize the same pair of hedges with the same variable, as it is illustrated in the Rec rule: The already solved AUP $\chi : \tilde{s} \triangleq \tilde{q}$ from the store helps to reuse χ instead of X as a generalization of $\tilde{s}$ and $\tilde{q}$. This is important, since we aim at computing lggs.

In the condition of Dec1-H we use a hedge variable U while in Dec2-H and Dec3-H in the same role χ appears. The reason is that in Dec1-H, the hedge $s, \tilde{s}$ or the hedge $q, \tilde{q}$ is not a term and, hence, we can not have a term variable in place of U. On the other hand, in Dec2-H and Dec3-H it can happen that the AUP in the condition is between terms with χ being a term variable.

One can notice that we do not have a rule for AUPs like $X : x \triangleq x$. This is because we assume the hedges to be generalized are variable disjoint and, hence, such problems do not appear.

To compute generalizations for hedges $\tilde{s}$ and $\tilde{q}$, the procedure starts with $\{X : \tilde{s} \triangleq \tilde{q}\}; \emptyset; Id$ where X is a fresh hedge variable and applies the rules on each selected anti-unification equation in all possible ways. We denote this procedure by $\mathfrak{G}$.

To show that the process terminates, we define a complexity measure of the triple $A; S; \sigma$ as a multiset $M(A) := \{size(\tilde{s} \triangleq \tilde{q}) + 1 \mid X : \tilde{s} \triangleq \tilde{q} \in A\}$. We order complexity measures by the multiset extension $>_m$ of the standard ordering on natural numbers. It is easy to check that each rule in $\mathfrak{G}$ strictly reduces the measure: If $A_1; S_1; \sigma_1 \Longrightarrow A_2; S_2; \sigma_2$ in $\mathfrak{G}$, then $M(A_1) >_m M(A_2)$. It immediately implies termination:

Theorem 3.1 *The procedure $\mathfrak{G}$ terminates for any input.*

Hence, starting from $\{X : \tilde{s} \triangleq \tilde{q}\}; \emptyset; Id$, each sequence of transformations by $\mathfrak{G}$ necessarily terminates with a triple of the form $\emptyset; S; \sigma$.

Example 3.2 Let $f(g(a,a), g(b,b), f(g(a), g(a)))$ and $f(g(a,a), f(g(a), g))$ be two given terms. We illustrate how the algorithm $\mathfrak{G}$ computes one of their generalizations, the term $f(Z, g(x,x), f(g(a), g(X)))$:

$$\{X_0 : f(g(a,a), g(b,b), f(g(a), g(a))) \triangleq f(g(a,a), f(g(a), g))\}; \emptyset; Id \Longrightarrow_{\text{Dec-T}}$$
$$\{X_1 : g(a,a), g(b,b), f(g(a), g(a)) \triangleq g(a,a), f(g(a), g)\};$$
$$\emptyset; \{X_0 \mapsto f(X_1)\} \Longrightarrow_{\text{Dec2-H}}$$
$$\{Z : g(a,a) \triangleq \varepsilon, X_2 : g(b,b), f(g(a), g(a)) \triangleq g(a,a), f(g(a), g)\};$$
$$\emptyset; \{X_0 \mapsto f(Z, X_2), \ldots\} \Longrightarrow_{\text{Sol1-H}}$$
$$\{X_2 : g(b,b), f(g(a), g(a)) \triangleq g(a,a), f(g(a), g)\};$$
$$\{Z : g(a,a) \triangleq \varepsilon\}; \{X_0 \mapsto f(Z, X_2), \ldots\} \Longrightarrow_{\text{Dec1-H}}$$

$$\{X_3 : g(b,b) \triangleq g(a,a),\ X_4 :\ f(g(a),g(a)) \triangleq\ f(g(a),g)\};$$
$$\{Z : g(a,a) \triangleq \varepsilon\};\ \{X_0 \mapsto\ f(Z, X_3, X_4), \ldots\} \Longrightarrow_{\text{Dec-T}}$$
$$\{X_5 : b, b \triangleq a, a,\ X_4 :\ f(g(a),g(a)) \triangleq\ f(g(a),g)\};$$
$$\{Z : g(a,a) \triangleq \varepsilon\};\ \{X_0 \mapsto\ f(Z, g(X_5), X_4), \ldots\} \Longrightarrow_{\text{Dec1-H}}$$
$$\{X_6 : b \triangleq a,\ X_7 : b \triangleq a,\ X_4 :\ f(g(a),g(a)) \triangleq\ f(g(a),g)\};$$
$$\{Z : g(a,a) \triangleq \varepsilon\};\ \{X_0 \mapsto\ f(Z, g(X_6, X_7), X_4), \ldots\} \Longrightarrow_{\text{Sol-T}}$$
$$\{X_7 : b \triangleq a,\ X_4 :\ f(g(a),g(a)) \triangleq\ f(g(a),g)\};$$
$$\{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};\ \{X_0 \mapsto\ f(Z, g(x, X_7), X_4), \ldots\} \Longrightarrow_{\text{Rec}}$$
$$\{X_4 :\ f(g(a),g(a)) \triangleq\ f(g(a),g)\};$$
$$\{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};\ \{X_0 \mapsto\ f(Z, g(x,x), X_4), \ldots\} \Longrightarrow_{\text{Dec-T}}$$
$$\{X_8 : g(a), g(a) \triangleq g(a), g\};$$
$$\{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};\ \{X_0 \mapsto\ f(Z, g(x,x),\ f(X_8)), \ldots\} \Longrightarrow_{\text{Dec1-H}}$$
$$\{X_9 : g(a) \triangleq g(a),\ X_{10} : g(a) \triangleq g\};$$
$$\{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};\ \{X_0 \mapsto\ f(Z, g(x,x),\ f(X_9, X_{10})), \ldots\} \Longrightarrow_{\text{Dec-T}}$$
$$\{X_{11} : a \triangleq a,\ X_{10} : g(a) \triangleq g\};\ \{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};$$
$$\{X_0 \mapsto\ f(Z, g(x,x),\ f(g(X_{11}), X_{10})), \ldots\} \Longrightarrow_{\text{Dec-T}}$$
$$\{X_{12} : \varepsilon \triangleq \varepsilon,\ X_{10} : g(a) \triangleq g\};\ \{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};$$
$$\{X_0 \mapsto\ f(Z, g(x,x),\ f(g(a(X_{12})), X_{10})), \ldots\} \Longrightarrow_{\text{T-H}}$$
$$\{X_{10} : g(a) \triangleq g\};\ \{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};$$
$$\{X_0 \mapsto\ f(Z, g(x,x),\ f(g(a), X_{10})), \ldots\} \Longrightarrow_{\text{Dec-T}}$$
$$\{X : a \triangleq \varepsilon\};\ \{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a\};$$
$$\{X_0 \mapsto\ f(Z, g(x,x),\ f(g(a), g(X))), \ldots\} \Longrightarrow_{\text{Sol1-H}}$$
$$\emptyset;\ \{Z : g(a,a) \triangleq \varepsilon, x : b \triangleq a, X : a \triangleq \varepsilon\};$$
$$\{X_0 \mapsto\ f(Z, g(x,x),\ f(g(a), g(X))), \ldots\}$$

The last substitution maps X_0 to $f(Z, g(x,x),\ f(g(a), g(X)))$. One can see that this term, indeed, generalizes the given terms. Hence, the substitution $\{X_0 \mapsto f(Z, g(x,x),\ f(g(a), g(X)))\}$ is their anti-unifier.

Notice that from the final system one can also get the substitutions that show how the original terms can be obtained from the computed generalization. These substitutions can be extracted from the store: $\sigma_1 = \{Z \mapsto g(a,a),\ x \mapsto b, X \mapsto a\}$ with $f(Z, g(x,x),\ f(g(a),\ g(X)))\sigma_1 = f(g(a,a), g(b,b),\ f(g(a), g(a)))$ and $\sigma_2 = \{Z \mapsto \varepsilon,\ x \mapsto a, X \mapsto \varepsilon\}$ with $f(Z,\ g(x,x),\ f(g(a),\ g(X)))\sigma_2 = f(g(a,a),\ f(g(a), g))$. In this way, we can say that the store gives us the difference between the input terms.

Soundness of $\mathfrak{G}$ is not hard to establish:

Theorem 3.3 (Soundness of $\mathfrak{G}$) *If* $\{X : \tilde{s} \triangleq \tilde{q}\}; \emptyset; Id \Longrightarrow^* \emptyset; S; \sigma$ *is a derivation in* $\mathfrak{G}$, *then* $X\sigma \preceq \tilde{s}$ *and* $X\sigma \preceq \tilde{q}$.

Proof The theorem follows from the straightforward fact that if $X\sigma \preceq \tilde{s}$ and $X\sigma \preceq \tilde{q}$ and $\{X : \tilde{s} \triangleq \tilde{q}\} \cup A; S; \sigma \Longrightarrow A', S', \sigma'$ is a transformation step in $\mathfrak{G}$, then $X\sigma' \preceq \tilde{s}$ and $X\sigma' \preceq \tilde{q}$. □

If $\{X : \tilde{s} \triangleq \tilde{q}\}; \emptyset; Id \Longrightarrow^* \emptyset; S; \sigma$ is a derivation in $\mathfrak{G}$, then we say that

- σ is a *substitution computed by* $\mathfrak{G}$ *for* $X : \tilde{s} \triangleq \tilde{q}$;
- the restriction of σ on X, denoted by $\sigma|_X$, is an *anti-unifier of* $X : \tilde{s} \triangleq \tilde{q}$ *computed by* $\mathfrak{G}$;
- the hedge $X\sigma$ is a *generalization of* $\tilde{s}$ *and* $\tilde{q}$ *computed by* $\mathfrak{G}$.

The proof of completeness of the algorithm requires auxiliary definitions and lemmas. We start generalizing the notion of anti-unifier for sets of equations.

Definition 3.4 A set of AUPs is a set $A = \{\chi_1 : \tilde{s}_1 \triangleq \tilde{q}_1 \ldots, \chi_n : \tilde{s}_n \triangleq \tilde{q}_n\}$, where variables do not occur more than once. We define the set of generalization variables $gvar(A) = \{\chi_1, \ldots, \chi_n\}$.

Definition 3.5 A substitution σ is called an anti-unifier of a set of AUPs A, if $dom(\sigma) \subseteq gvar(A)$ and for each $(\chi : \tilde{s} \triangleq \tilde{q}) \in A$, $\chi\sigma$ is a generalization of both $\tilde{s}$ and $\tilde{q}$.

Similarly, least general anti-unifiers are also generalized for sets of AUPs.

Definition 3.6 We say that a set of AUPs A is simplified if any anti-unifier of A is equal to Id modulo variable renaming.

Notice that if A is simplified then A cannot contain equations with pairs of terms with the same root $x : f(\tilde{s}) \triangleq f(\tilde{q})$, equations between equal sequences $X : \tilde{s} \triangleq \tilde{s}$, equations between terms $X : f(\tilde{s}) \triangleq g(\tilde{q})$ where $X \in \mathcal{V}_{\mathsf{H}}$, nor pairs of identical equations $\chi : \tilde{s} \triangleq \tilde{q}$, $\chi' : \tilde{s} \triangleq \tilde{q}$. The purpose of simplified AUPs is to restrict the set of problems that can form the store (see Lemma 3.8).

The following lemmas are not hard to prove. (The derivations are performed in $\mathfrak{G}$. We do not write it explicitly.)

Lemma 3.7 *If* $A; S; \sigma \Longrightarrow^* A'; S'; \sigma'$, *then, for any substitution* ϑ, *we have* $A; S; \vartheta\sigma \Longrightarrow^* A'; S'; \vartheta\sigma'$.

Lemma 3.8 *If* S *is simplified and* $A; S; \sigma \Longrightarrow A'; S'; \sigma'$ *then* S' *is also simplified.*

Next, we prove the following lemma that will imply the completeness theorem (Theorem 3.10).

Lemma 3.9 *Let A be a set of AUPs satisfying $gvar(A) \subseteq \mathcal{V}_{\mathsf{H}}$. Let S be an simplified set of AUPs. Let ϑ be an anti-unifier of A. Then, there exists a sequence of transformations $A; S; Id \Longrightarrow^* \emptyset; S'; \sigma$ where $\vartheta \preceq_{gvar(A)} \sigma$.*

Proof Since a minimal complete set of anti-unifiers exists, we can assume without loss of generality that ϑ is a least general anti-unifier.

We proceed by structural induction on A.

If $A = \emptyset$, the empty sequence of transformations allows us to get $\sigma = Id$ that is the unique anti-unifier of A, i.e., satisfying $dom(\sigma) \subseteq gvar(A) = \emptyset$.

If A is nonempty, we consider some equation $A = \{X : \tilde{s} \triangleq \tilde{s}'\} \cup A'$. There are several cases:

1. Case $(\chi' : \tilde{s} \triangleq \tilde{s}') \in S$, for some variable χ'.
   Since $(\chi' : \tilde{s} \triangleq \tilde{s}') \in S$ we can apply the rule Rec to $\{X : \tilde{s} \triangleq \tilde{s}'\}$, getting $A'; S; \{X \mapsto \chi'\}$. The set A' is smaller than A. If ϑ is an anti-unifier of A, then ϑ, with the domain restricted to $gvar(A')$, is an anti-unifier of A'. We can apply the induction hypothesis and Lemma 3.7 and construct a sequence

   $$\{X : \tilde{s} \triangleq \tilde{s}'\} \cup A'; S; Id \Longrightarrow_{\mathsf{Rec}} A'; S; \{X \mapsto \chi'\} \Longrightarrow^* \emptyset; S'; \{X \mapsto \chi'\}\sigma$$

   where $\vartheta \preceq_{gvar(A')} \sigma$. Now we have to prove $\vartheta \preceq_{gvar(A)} \{X \mapsto \chi'\}\sigma$, where $gvar(A) = gvar(A') \cup \{X\}$.
   Since ϑ is an anti-unifier of a set of AUPs that contains $X : \tilde{s} \triangleq \tilde{s}'$, and $(\chi' : \tilde{s} \triangleq \tilde{s}') \in S$ being S simplified, we must have $X\vartheta = \chi''$ for some variable χ''. On the other hand, $X\{X \mapsto \chi'\}\sigma = \chi'$. Notice that, since ϑ is least general and S is simplified, χ'' and χ' are both term variables, if both $\tilde{s}$ and $\tilde{s}'$ are terms, or both are hedge variables, if $\tilde{s}$ or $\tilde{s}'$ are not terms.
   Therefore, to prove the inequality $\vartheta \preceq_{gvar(A)} \{X \mapsto \chi'\}\sigma$, we have to prove that any variable satisfying $Z\vartheta = \chi''$ also satisfies $Z\{X \mapsto \chi'\}\sigma = Z\sigma = \chi'$. Since $X\vartheta = Z\vartheta$, we have $(Z : \tilde{s} \triangleq \tilde{s}') \in A'$. In the transformation $A'; S; \{X \mapsto \chi'\} \Longrightarrow^* \emptyset; S'; \{X \mapsto \chi'\}\sigma$ we will deal with this equation $Z : \tilde{s} \triangleq \tilde{s}'$. Since we will only be able to apply rule Rec to this equation, and σ will contain an instantiation $Z \mapsto \chi'$, we get $Z\{X \mapsto \chi'\}\sigma = \chi'$.
2. Otherwise, $(\chi' : \tilde{s} \triangleq \tilde{s}') \notin S$ for any χ'. Let $\tilde{s}$ and $\tilde{s}'$ be sequences of the form $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_{n'}$, respectively. We consider three cases, depending on the values of n and n'.
   (a) Case $n = n' = 0$. We have $\vartheta = \{X \mapsto \varepsilon\}$. We can get this anti-unifier applying the rule T-H.
   (b) Case $n = 0$ and $n' = 1$, or $n = 1$ and $n' = 0$. Since the equation is not in S, we can apply the rule Sol1-H or Sol2-H and the analysis is simple, because nothing is added to σ.
   (c) Case $n + n' \geq 2$. Since ϑ is an lgg, using a similar argument as in the proof of Lemma 2.1, $X\vartheta = q_1, \ldots, q_m$, where $m \leq n + n'$. We also have that there exist two substitutions ρ and ρ' such that for all $i \in \{1, \ldots, m\}$, either $q_i\rho = \varepsilon$, or $q_i\rho' = \varepsilon$, or there exist $j \in \{1, \ldots, n\}$ and $j' \in \{1, \ldots, n'\}$ such that $q_i\rho = s_j$ and $q_i\rho' = s'_{j'}$. Finally, least generality of ϑ also implies that for all $i \in \{1, \ldots, m\}$, either $q_i\rho \neq \varepsilon$ or $q_i\rho' \neq \varepsilon$.
   Therefore, there are three cases:
      i. Case $q_1\rho = s_1$ and $q_1\rho' = s'_1$.

A. If $n = n' = 1$ and s_1 and s'_1 are both terms with the same top symbol. Hence, let $A = \{X : f(\tilde{q}) \triangleq f(\tilde{q}')\} \cup A'$ be the selected equation, and $X\vartheta\rho = f(\tilde{q})$ and $X\vartheta\rho' = f(\tilde{q}')$. Since ϑ is least general, we have $X\vartheta = f(\tilde{s}'')$ for some sequence $\tilde{s}''$.
We can apply the transformation Dec-T:

$$\{X : f(\tilde{q}) \triangleq f(\tilde{q}')\} \cup A'; S; Id \Longrightarrow_{\text{Dec-T}}$$
$$\{Y : \tilde{q} \triangleq \tilde{q}'\} \cup A'; S; \{X \mapsto f(Y)\}$$

We can construct ϑ' as a substitution satisfying $dom(\vartheta') = dom(\vartheta) \setminus \{X\} \cup \{Y\}$, $Y\vartheta' = \tilde{s}''$ and $\chi'\vartheta' = \chi'\vartheta$, for any other variable χ' distinct from Y.
It is easy to see that ϑ' is an anti-unifier of $\{Y : \tilde{q} \triangleq \tilde{q}'\} \cup A'$.
Notice that the set $\{Y : \tilde{q} \triangleq \tilde{q}'\} \cup A'$ is structurally smaller than A. Therefore, by the induction hypothesis there exists a sequence of transformations $\{Y : \tilde{q} \triangleq \tilde{q}'\} \cup A'; S; Id \Longrightarrow^* \emptyset; S'; \sigma'$, where $\vartheta' \preceq_{gvar(A')\cup\{Y\}} \sigma'$. Therefore, by Lemma 3.7,

$$\{Y : \tilde{q} \triangleq \tilde{q}'\} \cup A'; S; \{X \mapsto f(Y)\} \Longrightarrow^* \emptyset; S'; \{X \mapsto f(Y)\}\sigma'$$

It is easy to see that $\vartheta \preceq_{gvar(A)} \{X \mapsto f(Y)\}\sigma'$. In particular, for the variable X we have $X\vartheta = f(\tilde{s}'') = f(Y)\vartheta' = X\{X \mapsto f(Y)\}\vartheta' \preceq X\{X \mapsto f(Y)\}\sigma'$ Therefore, the sequence of transformations

$$\{X : f(\tilde{q}) \triangleq f(\tilde{q}')\} \cup A'; S; Id \Longrightarrow_{\text{Dec-T}}$$
$$\{Y : \tilde{q} \triangleq \tilde{q}'\} \cup A'; S; \{X \mapsto f(Y)\} \Longrightarrow^*$$
$$\emptyset; S'; \{X \mapsto f(Y)\}\sigma'$$

satisfies the statement of the lemma.

B. If $n = n' = 1$ and s_1 and s'_1 are both terms with distinct top. Hence, let $A = \{X : l \triangleq r\} \cup A'$ be the selected equation, where l and r are non-variable terms and $top(l) \neq top(r)$. Let $X\vartheta\rho = l$ and $X\vartheta\rho' = r$. Since S does not contain any equation of the form $\chi : l \triangleq r$, we can apply the rule Sol-T. If ϑ is an anti-unifier of A, then, removing X from its domain, it is also an anti-unifier of A'. By induction hypothesis, and Lemmas 3.7 and 3.8, we can construct a sequence

$$\{X : l \triangleq r\} \cup A'; S; Id \Longrightarrow_{\text{Sol-T}}$$
$$A'; \{y : l \triangleq r\} \cup S; \{X \mapsto y\} \Longrightarrow^*$$
$$\emptyset; S'; \{X \mapsto y\}\sigma$$

where $\vartheta \preceq_{gvar(A')} \sigma$. Now we have to prove the relation $\vartheta \preceq_{gvar(A)} \{X \mapsto y\}\sigma$. The argument is similar to the one in Case 1.

C. If $n = n' = 1$ and s_1 or s_2 is a hedge variable.
We apply the rule Sol3-H, and the analysis is simple.

D. Either $n \geq 2$ or $n' \geq 2$.
We apply the rule Dec1-H, obtaining the system $\{Y : s_1 \triangleq s'_1,\ Z : s_2, \ldots, s_n \triangleq s'_2, \ldots, s'_{n'}\} \cup A'; S; \{X \mapsto Y, Z\}$, which is structurally

smaller. If ϑ is an anti-unifier of A then we can construct ϑ' with domain $dom(\vartheta') = dom(\vartheta) \setminus \{X\} \cup \{Y, Z\}$ where $Y\vartheta' = q_1$ and $Z\vartheta' = q_2, \ldots, q_m$, and $\chi\vartheta = \chi\vartheta'$ for $\chi \neq Y, Z$. By the induction hypothesis, we can construct

$$\{X : s_1, \ldots, s_n \triangleq s'_1, \ldots, s'_{n'}\} \cup A'; S; \Longrightarrow_{\text{Dec1-H}}$$
$$\{Y : s_1 \triangleq s'_1, Z : s_2, \ldots, s_n \triangleq s'_2, \ldots, s'_{n'}\} \cup A';$$
$$S; \{X \mapsto Y, Z\} \Longrightarrow^*$$
$$\emptyset; S'; \{X \mapsto Y, Z\}\sigma$$

where $\vartheta' \preceq_{gvar(A')} \sigma$, hence $\vartheta \preceq_{gvar(A)} \{X \mapsto Y, Z\}\sigma$.

ii. Case $q_1\rho = \varepsilon$ and $q_1\rho' \neq \varepsilon$. Hence, q_1 must be a hedge variable. Let $q_1\rho' = s'_1, \ldots, s'_i$, with $i \geq 1$. We apply i times the rule Dec3-H, obtaining $\{Y_1 : \varepsilon \triangleq s'_1, \ldots, Y_i : \varepsilon \triangleq s'_i,\ Z : s_1, \ldots, s_n \triangleq s'_{i+1}, \ldots, s'_{n'}\} \cup A'; S; \{X \mapsto Y_1, \ldots, Y_i, Z\}$. We construct ϑ' such that $Z\vartheta' = q_2, \ldots, q_m$ and $Y_j\vartheta' = Y_j$, for $j = 1, \ldots, i$. If ϑ is least general, so must be ϑ'. By the induction hypothesis we can construct

$$\{X : s_1, \ldots, s_n \triangleq s'_1, \ldots, s'_{n'}\} \cup A'; S; Id \Longrightarrow^{(i)}_{\text{Dec3-H}}$$
$$\{Y_1 : \varepsilon \triangleq s'_1, \ldots, Y_i : \varepsilon \triangleq s'_i,$$
$$Z : s_1, \ldots, s_n \triangleq s'_{i+1}, \ldots, s'_{n'}\} \cup A';$$
$$S; \{X \mapsto Y_1, \ldots, Y_i, Z\} \Longrightarrow^*$$
$$\emptyset; S'; \{X \mapsto Y_1, \ldots, Y_i, Z\}\sigma$$

where $\vartheta' \preceq_{gvar(A')} \sigma$, hence $\vartheta \preceq_{gvar(A)} \{X \mapsto Y_1, \ldots, Y_i, Z\}\sigma$. Notice that the following relations hold:

$$X\vartheta = q_1, \ldots, q_m,$$
$$(Y_1, \ldots, Y_i, Z)\vartheta' = Y_1, \ldots, Y_i, q_2, \ldots, q_m,$$
$$Y_1, \ldots, Y_i, q_2, \ldots, q_m \preceq_{gvar(A')} (Y_1, \ldots, Y_i, Z)\sigma,$$

where q_1 is a hedge variable. But we have that $(Y_1, \ldots, Y_i, Z)\sigma = X\{X \mapsto Y_1, \ldots, Y_i, Z\}\sigma$. Hence, the hedge $X\{X \mapsto Y_1, \ldots, Y_i, Z\}\sigma$ is less general than the hedge $Y_1, \ldots, Y_i, q_2, \ldots, q_m$ that is also less general than $q_1, \ldots, q_m$.

iii. Case $q_1\rho \neq \varepsilon$ and $q_1\rho' = \varepsilon$. This case is analogous to the previous one but using Dec2-H instead of Dec3-H. □

Theorem 3.10 (Completeness of $\mathfrak{G}$) *Let ϑ be an anti-unifier of $X : \tilde{s} \triangleq \tilde{q}$. Then $\mathfrak{G}$ computes a substitution σ such that $X\vartheta \preceq X\sigma$.*

Proof Immediate consequence of Lemma 3.9 with $A = \{X : \tilde{s} \triangleq \tilde{q}\}$ and $S = \emptyset$. □

Hence, collecting all the hedges $X\sigma$ such that $\{X : \tilde{s} \triangleq \tilde{q}\}; \emptyset; Id \Longrightarrow^* \emptyset; S; \sigma$, we obtain a finite complete set of generalizations of $\tilde{s}$ and $\tilde{q}$. In general, this set is

not minimal. Even for such a simple input as $\{X : f(a) \triangleq f(b)\}; \emptyset; Id$, the iterative application of $\mathfrak{G}$ produces five generalizations: two hedges Y_1, Y_2 and Z_1, Z_2 and three terms $f(U_1, U_2)$, $f(V_1, V_2)$, and $f(x)$.

Nevertheless, this redundancy is not trivially avoidable because rules allowing apparently useless alignments are needed for completeness:

Answer to Quiz 1: *Besides the "expected" lgg $f(a), f(X)$, the set $mcg(\tilde{s} \triangleq \tilde{q})$ for $\tilde{s} = f(a), f(a)$ and $\tilde{q} = f(a), f$ also contains two less obvious ones: the hedges $f(X, Y), f(X)$ and $f(X, Y), f(Y)$.*

In order not to miss such "less obvious" generalizations, we need all the rules from $\mathfrak{G}$. However, these rules generate much more than necessary. (In the case of Quiz 1, $\mathfrak{G}$ computes 33 generalizations.) Therefore, the computed set should be further minimized to keep only least general generalizations. Minimization involves a matchability test between two hedges. If two hedges $\tilde{s}$ and $\tilde{q}$ are in the set we are going to minimize, then we proceed as follows:

- If $\tilde{s} \simeq \tilde{q}$, then we delete one of them and keep the other (e.g., with the smaller size).
- If one of them is strictly more general than the other one, we delete the more general hedge and keep the more specific one.

For matchability, one could, in principle, use the hedge matching algorithm from Kutsia [20], but there is a subtlety one should take into account: The hedges that are to be matched, in general, are not ground. Therefore, when trying to match, e.g., $\tilde{s} = X, X$ to $\tilde{q} = X, a$, we should rename X in $\tilde{q}$ into a new constant. Furthermore, we should introduce a restriction that no term variable matches such new constants. Thus, the matchability test should fail for the problems like $X, X \ll X, a$ and $x \ll X$.

Hence, combining $\mathfrak{G}$ with minimization, we can compute $mcg(\tilde{s}_1 \triangleq \tilde{s}_2)$ for each $\tilde{s}_1$ and $\tilde{s}_2$.

Example 3.11 For the terms $f(g(a, X), a, X, b)$ and $f(g(b), b)$, the algorithm $\mathfrak{G}$ computes the *mcg* consisting of four lggs: $\{f(g(x, Y), Z, Y, b), f(g(x, Y), x, Y, Z), f(g(U, Y, Z), Y, Z, b), f(g(U, Y, Z), U, Y, b)\}$. They are selected from 169 generalizations computed in the first step of the algorithm.

Example 3.12 For $f(g(a, a), g(b, b), f(g(a), g(a)))$ and $f(g(a, a), f(g(a), g))$ the algorithm $\mathfrak{G}$ computes the *mcg* consisting of 65 elements. This set can be compactly written as

$$\{f(g(a, a), x, X), f(Z, g(x, x), f(g(a), g(X)))\} \cup$$
$$\{f(t_1, Z, t_2) \mid t_1 \in S_1, t_2 \in S_2\} \cup$$
$$\{f(Z, t_1, t_2) \mid t_1 \in S_3, t_2 \in S_4\} \cup$$
$$\{f(g(a, a), Z, t) \mid t \in S_5\},$$

where

$$
\begin{aligned}
S_1 &= \{g(X, X, Y, Y), g(X, Y, Y, X), g(X, Y, X, Y),\\
&\quad g(X, a, Y), g(X, Y, a), g(a, X, Y)\},\\
S_2 &= \{f(g(Y, X), g(Y)),\ f(g(X, Y), g(Y)),\ f(g(a), g(Y)),\\
&\quad f(g(Y, X), g(X)),\ f(g(X, Y), g(X)),\ f(g(a), g(X))\},\\
S_3 &= \{g(X, X, U, U), g(U, U, X, X), g(X, U, U, X),\\
&\quad g(X, U, X, U), g(U, X, X, U), g(U, X, U, X),\\
&\quad g(x, X, U), g(x, U, X), g(X, x, U),\\
&\quad g(U, x, X), g(X, U, x), g(U, X, x)\},\\
S_4 &= \{f(g(X, Y), g(Y)),\ f(g(Y, X), g(Y))\},\\
S_5 &= \{f(g(a), g(X))\} \cup S_4.
\end{aligned}
$$

We have seen in Example 3.2 how one of these generalizations is computed. These lggs are selected from 1,866 generalizations generated in the first step of the algorithm.

The drawback of the algorithm $\mathfrak{G}$ is that it is highly nondeterministic. It may compute up to 3^n generalizations,[2] where n is the size of the input. (For instance, for $f(a_1, a_2, a_3, a_4, a_5)$ and $f(b_1, b_2, b_3, b_4, b_5)$ it computes 11,685 generalizations, most of them several times, until it selects a single one, e.g., $f(x_1, x_2, x_3, x_4, x_5)$, on the minimization step.) The minimization step involves NP-complete hedge matching algorithm (see [20, 25]) performed on the pairs of elements of the generalization set. Hence, this algorithm is only of theoretical interest and falls short of being practically useful. From the results computed by it one can see that consecutive hedge variables in generalizations are permitted. It makes computations quite costly, because it requires all possible applications of the decomposition rules. Our goal is to impose requirements on the set of generalizations such that, on the one hand, it is still "interesting", on the other hand, it can be computed faster in many cases. This leads us to the notion of rigid generalization, described in the next section.

4 Computing Rigid Generalizations

The main intuition behind rigid generalizations is to capture the structure (modulo a given rigidity property) of as many nonvariable terms in the input hedges as possible.

[2]Notice that the hedge decomposition rule has three non-deterministic choices.

It is parametrized by a binary *rigidity function* $\mathscr{R}$, which computes a finite set of *alignments* for strings, defined as follows:

Definition 4.1 (Alignment, Rigidity Function) Let w_1 and w_2 be strings of symbols. Then the sequence $a_1[i_1, j_1] \cdots a_n[i_n, j_n]$, for $n \geq 0$, is an alignment if

- i's and j's are positive integers such that $0 < i_1 < \cdots < i_n < |w_1|$ and $0 < j_1 < \cdots < j_n < |w_2|$, and
- $a_k = w_1|_{i_k} = w_2|_{j_k}$, for all $1 \leq k \leq n$.

A rigidity function $\mathscr{R}$ is a function that returns, for every pair of strings of symbols w_1 and w_2, a set of alignments of w_1 and w_2.

Example 4.2 We give some examples of rigidity functions.

- $\mathscr{R}$ returns the set of all longest common subsequences: $\mathscr{R}(abc, dd) = \{\varepsilon\}$, $\mathscr{R}(abcda, bcad) = \{b[2, 1]c[3, 2]a[5, 3], b[2, 1]c[3, 2]d[4, 4]\}$.
- $\mathscr{R}$ returns the set of all longest common subsequences whose length is at least 4: $\mathscr{R}(abcda, bcacda) = \{a[1, 3]c[3, 4]d[4, 5]a[5, 6], \quad b[2, 1]c[3, 4]d[4, 5]a[5, 6]\}$, $\mathscr{R}(abcda, bca) = \emptyset$.
- $\mathscr{R}$ returns the set of all longest common substrings: $\mathscr{R}(abc, dd) = \{\varepsilon\}$, $\mathscr{R}(abcda, bcada) = \{b[2, 1]c[3, 2], \ d[4, 4]a[5, 5]\}$, and $\mathscr{R}(abcda, bcad) = \{b[2, 1]c[3, 2]\}$.

Definition 4.3 ($\mathscr{R}$-Generalization) Given two (variable disjoint) hedges $\tilde{s}_1$ and $\tilde{s}_2$ and the rigidity function $\mathscr{R}$, we say that a hedge $\tilde{s}$ that generalizes both $\tilde{s}_1$ and $\tilde{s}_2$ is their *generalization with respect to* $\mathscr{R}$, or, in short, an $\mathscr{R}$-generalization, if either $\mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2)) = \emptyset$ and $\tilde{s}$ is a hedge variable, or there exists an alignment $f_1[i_1, j_1] \cdots f_n[i_n, j_n] \in \mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2))$, such that the following conditions are fulfilled:

1. The sequence $\tilde{s}$ does not contain pairs of consecutive hedge variables.
2. If we remove all hedge variables that occur as elements of $\tilde{s}$, we get a sequence of the form $f_1(\tilde{q}_1), \ldots, f_n(\tilde{q}_n)$.
3. For every $1 \leq k \leq n$, there exists a pair of sequences $\tilde{s}'_1$ and $\tilde{s}'_2$ such that $\tilde{s}_1|_{i_k} = f_k(\tilde{s}'_1), \tilde{s}_2|_{j_k} = f_k(\tilde{s}'_2)$ and $\tilde{q}_k$ is an $\mathscr{R}$-generalization of $\tilde{s}'_1$ and $\tilde{s}'_2$.

The motivation behind the item 3 of this definition is to define rigid generalizations recursively: If two terms have the same top function symbol which finds its way in an $\mathscr{R}$-generalization, then the arguments of these terms should be generalized with respect to the same rigidity function $\mathscr{R}$.

Note that, under this definition, $\mathscr{R}$-generalizations do not contain term variables. The minimal complete set of $\mathscr{R}$-generalizations of $\tilde{s}_1$ and $\tilde{s}_2$ is denoted by $mcg_{\mathscr{R}}(\tilde{s}_1 \triangleq \tilde{s}_2)$. An *$\mathscr{R}$-anti-unifier* of $X : \tilde{s}_1 \triangleq \tilde{s}_2$ is a substitution σ such that $X\sigma$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$.

Example 4.4 Let $\mathscr{R}(w_1, w_2)$ be the set of all longest common subsequences of w_1 and w_2.

- Let $t_1 = f(g(a, X), a, X, b)$ and $t_2 = f(g(b), b))$. Then $mcg_{\mathscr{R}}(t_1 \triangleq t_2) = \{f(g(Y), Z, b)\}$. Note that $mcg_{\mathscr{R}}(t_1 \triangleq t_2) \cap mcg(t_1 \triangleq t_2) = \emptyset$, see Example 3.11.

- Let $t_1 = f(g(a,a), g(b,b), f(g(a), g(a)))$ and $t_2 = f(g(a,a), f(g(a), g))$. Then we have $mcg_{\mathscr{R}}(t_1 \triangleq t_2) = \{f(g(a,a), Z, f(g(a), g(X))), f(Z, g(U), f(g(a), g(X)))\}$. One can note that $f(Z, g(U), f(g(a), g(X))) \notin mcg(t_1 \triangleq t_2)$, see Example 3.12.
- $mcg_{\mathscr{R}}(a, b \triangleq b, c) = mcg(a, b \triangleq b, c)$ and contains a single hedge X, b, Y.

Example 4.5 Let $\mathscr{R}(w_1, w_2)$ be the set of all longest common substrings of w_1 and w_2.

- $mcg_{\mathscr{R}}(f(g(a, X), a, X, b) \triangleq f(g(b), b)) = \{f(g(Y), Z), f(Z, b)\}$ (cf. Example 4.4).
- $mcg_{\mathscr{R}}(f(g(a,a), g(b,b), f(g(a), g(a))) \triangleq f(g(a,a), f(g(a), g)) = \{f(Z, g(U), f(g(a), g(X)))\}$ (cf. Example 4.4).
- $mcg_{\mathscr{R}}(a, a, b, f, f, f(a, a, b) \triangleq a, a, c, f, f, f(a, a, c)) = \{X, f, f, f(a, a, Y)\}$.
- $mcg_{\mathscr{R}}(a, a, b, b, f, f, f(a, a, b, b) \triangleq a, a, c, f, f, f(a, a, c)) = \{X, f, f, f(a, a, Y)\}$.

Quiz 2: *Let $\mathscr{R}$ be a function that computes the set of all common subsequences of minimal length 3 of its arguments. What is the $mcg_{\mathscr{R}}(\tilde{s} \triangleq \tilde{q})$ for two identical hedges $\tilde{s} = \tilde{q} = f(a, b, c), g(a), h(a)$?*

Our goal is to compute a minimal complete set of $\mathscr{R}$-generalizations. For this, we design a new set of transformation rules. It consists of only four rules shown below:

$\mathscr{R}$-Dec-H: **$\mathscr{R}$-Rigid Decomposition for Hedges**

$$\{X : \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma \Longrightarrow$$
$$\{Z_k : \tilde{s}_k \triangleq \tilde{q}_k \mid 1 \le k \le n\} \cup A;$$
$$\{Y_0 : \tilde{s}|_0^{i_1} \triangleq \tilde{q}|_0^{j_1}\} \cup \{Y_k : \tilde{s}|_{i_k}^{i_{k+1}} \triangleq \tilde{q}|_{j_k}^{j_{k+1}} \mid 1 \le k \le n-1\} \cup$$
$$\{Y_n : \tilde{s}|_{i_n}^{|\tilde{s}|+1} \triangleq \tilde{q}|_{j_n}^{|\tilde{q}|+1}\} \cup S;$$
$$\sigma\{X \mapsto Y_0, f_1(Z_1), Y_1, \ldots, Y_{n-1}, f_n(Z_n), Y_n\},$$

if $\mathscr{R}(top(\tilde{s}), top(\tilde{q}))$ contains a sequence $f_1[i_1, j_1] \cdots f_n[i_n, j_n]$ such that for all $1 \le k \le n$, $\tilde{s}|_{i_k} = f_k(\tilde{s}_k)$, $\tilde{q}|_{j_k} = f_k(\tilde{q}_k)$, and Y_0, Y_k's and Z_k's are fresh.

$\mathscr{R}$-S-H: **$\mathscr{R}$-Rigid Solve for Hedges**

$$\{X : \tilde{s} \triangleq \tilde{q}\} \cup A;\ S;\ \sigma \Longrightarrow A;\ \{X : \tilde{s} \triangleq \tilde{q}\} \cup S;\ \sigma,$$

if $\mathscr{R}(top(\tilde{s}), top(\tilde{q})) = \emptyset$. (Notice that this transformation is equivalent to rule $\mathscr{R}$-Dec-H where $\mathscr{R}(top(\tilde{s}), top(\tilde{q})) = \{\varepsilon\}$).

$\mathscr{R}$-CS1: **$\mathscr{R}$-Rigid Clean Store 1**

$$A;\ \{X_1 : \tilde{s} \triangleq \tilde{q}, X_2 : \tilde{s} \triangleq \tilde{q}\} \cup S;\ \sigma \Longrightarrow A;\ \{X_1 : \tilde{s} \triangleq \tilde{q}\} \cup S;\ \sigma\{X_2 \mapsto X_1\},$$

if $X_1 \neq X_2$.

$\mathscr{R}$-CS2: **$\mathscr{R}$-Rigid Clean Store 2**

$$A;\ \{X : \varepsilon \triangleq \varepsilon\} \cup S;\ \sigma \Longrightarrow A;\ S;\ \sigma\{X \mapsto \varepsilon\}$$

To compute $\mathscr{R}$-generalizations of $\tilde{s}$ and $\tilde{q}$, we start with $\{X : \tilde{s} \triangleq \tilde{q}\}; \emptyset; Id$ and apply the rules on the selected anti-unification equation(s) in all possible ways. The obtained procedure is denoted by $\mathfrak{G}(\mathscr{R})$.

The intuition behind the $\mathscr{R}$-Dec-H rule is that, once $\mathscr{R}$ gives the set of alignments of the strings $top(\tilde{s})$ and $top(\tilde{q})$, we choose one alignment from it, and rigid decomposition is not permitted to be performed on the equations formed by the

remaining subsequences of $\tilde{s}$ and $\tilde{q}$ (i.e, the ones that are generalized by Y's in $\mathscr{R}$-Dec-H). Otherwise, the generalization might violate the restrictions of Definition 4.3. Therefore, we move these equations to the store where the decomposition and solve rules do not apply. However, it may introduce certain redundancies in the store. These redundancies are dealt with the store cleaning rules. Another interesting observation is that $\mathfrak{G}(\mathscr{R})$ never introduces in the set A or S equations of the form $x : l \triangleq r$ where x is a term variable.

It should be noted that since we generalize variable disjoint hedges, the strings in $\mathscr{R}(top(\tilde{s}), top(\tilde{q}))$ (that are common subsequences of $top(\tilde{s})$ and $top(\tilde{q})$) do not contain variables. After application of rule $\mathscr{R}$-Dec-H, each hedge variable in the anti-unifier gets separated from the other variables by a nonvariable term, to obey the restriction 1 of Definition 4.3.

We did not have the store cleaning rules in our previous algorithm $\mathfrak{G}$, because the store there simply will never contain AUPs that need to be cleaned.

To show that the procedure $\mathfrak{G}(\mathscr{R})$ terminates, we define the complexity measure for $A; S; \sigma$ as a pair $(M(A), M(S))$, where M is defined as in the termination proof of $\mathfrak{G}$. The measures are compared lexicographically. Each rule strictly reduces it, therefore there can be no infinite transformation chains. All the rules, except $\mathscr{R}$-Dec-H, transform the selected equation(s) uniquely. $\mathscr{R}$-Dec-H can introduce only finitely many branchings, because $\mathscr{R}$ returns a finite set. Hence, the following theorem holds:

Theorem 4.6 *The procedure $\mathfrak{G}(\mathscr{R})$ terminates on any input and produces a system $\emptyset; S; \sigma$ where S is irreducible with respect to the store cleaning rules.*

Example 4.7 Let $f(g(a,a), g(b,b), f(g(a), g(a)))$ and $f(g(a,a), f(g(a), g))$ be two given terms and $\mathscr{R}$ be the function computing the set of all longest common subsequences. In Example 4.4 we saw that the $mcg_{\mathscr{R}}$ of these terms consists of two elements. Now we illustrate how the algorithm $\mathfrak{G}(\mathscr{R})$ computes one of these lggs, the term $f(Z, g(U), f(g(a), g(X)))$. Note that the $\mathscr{R}$-Dec-H rule puts in the store the AUPs of the form $Y : \varepsilon \triangleq \varepsilon$, which are subsequently eliminated by the $\mathscr{R}$-CS2 rule. Below we contract these steps into one to shorten the sequence of transformations. The contracted step is still denoted by $\mathscr{R}$-Dec-H, but the scrutinizing reader will notice that there are a couple of hidden $\mathscr{R}$-CS2 steps behind.

$$\{X_0 : f(g(a,a), g(b,b), f(g(a), g(a))) \triangleq f(g(a,a), f(g(a), g))\}; \emptyset; Id \Longrightarrow_{\mathscr{R}\text{-Dec-H}}$$
$$\{X_1 : g(a,a), g(b,b), f(g(a), g(a)) \triangleq g(a,a), f(g(a), g)\}; \emptyset; \{X_0 \mapsto f(X_1)\}$$

This system is transformed by the $\mathscr{R}$-Dec-H rule with the alignment $g[2,1]\,f[3,2]$:

$$\{U : b, b \triangleq a, a, X_2 : g(a), g(a) \triangleq g(a), g\}; \{Z : g(a,a) \triangleq \varepsilon\}; \{X_0 \mapsto f(Z, g(U), f(X_2)), \ldots\} \Longrightarrow_{\mathscr{R}\text{-S-H}}$$
$$\{X_3 : g(a), g(a) \triangleq g(a), g\}; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a\}; \{X_0 \mapsto f(Z, g(U), f(X_2)), \ldots\} \Longrightarrow_{\mathscr{R}\text{-Dec-H}}$$
$$\{X_4 : a \triangleq a, X : a \triangleq \varepsilon\}; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a\};$$

$$\{X_0 \mapsto f(Z, g(U), f(g(X_4), g(X))), \ldots\} \Longrightarrow_{\mathscr{R}\text{-Dec-H}}$$
$$\{X_5 : \varepsilon \triangleq \varepsilon, X : a \triangleq \varepsilon\}; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a\};$$
$$\{X_0 \mapsto f(Z, g(U), f(g(a(X_5)), g(X))), \ldots\} \Longrightarrow_{\mathscr{R}\text{-S-H}}$$
$$\{X : a \triangleq \varepsilon\}; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a, X_5 : \varepsilon \triangleq \varepsilon\};$$
$$\{X_0 \mapsto f(Z, g(U), f(g(a(X_5)), g(X))), \ldots\} \Longrightarrow_{\mathscr{R}\text{-CS2}}$$
$$\{X : a \triangleq \varepsilon\}; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a\};$$
$$\{X_0 \mapsto f(Z, g(U), f(g(a), g(X))), \ldots\} \Longrightarrow_{\mathscr{R}\text{-S-H}}$$
$$\emptyset; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a, X : a \triangleq \varepsilon\};$$
$$\{X_0 \mapsto f(Z, g(U), f(g(a), g(X))), \ldots\}$$

To prove soundness of $\mathfrak{G}(\mathscr{R})$, we need a couple of lemmas:

Lemma 4.8 *Let* $A; S; \vartheta \Longrightarrow_{\mathsf{R}_1} A_1; S_1; \vartheta\sigma_1 \Longrightarrow_{\mathsf{R}_2} A_2; S_2; \vartheta\sigma_1\sigma_2$ *be a sequence of transformations where* $\mathsf{R}_1 \in \{\mathscr{R}\text{-CS1}, \mathscr{R}\text{-CS2}\}$ *and* $\mathsf{R}_2 \in \{\mathscr{R}\text{-Dec-H}, \mathscr{R}\text{-S-H}\}$. *Then there exists a transformation sequence* $A; S; \vartheta \Longrightarrow_{\mathsf{R}_2} A'_1; S'_1; \vartheta\sigma_2 \Longrightarrow_{\mathsf{R}_1} A'_2; S'_2; \vartheta\sigma_2\sigma_1$ *such that* $A'_2 = A_2$, $S'_2 = S_2$, *and* $\vartheta\sigma_1\sigma_2 = \vartheta\sigma_2\sigma_1$.

Proof Since R_1 does not affect the first component in the system, we have $A_1 = A$ and $A'_2 = A'_1$. We perform the step R_2 in the second transformation sequence exactly in the same way as in the first one, choosing the same rule, the same AUP in A, the same alignment, and the same fresh variables. Then $A'_1 = A_2$ and, hence, $A'_2 = A_2$. As for the stores, S_2 consists of all the AUPs in S except those deleted by R_1 and R_2 and, in addition, it contains the AUPs introduced by R_2. In the second sequence of transformations, S'_1 consists of all the AUPs in S except the one deleted by R_2 and the ones introduced by R_2. In the last step, we delete from S'_1 exactly the same AUP that was deleted from S_1 by R_1. Therefore, we get $S'_2 = S_2$. Finally, σ_1 and σ_2 commute, because their domains and ranges are disjoint. Hence, $\vartheta\sigma_1\sigma_2 = \vartheta\sigma_2\sigma_1$. □

Lemma 4.9 *If* $A; S_1; \vartheta \Longrightarrow^* \emptyset; S_2; \vartheta\sigma$ *is a derivation in* $\mathfrak{G}(\mathscr{R})$ *using only the rules* $\mathscr{R}$-Dec-H *and* $\mathscr{R}$-S-H, *then for all* $(X : \tilde{s} \triangleq \tilde{q}) \in A$, *the hedge* $X\sigma$ *is an* $\mathscr{R}$*-generalization of* $\tilde{s}$ *and* $\tilde{q}$.

Proof We proceed by induction on the length of the derivation. If it is 1, then the derivation has the form $\{X : \tilde{s} \triangleq \tilde{q}\}; S; \vartheta \Longrightarrow^* \emptyset; \{X : \tilde{s} \triangleq \tilde{q}\} \cup S; \vartheta\sigma$, where $\sigma = \{X \mapsto Y_0\}$ for a fresh Y_0 if the used rule is $\mathscr{R}$-Dec-H, and $\sigma = Id$ if the used rule is $\mathscr{R}$-S-H. Since $\mathscr{R}(top(\tilde{s}), top(\tilde{q})) \subseteq \{\varepsilon\}$, $X\sigma$ is an $\mathscr{R}$-generalization of $\tilde{s}$ and $\tilde{q}$.

Now we assume that the lemma holds for all derivations with the length less than $m > 1$ and prove it for m. Let the system to be transformed be $\{X : \tilde{s} \triangleq \tilde{q}\} \cup A'; S; \vartheta$. If it is transformed by the rule $\mathscr{R}$-S-H then $\mathscr{R}(top(\tilde{s}), top(\tilde{q})) = \emptyset$, $\sigma = Id$, and we obtain a new system $A'; \{X : \tilde{s} \triangleq \tilde{q}\} \cup S; \vartheta$. By the induction hypothesis, $X'\sigma$ is an $\mathscr{R}$-generalization of $\tilde{s}'$ and $\tilde{q}'$ for all $(X' : \tilde{s}' \triangleq \tilde{q}') \in A'$. By the definition of $\mathscr{R}$-generalization, the same holds for $X\sigma, \tilde{s}$, and $\tilde{q}$ because $\mathscr{R}(top(\tilde{s}), top(\tilde{q})) = \emptyset$.

If the rule $\mathscr{R}$-Dec-H is used to transform $\{X : \tilde{s} \triangleq \tilde{q}\} \cup A'; S; \vartheta$, then the new system is $\{Z_k : \tilde{s}_k \triangleq \tilde{q}_k \mid 1 \le k \le n\} \cup A'; S'; \vartheta\sigma'$, where $\sigma' = \{X \mapsto$

$Y_0, f_1(Z_1), Y_1, \ldots, Y_{n-1}, f_n(Z_n), Y_n\}$ and the conditions of $\mathscr{R}$-Dec-H are satisfied. By the induction hypothesis, We have a derivation $\{Z_k : \tilde{s}_k \triangleq \tilde{q}_k \mid 1 \leq k \leq n\} \cup A'; S'; \vartheta\sigma' \Longrightarrow^* \emptyset; S''; \vartheta\sigma'\sigma''$ using only the rules $\mathscr{R}$-Dec-H and $\mathscr{R}$-S-H such that for all $(X' : \tilde{s}' \triangleq \tilde{q}') \in \{Z_k : \tilde{s}_k \triangleq \tilde{q}_k \mid 1 \leq k \leq n\} \cup A'$, the hedge $X'\sigma''$ is an $\mathscr{R}$-generalization of $\tilde{s}'$ and $\tilde{q}'$. In particular, this holds for Z's. Therefore, $X\sigma'\sigma''$ is an $\mathscr{R}$-generalization of $\tilde{s}$ and $\tilde{q}$. This finishes the proof. □

Lemma 4.10 *If* $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id \Longrightarrow^* \emptyset; S_1; \vartheta \Longrightarrow_{\mathsf{R}} \emptyset; S_2; \vartheta\sigma$ *is a derivation in* $\mathfrak{G}(\mathscr{R})$ *such that* $X\vartheta$ *is an* $\mathscr{R}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$ *and* $\mathsf{R} \in \{\mathscr{R}\text{-CS1}, \mathscr{R}\text{-CS2}\}$. *Then* $X\vartheta\sigma$ *is an* $\mathscr{R}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$.

Proof Let R be $\mathscr{R}$-CS1, transforming $\{X_1 : \tilde{s}'_1 \triangleq \tilde{s}'_2, X_2 : \tilde{s}'_1 \triangleq \tilde{s}'_2\} \subseteq S_1$ into $\{X_1 : \tilde{s}'_1 \triangleq \tilde{s}'_2\} \subseteq S_2$ with the substitution $\sigma = \{X_2 \mapsto X_1\}$. The hedges $\tilde{s}'_1$ and $\tilde{s}'_2$ occur in $\tilde{s}_1$ and $\tilde{s}_2$, respectively, so that the corresponding occurrences are abstracted by the same variable in $X\vartheta$. This variable for some pairs of occurrences of $\tilde{s}'_1$ and $\tilde{s}'_2$ is X_1 and for some others X_2. (There can be other variables as well that abstract $\tilde{s}'_1$ and $\tilde{s}'_2$.) Hence, if we replace X_2 with X_1 in $X\vartheta$, the obtained hedge $X\vartheta\sigma$ will be a generalization of $\tilde{s}_1$ and $\tilde{s}_2$.

To prove that after this replacement we still have an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$, we proceed by induction on the maximal depth d of the occurrences of X_2 in $X\vartheta$. It is enough to show that replacing only one occurrence of X_2 with X_1 retains the $\mathscr{R}$-generalization property.

Let first $\mathsf{d} = 0$. Then $X\vartheta$ has a form $\tilde{q}_1, X_2, \tilde{q}_2$. Replacing X_2 with X_1 gives $\tilde{q}_1, X_1, \tilde{q}_2$, that keeps the same alignment from $\mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2))$ that was in $X\vartheta$ and satisfies all three conditions of the definition of $\mathscr{R}$-generalization. Hence, $\tilde{q}_1, X_1, \tilde{q}_2$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$.

Now assume that $\mathsf{d} > 0$. It means that there exists a term $f(\tilde{q})$ in $X\vartheta$, such that X_2 occurs at depth $\mathsf{d} - 1$ in $\tilde{q}$. Then there are terms $f(\tilde{s}''_1)$ in $\tilde{s}_1$ and $f(\tilde{s}''_2)$ in $\tilde{s}_2$ such that $\tilde{q}$ is an $\mathscr{R}$-generalization of $\tilde{s}''_1$ and $\tilde{s}''_2$. By the induction hypothesis, replacing an occurrence of X_2 in $\tilde{q}$ with X_1 gives a hedge that is again an $\mathscr{R}$-generalization of $\tilde{s}''_1$ and $\tilde{s}''_2$. Hence, the hedge obtained from $X\vartheta$ by replacing one occurrence of X_2 with X_1 is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$, because we just showed that the third condition of the definition of $\mathscr{R}$-generalization is satisfied, while the other two conditions were not affected.

Repeating the process of replacement of one occurrence of X_2 by X_1 iteratively until there are no more X_2's in $X\vartheta$, we prove that $X\vartheta\sigma$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$.

Proof for $\mathsf{R} = \mathscr{R}$-CS2 is straightforward. □

Now we can prove the soundness theorem for $\mathfrak{G}(\mathscr{R})$:

Theorem 4.11 (Soundness of $\mathfrak{G}(\mathscr{R})$) *If* $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id \Longrightarrow^* \emptyset; S; \sigma$ *is a derivation in* $\mathfrak{G}(\mathscr{R})$, *then* $X\sigma$ *is an* $\mathscr{R}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$.

Proof By Lemma 4.8, every derivation in $\mathfrak{G}(\mathscr{R})$ can be reordered so that first only the rules $\mathscr{R}$-Dec-H and $\mathscr{R}$-S-H are applied until the set of AUPs becomes empty, and then the store is cleaned. The substitutions computed by the original derivation and by the reordered derivation coincide. Let σ' be the substitution

obtained at the end of the subderivation with $\mathscr{R}$-Dec-H and $\mathscr{R}$-S-H. By Lemma 4.9, $X\sigma'$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. By Lemma 4.10, substitutions introduced by the store cleaning rules keep the $\mathscr{R}$-generalization property. Hence, $X\sigma$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. □

Moreover, the algorithm $\mathfrak{G}(\mathscr{R})$ is complete, as the following theorem shows:

Theorem 4.12 (Completeness of $\mathfrak{G}(\mathscr{R})$) *Let $\tilde{q}$ be an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. Then $\mathfrak{G}(\mathscr{R})$ computes an $\mathscr{R}$-anti-unifier σ for $X : \tilde{s}_1 \triangleq \tilde{s}_2$ such that $\tilde{q} \preceq X\sigma$.*

Proof We prove the theorem by induction on the size of $\tilde{q}$. Let the initial system be $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id$. If the size of $\tilde{q}$ is 0, i.e., if $\tilde{q} = \varepsilon$, then $\tilde{s}_1 = \tilde{s}_2 = \varepsilon$. Therefore, $\mathfrak{G}(\mathscr{R})$ constructs the desired derivation first applying $\mathscr{R}$-Dec-H and then $\mathscr{R}$-CS2. The resulting anti-unifier is $\sigma = \{X \mapsto \varepsilon\}$. If the size of $\tilde{q}$ is 1, then there are several alternatives:

1. $\tilde{q} = X$. It means that $\mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2)) = \emptyset$. The desired derivation transforms the system with the $\mathscr{R}$-S-H rule, obtaining $\sigma = Id$.
2. $\tilde{q} = Y_0$ and $X \neq Y_0$. It means that $\varepsilon \in \mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2))$. The desired derivation transforms the system with the $\mathscr{R}$-Dec-H rule, obtaining $\sigma = \{X \mapsto Y'_0\}$ for a fresh Y'_0.
3. $\tilde{q} = a$. It means that $a[1, 1] \in \mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2))$. We can transform the system by the rule $\mathscr{R}$-Dec-H that is followed by a sequence of applications of the $\mathscr{R}$-CS2 rule, obtaining $\sigma = \{X \mapsto a\}$.

In general, since $\tilde{q}$ is an $\mathscr{R}$-generalization of the hedges $\tilde{s}_1$ and $\tilde{s}_2$, there exists an alignment $f_1[i_1, j_1] \cdots f_n[i_n, j_n] \in \mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2))$ such that

- $\tilde{q}$ does not contain two consecutive occurrences of hedge variables,
- the maximal subsequence of nonvariable terms occurring in $\tilde{q}$ has a form $f_1(\tilde{q}_1), \ldots, f_n(\tilde{q}_n)$, and
- for every $1 \leq k \leq n$, there exists a pair of sequences $\tilde{s}'_{1k}$ and $\tilde{s}'_{2k}$ such that $\tilde{s}_1|_{i_k} = f_k(\tilde{s}'_{1k})$, $\tilde{s}_2|_{j_k} = f_k(\tilde{s}'_{2k})$, and $\tilde{q}_k$ is an $\mathscr{R}$-generalization of $\tilde{s}'_{1k}$ and $\tilde{s}'_{2k}$.

Now, we assume as the induction hypothesis that for all hedges $\tilde{q}^h$ of size less than $\mathsf{s} > 1$ and for all $\tilde{s}^h_1, \tilde{s}^h_2$, if $\tilde{q}^h$ is an $\mathscr{R}$-generalization of $\tilde{s}^h_1, \tilde{s}^h_2$, then there exists a $\mathfrak{G}(\mathscr{R})$ computation of an $\mathscr{R}$-anti-unifier σ^h of $X^h : \tilde{s}^h_1 \triangleq \tilde{s}^h_2$ such that $\tilde{q}^h \preceq X^h\sigma_h$. We make the induction step for the hedges of size s.

Assume that $\tilde{q}$ has the form $U_0, f_1(\tilde{q}_1), U_1, \ldots, U_{n-1}, f_n(\tilde{q}_n), U_n$. Then, since $size(\tilde{q}_k) < size(\tilde{q})$, by the induction hypothesis there exists a derivation $\{Z_k : \tilde{s}'_{1k} \triangleq \tilde{s}'_{2k}\}; \emptyset; Id \Longrightarrow^* \emptyset; S_k, \sigma_k$ for each $1 \leq k \leq n$, such that $\tilde{q}_k \preceq Z_k\sigma_k$. Combining the decomposition step with them, we obtain a derivation $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id \Longrightarrow_{\mathscr{R}\text{-Dec-H}} \{Z_k : \tilde{s}'_{1k} \triangleq \tilde{s}'_{2k} \mid 1 \leq k \leq n\}; S_{\mathrm{dec}}; \sigma_{\mathrm{dec}} \Longrightarrow^* \emptyset; S_{\mathrm{dec}} \cup S_1 \cup \cdots \cup S_n; \sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n$. By Theorem 4.11, $X\sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. Since all Z_k's are distinct, $Z_k\sigma_k = Z_k\sigma_1 \cdots \sigma_n$ for all $1 \leq k \leq n$.

To get a computation of an $\mathscr{R}$-anti-unifier in $\mathfrak{G}(\mathscr{R})$ from the constructed derivation, we extend it with the store cleaning rules: $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id \Longrightarrow_{\mathscr{R}\text{-Dec-H}} \{Z_k : \tilde{s}'_{1k} \triangleq \tilde{s}'_{2k} \mid 1 \leq k \leq n\}; S; \sigma_{\mathrm{dec}} \Longrightarrow^* \emptyset; S_{\mathrm{dec}} \cup S_1 \cup \cdots \cup S_n; \sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n \Longrightarrow^*_{\mathscr{R}\text{-CS1}|\mathscr{R}\text{-CS2}} \emptyset; S; \sigma$.

If $\tilde{q}$ does not contain duplicated variables, then, by the construction of the derivation, we have $\tilde{q} \preceq X\sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n \preceq X\sigma$. To show that $\tilde{q} \preceq X\sigma$ also when there are duplicated variables in $\tilde{q}$, we assume without loss of generality that Y is a variable that occurs in $\tilde{q}$ in two different places. If $U_m = U_k = Y$ for some $m \neq k \in \{0, \ldots, n\}$, then $\tilde{s}_1|_{i_m}^{i_{m+1}} = \tilde{s}_1|_{i_k}^{i_{k+1}}$ and $\tilde{s}_2|_{j_m}^{j_{m+1}} = \tilde{s}_2|_{j_k}^{j_{k+1}}$. The first step in the derivation, the $\mathscr{R}$-Dec-H rule, introduces two AUPs in S: $Y_m : \tilde{s}_1|_{i_m}^{i_{m+1}} \triangleq \tilde{s}_2|_{j_m}^{j_{m+1}}$ and $Y_k : \tilde{s}_1|_{i_k}^{i_{k+1}} \triangleq \tilde{s}_2|_{j_k}^{j_{k+1}}$. Later, the store cleaning rules make Y_m and Y_k identical ensuring that $\tilde{q} \preceq X\sigma$. We can reason similarly, if the occurrences of Y happen in $\tilde{q}_m$ and $\tilde{q}_k$ for distinct $m \neq k \in \{1, \ldots, n\}$, or if one occurrence of Y is in some $\tilde{q}_k$ and another is one of the U's. Finally, if these occurrences are within the same $\tilde{q}_k$, then, since by the induction hypothesis $\tilde{q}_k\rho_k = Z_k\sigma_k = Z_k\sigma_1 \cdots \sigma_n$ for some ρ_k, we have the same term $Y_k\rho_k$ occurring twice in the corresponding positions in $Z_k\sigma_1 \cdots \sigma_n$. Since $Z_k\sigma_1 \cdots \sigma_n \preceq Z_k\sigma$ and $Z_k\sigma_1 \cdots \sigma_n = Z_k\sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n$, we have $\tilde{q}_k \preceq Z_k\sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n$. Therefore, $\tilde{q} \preceq X\sigma_{\mathrm{dec}}\sigma_1 \cdots \sigma_n \preceq X\sigma$.

Hence, we constructed a derivation in $\mathfrak{G}(\mathscr{R})$ that computes an $\mathscr{R}$-anti-unifier σ of $\tilde{s}_1$ and $\tilde{s}_2$ with the property $\tilde{q} \preceq X\sigma$. □

We may prune the search space of the algorithm $\mathfrak{G}(\mathscr{R})$, giving priority to the rules $\mathscr{R}$-CS1 and $\mathscr{R}$-CS2. If they are applicable to a system, no other rule should apply to it. It can prevent re-computing equivalent $\mathscr{R}$-generalizations on different branches without violating completeness. In addition, we may forbid the rule $\mathscr{R}$-Dec-H to add to the set A the AUPs of the form $Z_k : \varepsilon \triangleq \varepsilon$ for $1 \leq k \leq n$, and to add to the set S the AUPs of the form $Y_m : \varepsilon \triangleq \varepsilon$ for $0 \leq m \leq n$. Respectively, such Z_k's and Y_m's are replaced by ε in the substitution computed by $\mathscr{R}$-Dec-H. These simplifications can be justified by the fact that those AUPs, anyway, eventually will be eliminated by the $\mathscr{R}$-CS2 rule. Therefore, they do not affect completeness. In the examples below we assume that $\mathfrak{G}(\mathscr{R})$ is optimized in such ways. The length of each derivation under the optimized $\mathfrak{G}(\mathscr{R})$ does not exceed the size of the input problem.

To compute a minimal complete set of $\mathscr{R}$-generalizations, we still need to perform the minimization step, unless the cardinality of the set that $\mathscr{R}$ computes is not greater than 1. In the latter case the $\mathfrak{G}(\mathscr{R})$ computes a single $\mathscr{R}$-generalization of the input hedges.

Hence, combining $\mathfrak{G}(\mathscr{R})$ with the minimization step, we can compute the set $mcg_{\mathscr{R}}(\tilde{s}_1 \triangleq \tilde{s}_2)$ for any hedges $\tilde{s}_1, \tilde{s}_2$, and the rigidity function $\mathscr{R}$.

Answer to the Quiz 2: *Given two identical hedges* $\tilde{s} = f(a, b, c), g(a), h(a)$ *and* $\tilde{q} = f(a, b, c), g(a), h(a)$ *and* $\mathscr{R}$ *being a function that computes the set of all common subsequences of the minimal length 3 of its arguments,* $mcg_{\mathscr{R}}(\tilde{s} \triangleq \tilde{q}) = \{f(a, b, c), g(X), h(X)\}$. *One might expect the lgg to be only the original hedge* $f(a, b, c), g(a), h(a)$ *itself, but it violates the condition* 3 *of Definition* 4.3.

The rigidity function in the Quiz 2 is motivated by the idea that if similarity between structures of two hedges is "sufficiently big", then it is retained in the generalization. "Too small" similar pieces in any level are ignored and abstracted by a variable. This can be useful, for instance, if one wants to see whether two trees branch in a similar way, where only "heavy" branching is interesting.

The example in this Quiz makes it clear why among the $\mathscr{R}$-generalization rules, we do not have the one that would generalize two identical terms with the same

term (the so called Trivial Terms rule). It would simply make the $\mathfrak{G}(\mathscr{R})$ algorithm unsound.

5 Rigid Generalizations with Term Variables

In this section we show how to improve the precision of rigid anti-unification, permitting term variables to occur in rigid generalizations. This can be done with a relatively small computational overhead. First, we need to redefine the notion of $\mathscr{R}$-generalization:

Definition 5.1 ($\mathscr{R}$-Generalization with Term Variables) Given two variable disjoint hedges $\tilde{s}_1$ and $\tilde{s}_2$ and the rigidity function $\mathscr{R}$, we say that a hedge $\tilde{s}$ that generalizes both $\tilde{s}_1$ and $\tilde{s}_2$ is their *generalization with respect to $\mathscr{R}$ allowing term variables,* or, in short, an $\mathscr{R}_\mathsf{T}$-generalization, if either $\mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2)) = \emptyset$ and $\tilde{s}$ is a hedge variable or a sequence of term variables, or there exists an alignment $f_1[i_1, j_1] \cdots f_n[i_n, j_n] \in \mathscr{R}(top(\tilde{s}_1), top(\tilde{s}_2))$, such that the following conditions are fulfilled:

1. If the sequence $\tilde{s}$ contains a pair of consecutive variables, then both of them are term variables.
2. If we remove all variables that occur as elements of $\tilde{s}$, we get a sequence of the form $f_1(\tilde{q}_1), \ldots, f_n(\tilde{q}_n)$.
3. For every $1 \leq k \leq n$, there exists a pair of sequences $\tilde{s}'_1$ and $\tilde{s}'_2$ such that $\tilde{s}_1|_{i_k} = f_k(\tilde{s}'_1), \tilde{s}_2|_{j_k} = f_k(\tilde{s}'_2)$ and $\tilde{q}_k$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}'_1$ and $\tilde{s}'_2$.

The minimal complete set of $\mathscr{R}_\mathsf{T}$-generalizations of $\tilde{s}_1$ and $\tilde{s}_2$ is denoted by $mcg_{\mathscr{R}_\mathsf{T}}(\tilde{s}_1 \triangleq \tilde{s}_2)$. An $\mathscr{R}_\mathsf{T}$-*anti-unifier* of $X : \tilde{s} \triangleq \tilde{q}$ is a substitution σ such that $X\sigma$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}$ and $\tilde{q}$.

Example 5.2 The $\mathscr{R}_\mathsf{T}$-counterparts of $\mathscr{R}$-generalizations of the hedges in the Examples 4.4, 4.5, and in quiz 2 look as follows:

1. $\mathscr{R}(w_1, w_2)$ is the set of all longest common subsequences of w_1 and w_2.
   - Let $t_1 = f(g(a, X), a, X, b)$ and $t_2 = f(g(b), b)$. Then $mcg_{\mathscr{R}_\mathsf{T}}(t_1 \triangleq t_2) = mcg_{\mathscr{R}}(t_1 \triangleq t_2) = \{f(g(Y), Z, b)\}$ and $mcg_{\mathscr{R}_\mathsf{T}}(t_1 \triangleq t_2) \cap mcg(t_1 \triangleq t_2) = \emptyset$.
   - Let $t_1 = f(g(a, a), g(b, b), f(g(a), g(a)))$ and $t_2 = f(g(a, a), f(g(a), g))$. Then we get $mcg_{\mathscr{R}_\mathsf{T}}(t_1 \triangleq t_2) = \{f(g(a, a), Z, f(g(a), g(X))), f(Z, g(x, x), f(g(a), g(X)))\}$. Hence, $mcg_{\mathscr{R}_\mathsf{T}}(t_1 \triangleq t_2) \neq mcg_{\mathscr{R}}(t_1 \triangleq t_2)$ and $mcg_{\mathscr{R}_\mathsf{T}}(t_1 \triangleq t_2) \subset mcg(t_1 \triangleq t_2)$.
   - $mcg_{\mathscr{R}_\mathsf{T}}(a, b \triangleq b, c) = mcg_{\mathscr{R}}(a, b \triangleq b, c) = mcg(a, b \triangleq b, c)$ and consists of a single hedge X, b, Y.
2. $\mathscr{R}(w_1, w_2)$ is the set of all longest common substrings of w_1 and w_2.
   - Let $t_1 = f(g(a, X), a, X, b)$ and $t_2 = f(g(b), b)$. Then $mcg_{\mathscr{R}_\mathsf{T}}(t_1 \triangleq t_2) = mcg_{\mathscr{R}}(t_1 \triangleq t_2) = \{f(g(Y), Z), f(Z, b)\}$.
   - $mcg_{\mathscr{R}_\mathsf{T}}(f(g(a, a), g(b, b), f(g(a), g(a))) \triangleq f(g(a, a), f(g(a), g))$ contains only the term $f(Z, g(x, x), f(g(a), g(X)))$. It is less general than the least general $\mathscr{R}$-generalization of these terms, $f(Z, g(U), f(g(a), g(X)))$.

- $mcg_{\mathscr{R}_\mathsf{T}}(a, a, b\,,\, f,\, f,\, f(a, a, b) \triangleq a, a, c,\, f,\, f,\, f(a, a, c))$ contains one hedge $x, x, y,\, f,\, f,\, f(a, a, y)$. It is less general than their least general $\mathscr{R}$-generalization $X,\, f,\, f,\, f(a, a, Y)$.
- $mcg_{\mathscr{R}_\mathsf{T}}(a, a, b\,, b\,,\, f,\, f,\, f(a, a, b\,, b) \triangleq a, a, c,\, f,\, f,\, f(a, a, c))$ contains only one hedge $X,\, f,\, f,\, f(a, a, Y)$. (The same as the least general $\mathscr{R}$-generalization.)

3. $\mathscr{R}(w_1, w_2)$ is the set of all common subsequences with the length greater or equal than 3 of w_1 and w_2.

   - $mcg_{\mathscr{R}_\mathsf{T}}(f(a, b\,, c), g(a), h(a) \triangleq f(a, b\,, c), g(a), h(a))$ consists of one hedge $f(a, b\,, c),\ g(x), h(x)$, which is less general than the least general $\mathscr{R}$-generalization $f(a, b\,, c),\ g(X), h(X)$. The difference between these two generalizations illustrates the difference in the motivation of using such a rigidity function with or without term variables. In both cases the "sufficiently big" similarities are retained. As for the "small similarities", in the case without term variables they are completely ignored (generalized by a hedge variable). In the presence of term variables the similar structure can be kept (generalized by term variables for each branch), only the labels are not interesting.

An algorithm $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ that computes $\mathscr{R}_\mathsf{T}$-generalizations can be obtained from $\mathfrak{G}(\mathscr{R})$ by adding two new store cleaning rules:

$\mathscr{R}$-CS3: **$\mathscr{R}$-Rigid Clean Store 3**
$A;\ \{x_1 : l \triangleq r,\ x_2 : l \triangleq r\} \cup S;\ \sigma \Longrightarrow A;\ \{x_1 : l \triangleq r\} \cup S;\ \sigma\{x_2 \mapsto x_1\}$

$\mathscr{R}$-CS4: **$\mathscr{R}$-Rigid Clean Store 4**
$A;\ \{X : l_1, \ldots, l_n \triangleq r_1, \ldots, r_n\} \cup S;\ \sigma \Longrightarrow$
$A;\ \{x_1 : l_1 \triangleq r_1, \ldots, x_n : l_n \triangleq r_n\} \cup S;\ \sigma\{X \mapsto x_1, \ldots, x_n\},$
where $n \geq 1$ and x_i's are fresh.

In fact, derivations performed by $\mathfrak{G}(\mathscr{R})$ can be extended by these rules to obtain $\mathscr{R}_\mathsf{T}$-generalizations. $\mathscr{R}_\mathsf{T}$-generalizations can be seen as further refinements of $\mathscr{R}$-generalizations.

Termination of $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ is not hard to establish: Define the complexity measure for $A; S; \sigma$ as a triple $(M(A), M(S), |S|)$, where M is defined as in the termination proof of $\mathfrak{G}$ and $|S|$ is the cardinality of S. Measures are ordered lexicographically. Then the rules taken from $\mathfrak{G}(\mathscr{R})$ strictly decrease it, because they decrease the lexicographic combination of the first two components, as it was shown in the proof of termination of $\mathfrak{G}(\mathscr{R})$. As for the new rules, $\mathscr{R}$-CS3 strictly decreases the third component without changing the first two, and $\mathscr{R}$-CS4 strictly decreases the second one without changing the first. Since the ordering is well-founded, we get termination of $\mathfrak{G}(\mathscr{R}_\mathsf{T})$.

To prove soundness, we need a couple of lemmas:

Lemma 5.3 *Let* $A; S; \vartheta \Longrightarrow_{\mathsf{R}_1} A_1; S_1; \vartheta\sigma_1 \Longrightarrow_{\mathsf{R}_2} A_2; S_2; \vartheta\sigma_1\sigma_2$ *be a sequence of transformations where* $\mathsf{R}_1 \in \{\mathscr{R}\text{-CS3}, \mathscr{R}\text{-CS4}\}$ *and* $\mathsf{R}_2 \in \mathfrak{G}(\mathscr{R})$. *Then there exists a transformation sequence* $A; S; \vartheta \Longrightarrow_{\mathsf{R}_2} A'_1; S'_1; \vartheta\sigma_2 \Longrightarrow_{\mathsf{R}_1} A'_2; S'_2; \vartheta\sigma_2\sigma_1$ *such that* $A'_2 = A_2$, $S'_2 = S_2$, *and* $\vartheta\sigma_1\sigma_2 = \vartheta\sigma_2\sigma_1$.

Proof Since R_1 does not affect the first component in the system, we have $A_1 = A$ and $A'_2 = A'_1$. We perform the step R_2 in the second transformation sequence exactly in the same way as in the first one, choosing the same rule, the same AUP in A, the same alignment, and the same fresh variables. Then $A'_1 = A_2$ and, hence, $A'_2 = A_2$. As for the stores, R_2 can not process in S_1 an AUP introduced by R_1. Hence, it can process in S_1 only an AUP that was already in S. So, S_2 is obtained from S by replacing at most two AUPs with zero or more AUPs. In the second derivation we just repeat the same transformation to obtain S'_1 from S. On the other hand, the AUPs processed by R_1 in the first derivation are still in S'_1. They remained there from S. To finish the construction of the second sequence, we can repeat transformation performed by R_1 in the first derivation. Then we get that S'_2 is obtained from S by replacing at most two AUPs with zero or more AUPs. Since the replaced and new AUPs in the second derivation are exactly those in the first derivation, we obtain that $S'_2 = S_2$. Finally, σ_1 and σ_2 commute, because their domains and ranges are disjoint. Hence, $\vartheta\sigma_1\sigma_2 = \vartheta\sigma_2\sigma_1$. □

Lemma 5.4 *If* $A; S_1; \vartheta \Longrightarrow^* \emptyset; S_2; \vartheta\sigma$ *is a derivation in* $\mathfrak{G}(\mathscr{R})$*, then for all* $(X : \tilde{s} \triangleq \tilde{q}) \in A$*, the hedge* $X\sigma$ *is an* $\mathscr{R}_\mathsf{T}$*-generalization of* $\tilde{s}$ *and* $\tilde{q}$*.*

Proof By soundness of $\mathfrak{G}(\mathscr{R})$, $X\sigma$ is an $\mathscr{R}$-generalization of $\tilde{s}$ and $\tilde{q}$. From Definitions 4.3 and 5.1, it is easy to see that an $\mathscr{R}$-generalization of two hedges is also their $\mathscr{R}_\mathsf{T}$-generalization. □

Lemma 5.5 *If* $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id \Longrightarrow^* \emptyset; S_1; \vartheta \Longrightarrow_\mathsf{R} \emptyset; S_2; \vartheta\sigma$ *is a derivation in* $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ *such that* $X\vartheta$ *is an* $\mathscr{R}_\mathsf{T}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$ *and* $\mathsf{R} \in \{\mathscr{R}\text{-CS3}, \mathscr{R}\text{-CS4}\}$*. Then* $X\vartheta\sigma$ *is an* $\mathscr{R}_\mathsf{T}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$*.*

Proof Let R be $\mathscr{R}$-CS3, transforming $\{x_1 : l \triangleq r, x_2 : l \triangleq r\} \subseteq S_1$ into $\{x_1 : l \triangleq r\} \subseteq S_2$ with the substitution $\sigma = \{x_2 \mapsto x_1\}$. The terms l and r occur in $\tilde{s}_1$ and $\tilde{s}_2$, respectively, so that the corresponding occurrences are abstracted by the same variable in $X\vartheta$. This variable for some pairs of occurrences of l and r is x_1 and $X\vartheta$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$ for some others x_2. Hence, if we replace x_2 with x_1 in $X\vartheta$, the obtained hedge $X\vartheta\sigma$ will be a generalization of $\tilde{s}_1$ and $\tilde{s}_2$.

To prove that after this replacement we still have an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$, we proceed by induction on the maximal depth d of the occurrences of x_2 in $X\vartheta$. This can be done in the same way as in the proof of Lemma 4.10.

Let R be $\mathscr{R}$-CS4, transforming $\{X_1 : l_1, \ldots, l_n \triangleq r_1, \ldots, r_n\}$ into $\{x_1 : l_1 \triangleq r_1, \ldots x_n : l_n \triangleq r_n\}$ with the substitution $\sigma = \{X_1 \mapsto x_1, \ldots, x_n\}$. Since $X\vartheta$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$, X_1 can be adjacent only to a term variable in $X\vartheta$. After replacing it (at any depth in $X\vartheta$) with the hedge of variables $x_1, \ldots, x_n$, these new variables can appear also only next to non-variable terms or term variables. Besides, removing them from $X\vartheta$ corresponds to removal of X_1 from there and, hence, it does not affect the alignment imposed by $\mathscr{R}$. This implies that for $X\vartheta\sigma$ the conditions of the definition of $\mathscr{R}_\mathsf{T}$-generalization are satisfied, i.e., $X\vartheta\sigma$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. □

Now we can prove the soundness theorem for $\mathfrak{G}(\mathscr{R}_\mathsf{T})$:

Theorem 5.6 (Soundness of $\mathfrak{G}(\mathscr{R}_\mathsf{T})$) *If* $\{X : \tilde{s}_1 \triangleq \tilde{s}_2\}; \emptyset; Id \Longrightarrow^* \emptyset; S; \sigma$ *is a derivation in* $\mathfrak{G}(\mathscr{R}_\mathsf{T})$*, then* $X\sigma$ *is an* $\mathscr{R}_\mathsf{T}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$*.*

Proof By Lemma 5.3, every derivation in $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ can be reordered so that first we have only the derivation in $\mathfrak{G}(\mathscr{R})$ and then the rules $\mathscr{R}$-CS3 and $\mathscr{R}$-CS4 are applied. The substitutions computed by the original derivation and by the reordered derivation coincide. Let σ' be the substitution obtained at the end of the subderivation in $\mathfrak{G}(\mathscr{R})$. By Lemma 5.4, $X\sigma'$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. By Lemma 5.5, substitutions introduced by the rules $\mathscr{R}$-CS3 and $\mathscr{R}$-CS4 keep the $\mathscr{R}_\mathsf{T}$-generalization property. Hence, $X\sigma$ is an $\mathscr{R}_\mathsf{T}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. □

The algorithm $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ is also complete:

Theorem 5.7 (Completeness of $\mathfrak{G}(\mathscr{R}_\mathsf{T})$) *Let* $\tilde{q}$ *be an* $\mathscr{R}_\mathsf{T}$*-generalization of* $\tilde{s}_1$ *and* $\tilde{s}_2$*. Then* $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ *computes an* $\mathscr{R}_\mathsf{T}$*-anti-unifier* σ *for* $X : \tilde{s}_1 \triangleq \tilde{s}_2$ *such that* $\tilde{q} \preceq X\sigma$*.*

Proof We prove the theorem on the number n of distinct term variables in $\tilde{q}$. (Since we always assume that the hedges to be generalized are variable disjoint, the variables in $\tilde{q}$ occur neither in $\tilde{s}_1$ nor in $\tilde{s}_2$.) If $n = 0$, then $\tilde{q}$ is an $\mathscr{R}$-generalization of $\tilde{s}_1$ and $\tilde{s}_2$. Then by Theorem 4.12 we can compute an $\mathscr{R}$-anti-unifier σ with the desired property using only the rules in $\mathscr{R}$. But σ is also an $\mathscr{R}_\mathsf{T}$-anti-unifier. Hence, for $n = 0$ the theorem holds.

Now assume the claim is true for any $\mathscr{R}_\mathsf{T}$-generalization with n distinct term variables and prove it for those that contain $n + 1$ distinct term variables. Let y be a term variable that occurs in $\tilde{q}$. Then there are subterms t_1 and t_2 in $\tilde{s}_1$ and $\tilde{s}_2$, respectively, that are generalized by y. Let $\tilde{q}'$ be a hedge obtained from $\tilde{q}$ by replacing each occurrence of y with a fresh variable Y. Then $\tilde{q}'$ is a $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ generalization of $\tilde{s}_1$ and $\tilde{s}_2$. It generalizes every subterm in $\tilde{s}_1$ and $\tilde{s}_2$ in the same way (modulo variable renaming) as $\tilde{q}$ does, except for t_1 and t_2. $\tilde{q}'$ contains n distinct term variables. Therefore, by the induction hypothesis, $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ computes an $\mathscr{R}_\mathsf{T}$-anti-unifier σ' for $X : \tilde{s}_1 \triangleq \tilde{s}_2$ such that $\tilde{q}' \preceq X\sigma'$. That is, there exists a substitution ρ such that $\tilde{q}'\rho = X\sigma'$. Depending on what $Y\rho$ is, we distinguish the following cases:

1. $Y\rho$ is a term variable or a non-variable term. Then $y\rho \preceq Y\rho$, therefore, $\tilde{q} \preceq X\sigma'$ and we can take σ' in the role of σ.
2. $Y\rho$ is a hedge variable Z. Then we extend the derivation with the rule $\mathscr{R}$-CS4 and the substitution $\rho' = \{Z \mapsto z\}$. Then $\tilde{q} \preceq X\sigma'\rho'$ and can take $\sigma'\rho'$ in the role of σ. □

Example 5.8 Let $f(g(a, a), g(b, b), f(g(a), g(a)))$ and $f(g(a, a), f(g(a), g))$ be two given terms and $\mathscr{R}$ be the function computing the set of all longest common subsequences. In Example 5.2 we saw that the $mcg_{\mathscr{R}_\mathsf{T}}$ of these terms consists of two elements. Now we illustrate how the algorithm $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ computes one of these

lggs, the term $f(Z, g(x, x), f(g(a), g(X)))$. We can continue where the derivation in Example 4.4 stopped and perform two more steps:

$$\begin{aligned}
&\emptyset; \{Z : g(a,a) \triangleq \varepsilon, U : b, b \triangleq a, a, X : a \triangleq \varepsilon\};\\
&\qquad \{X_0 \mapsto f(Z, g(U), f(g(a), g(X))), \ldots\} \Longrightarrow_{\mathscr{R}\text{-CS4}}\\
&\emptyset; \{Z : g(a,a) \triangleq \varepsilon, x : b, \triangleq a, y : b \triangleq a, X : a \triangleq \varepsilon\};\\
&\qquad \{X_0 \mapsto f(Z, g(x,y), f(g(a), g(X))), \ldots\} \Longrightarrow_{\mathscr{R}\text{-CS3}}\\
&\emptyset; \{Z : g(a,a) \triangleq \varepsilon, x : b, \triangleq a, X : a \triangleq \varepsilon\};\\
&\qquad \{X_0 \mapsto f(Z, g(x,x), f(g(a), g(X))), \ldots\}
\end{aligned}$$

Example 5.9 We illustrate how the algorithm $\mathfrak{G}(\mathscr{R}_{\mathsf{T}})$ generalizes the terms $f(a_1, a_2, a_3, a_4, a_5)$ and $f(b_1, b_2, b_3, b_4, b_5)$. (Earlier, we mentioned a bad performance of $\mathfrak{G}$ on this input). $\mathscr{R}$ computes the set of all common subsequences of its arguments. There is only one branch in the computation. It is a drastic improvement compared to more than eleven thousand branches in the nonrigid generalization algorithm:

$$\begin{aligned}
&\{X_0 : f(a_1,a_2,a_3,a_4,a_5) \triangleq f(b_1,b_2,b_3,b_4,b_5)\}; \emptyset; Id \Longrightarrow_{\mathscr{R}\text{-Dec-H}}\\
&\{X_1 : a_1,a_2,a_3,a_4,a_5 \triangleq b_1,b_2,b_3,b_4,b_5\}; \{Y_0 : \varepsilon \triangleq \varepsilon, Y_1 : \varepsilon \triangleq \varepsilon\};\\
&\qquad \{X_0 \mapsto Y_0, f(X_1), Y_1\} \Longrightarrow^2_{\mathscr{R}\text{-CS2}}\\
&\{X_1 : a_1,a_2,a_3,a_4,a_5 \triangleq b_1,b_2,b_3,b_4,b_5\}; \emptyset; \{X_0 \mapsto f(X_1)\} \Longrightarrow_{\mathscr{R}\text{-S-H}}\\
&\emptyset; \{X_1 : a_1,a_2,a_3,a_4,a_5 \triangleq b_1,b_2,b_3,b_4,b_5\}; \{X_0 \mapsto f(X_1)\} \Longrightarrow_{\mathscr{R}\text{-CS4}}\\
&\emptyset; \{x_1 : a_1 \triangleq b_1, x_2 : a_2 \triangleq b_2, x_3 : a_3 \triangleq b_3,\\
&\qquad x_4 : a_4 \triangleq b_4, x_5 : a_5 \triangleq b_5\};\\
&\qquad \{X_0 \mapsto f(x_1,x_2,x_3,x_4,x_5), \ldots\}.
\end{aligned}$$

6 Relationship with the Other Anti-Unification Problems

Simple Hedge Anti-Unification Rigid anti-unification can model simple anti-unification considered in Yamamoto et al. [37]. Simple hedges are called those that are linear (no duplicated occurrences of hedge variables) and for each subterm $f(s_1, \ldots, s_n)$ occurring in the hedge at any depth, there is at most one hedge variable among $s_1, \ldots, s_n$. Term variables are not permitted. The following two hedges $a, X, f(Y, g(b))$ and $X, f(Y, f(a, Z))$ are simple, while the hedges $a, X, f(b), Y, a$ and $X, f(X)$ are not. A simple anti-unifier for $f(a), f(a,c), a, b, g(a), g(b)$ and $f(b,a,b), f(b,a,b,c), b, g(a)$ is $f(X), f(Y,c), Z, g(U)$.

We can use the $\mathfrak{G}(\mathscr{R})$ algorithm without the rule $\mathscr{R}$-CS1 to compute simple generalizations. The rigidity function appropriate for this case should return a longest common subsequence of its arguments, composed by concatenating the non-overlapping longest common prefix and the longest common suffix. For instance, in

case of words *ffabgg* and *ffbg*, $\mathscr{R}$ should return *ffg*, because it is a concatenation of the longest common prefix *ff* and the longest common suffix *g*. If the longest common prefix and the longest common suffix overlap, then the input words are the same and $\mathscr{R}$ returns that word. Now we see how the algorithm with this rigidity function computes a simple anti-unifier:[3]

$$
\begin{aligned}
&\{X_0 : f(a),\ f(a,c), a, b, g(a), g(b) \triangleq f(b, a, b),\ f(b, a, b, c), b, g(a)\};\\
&\qquad\qquad \emptyset;\ Id \Longrightarrow_{\mathscr{R}\text{-Dec-H}}\\
&\{X : a \triangleq b, a, b, Y_0 : a, c \triangleq b, a, b, c, U : a \triangleq b\};\\
&\{Z : a, b, g(a) \triangleq b, \}; \{X_0 \mapsto f(X),\ f(Y_0),\ Z, g(U)\} \Longrightarrow_{\mathscr{R}\text{-S-H}}\\
&\{Y_0 : a, c \triangleq b, a, b, c, U : a \triangleq b\};\\
&\{X : a \triangleq b, a, b, Z : a, b, g(a) \triangleq b\}; \{X_0 \mapsto f(X),\ f(Y_0),\ Z, g(U)\} \Longrightarrow_{\mathscr{R}\text{-Dec-H}}\\
&\{Y : a \triangleq b, a, b, Y_1 : \varepsilon \triangleq \varepsilon, U : a \triangleq b\};\\
&\{X : a \triangleq b, a, b, Z : a, b, g(a) \triangleq b\};\\
&\{X_0 \mapsto f(X),\ f(Y, c(Y_1)),\ Z, g(U), \ldots\} \Longrightarrow_{\mathscr{R}\text{-S-H}}\\
&\{Y_1 : \varepsilon \triangleq \varepsilon, U : a \triangleq b\};\\
&\{X : a \triangleq b, a, b, Y : a \triangleq b, a, b, Z : a, b, g(a) \triangleq b\};\\
&\{X_0 \mapsto f(X),\ f(Y, c(Y_1)),\ Z, g(U), \ldots\} \Longrightarrow_{\mathscr{R}\text{-Dec-H}}\\
&\{U : a \triangleq b\};\\
&\{Y_1 : \varepsilon \triangleq \varepsilon, X : a \triangleq b, a, b, Y : a \triangleq b, a, b, Z : a, b, g(a) \triangleq b\};\\
&\{X_0 \mapsto f(X),\ f(Y, c(Y_1)),\ Z, g(U), \ldots\} \Longrightarrow_{\mathscr{R}\text{-CS2}}\\
&\{U : a \triangleq b\}; \{X : a \triangleq b, a, b, Y : a \triangleq b, a, b, Z : a, b, g(a) \triangleq b\};\\
&\{X_0 \mapsto f(X),\ f(Y, c),\ Z, g(U), \ldots\} \Longrightarrow_{\mathscr{R}\text{-S-H}}\\
&\emptyset; \{X : a \triangleq b, a, b, Y : a \triangleq b, a, b, Z : a, b, g(a) \triangleq b, U : a \triangleq b\};\\
&\{X_0 \mapsto f(X),\ f(Y, c),\ Z, g(U), \ldots\}.
\end{aligned}
$$

Word Anti-Unification It is interesting to see how our approach to hedge generalization compares to existing works on its special case, word anti-unification. The algorithm for the latter defined in Cicekli and Cicekli [11] computes not all, but only a specific word generalization. The input words w_1 and w_2 should satisfy the following properties: (i) Their longest common subsequence w should not share a letter with the words $w_1 - w$ and $w_2 - w$ (called the differences), obtained from w_1 and w_2, respectively, by deleting the subsequence w, and (ii) $w_1 - w$ and $w_2 - w$ should not share a letter. If w_1 and w_2 do not satisfy these properties, they are abstracted by a hedge variable. Otherwise the generalization is computed by keeping w in it and generalizing the pieces of difference words between the letters of w in w_1 and w_2 in

[3] As in Example 4.7, we contract applications of $\mathscr{R}$-Dec-H and $\mathscr{R}$-CS2 rules into one $\mathscr{R}$-Dec-H step.

the usual way. These pieces are processed to see whether there are differences that can be abstracted by the same variable.

This algorithm can be modeled with a combination of rigid and complete algorithms in our case. We define a rigidity function $\mathscr{R}$ such that for words that satisfy the conditions (i) and (ii), it returns the set containing their longest common subsequence. (There can be only one.) If the conditions are violated, then the empty set is returned. After the $\mathfrak{G}(\mathscr{R})$ algorithm computes a terminal system $\emptyset, S, \sigma$, we continue with the algorithm $\mathfrak{G}$ starting from the system $S, \emptyset, \sigma$. The reason of such a combination is that in S the AUPs are between difference words. According to Cicekli and Cicekli [11], they are generalized differently than the original words. They should be decomposed to see whether there are two AUPs that can be generalized with the same variable. We can model this kind of generalization by the algorithm $\mathfrak{G}$.

In Biere [7], the notion of ε-free generalization is introduced. A word w is an ε-free generalization of w_1 and w_2 if for all σ_1, σ_2 with $w\sigma_1 = w_1$ and $w\sigma_2 = w_2$, there is no variable X in w such that $X\sigma_1 = \varepsilon$ or $X\sigma_2 = \varepsilon$ holds. Rigid anti-unification models computation of ε-free generalization, if we choose the rigidity function to return longest common subsequences so that the alignments $a[i_1, j_1] \cdots a[i_n, j_n]$ computed by the function satisfy the property: for all $1 \leq k < n$, $i_{k+1} = i_k + 1$ iff $j_{k+1} = j_k + 1$. That means that consecutive letters of the computed subsequence either occur consecutively in both of the original words, or are separated in both of them by at least one letter.

Standard Anti-Unification The standard anti-unification over ranked terms [31, 32] can be modeled by $\mathscr{R}_{\mathsf{T}}$-rigid anti-unification, choosing $\mathscr{R}$ as the function that returns a singleton set $\mathscr{R}(w_1, w_2) = \{a_1[i_1, i_1] \cdots a_n[i_n, i_n]\}$, where $a_1 \cdots a_n$ is the longest common subsequence of w_1 and w_2 such that all a_is occur at the same positions in w_1 and w_2.

Associative Anti-Unification with Unit Anti-unification with an associative function symbol (A) and a unit element (U) has been studied in Alpuente et al. [2]. AU anti-unification with constants (where terms are built over associative function symbol, unit element, and free constants) can be directly modeled as unranked anti-unification, using the algorithm $\mathfrak{G}$ to compute generalizations.

Example 6.1 Let f be a binary associative function symbol, e the unit element for f (i.e., $f(t, e) = f(e, t) = t$ for all t), and a, b, and c free constants. The AU anti-unification problem $x_0 : f(f(a, b), f(c, f(b, c))) \triangleq a$ can be modeled as an unranked anti-unification problem $X_0 : a, b, c, b, c \triangleq a$. The algorithm $\mathfrak{G}$ computes its generalization a, X, Y, X, Y, from which we can reconstruct the corresponding AU generalization in the flattened form $f(a, x, y, x, y)$.

If, in addition, there are free function symbols in the terms as well (general AU anti-unification), then we need a combination of the rules from $\mathfrak{G}$ and $\mathfrak{G}(\mathscr{R}_{\mathsf{T}})$. The rigidity function in $\mathfrak{G}(\mathscr{R}_{\mathsf{T}})$ is the same as the one for modeling the standard anti-unification above. The combination is not straightforward, since we need to distinguish between AUPs obtained by decomposing terms with the associative head, from the AUPs obtained by decomposing terms with free heads. Differences and

similarities between occurrences of the unit element and the empty hedge requires also a special treatment. One can add extra rules to the algorithms for these special cases, but we do not go into detail here, as it is merely a technical exercise and does not give a new insight into relationship between these problems.

7 Discussion on Applications

It is not our intention to give here an exhaustive list of application areas of various forms and kinds of anti-unification. We just mention few representative ones, such as reasoning by analogy [19], machine learning [4], program synthesis [17, 35], program verification [28]. What we discuss in this section is one potentially interesting application of this technique in clone detection, indicating the ways how $\mathscr{R}_{\mathsf{T}}$-generalization can help in the process of detecting (software code) clones.

Having said that, we would like to emphasize that clone detection by anti-unification is not the main topic of this paper. It can be a subject of separate research. Here we just try to briefly indicate some possibilities of application of rigid anti-unification in detecting clones.

Clone detection is an active research topic since clones are considered to be a significantly problematic issue for software maintenance. Studies show that from 5 to 20 % of software systems can contain duplicated code. Due to various complications such duplicated pieces cause, it is widely agreed that clone detection an important part of software analysis. The survey papers [33, 34] give a detailed characterization of code duplication reasons and drawbacks, introduce clone types, describe and evaluate clone detection process and techniques, and list open problems in clone detection research. The proposed classification distinguishes four types of clones:

Type I: Identical code fragments except for variations in whitespace, layout and comments.
Type II: Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.
Type III: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.
Type IV: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

Complexity and sophistication in detecting such clones increases from Type I through Type IV with Type IV being the highest. (Although it does not mean that Type IV contains other types as special cases.)

Some clone detection techniques are based on tree representation of the code, like parse trees, abstract syntax trees, or an XML form of abstract syntax trees; see, e.g., Baxter et al. [6], Evans et al. [14], Koschke et al. [18], Yang [38], Wahler et al. [36], for some of the works that follow this approach. We assume that the code is represented in a structural form that can be encoded with unranked terms (or hedges). We keep the representation abstract, without specifying what exactly this structural form is. This helps to make the technique independent of the underlying programming language.

Usually, clone detection tools first preprocess the code, then find potential clone candidates, and, finally, analyze them to return actual clones. One can employ the $\mathscr{R}_{\mathsf{T}}$-generalization algorithm in the process of finding potential code clones. Further analysis can be based on various measures, like, e.g., on the size of the generalization, or on the maximal length of a nonvariable hedge in the generalization, etc. Although we are not concerned with this part, by choosing appropriate $\mathscr{R}$ we can anticipate this last filtering process. The choice of the $\mathscr{R}$ depends on what is considered as interesting clone.

$\mathscr{R}_{\mathsf{T}}$-anti-unification can be useful as an ingredient for a clone detection technique/tool for types I-III. We illustrate it on an example composed from the taxonomy of editing scenarios for different clone types, described in Roy et al. [34].

In Fig. 1, we show a program (on the top) and its three clones (in three columns below). The first one is the clone of type II, the second and the third ones are clones of type III.

We show now how rigid anti-unification can be used to find similarities between these pieces of code and the original code. The rigid lggs will show how similar these pieces are and at which parts they differ. A clone detection tool then can use this information to conclude that clones have been detected.

First, we have to represent the pieces of code above as unranked terms. For better comprehension, we write these terms in several lines, following the structure of the code they abstract. The representation is shown in Fig. 2, where the term t corresponds to the original code. Its modified copies are denoted, respectively, with r_1, r_2, and r_3. In the first two, the parts that correspond to the modified pieces of code are written in bold face.

Let $\mathscr{R}$ compute the set of longest common subsequences. We choose it because it captures the idea that the clones have a lot in common. Such an $\mathscr{R}$ is supposed to draw out as much as possible of the common statements from two pieces of code. Both $\mathscr{R}$- and $\mathscr{R}_{\mathsf{T}}$-generalizations can be used to compute clone candidates. We show here the $\mathscr{R}_{\mathsf{T}}$-generalizations as they provide more precise results.

For r_1 and r_2, the sets of their lggs with t are singleton sets, depicted in Fig. 3. These lggs show that similarities between r_1 and t and between r_2 and t are quite significant. It indicates that the corresponding code pieces are clones.

In case of r_3, which is obtained from t by deleting an argument (i.e., a line in the code), we have two lggs shown in Fig. 4. Among them, we say that the first one is

```
                        void sumProd(int n) {
                        float sum=0.0; //C1
                        float prod =1.0;
                        for (int i=1; i<=n; i++)
                           {sum=sum + i;
                           prod = prod * i;
                           foo(sum, prod); }}

void sumProd(int n) {    void sumProd(int n) {    void sumProd(int n) {
float sum=0.0; //C1      float sum=0.0; //C1      float sum=0.0; //C1
float prod =1.0;         float prod =1.0;         float prod =1.0;
for (int i=1; i<=n; i++) for (int i=1; i<=n; i++) for (int i=1; i<=n; i++)
   {sum=sum + (i*i);        {sum=sum + i;            {sum=sum + i;
   prod = prod*(i*i);       prod = prod*i;           //line deleted
   foo(sum,prod); }}        foo(sum,prod,n); }}      foo(sum,prod); }}
```

Fig. 1 A program and its clones

$$
\begin{array}{l}
sumProd(\\
\quad input(type(int), n), \\
\quad returnType(void), \\
\quad =(type(float), n, 0.0), \\
\quad =(type(float), prod, 1.0), \\
\quad for(=(type(int), i, 1), \\
\qquad \leq (i, n), {+}{+}(i), \\
\qquad =(sum, +(sum, i)), \\
\qquad =(prod, *(prod, i)), \\
\qquad foo(sum, prod)))
\end{array}
$$

The term t.

$$
\begin{array}{l}
sumProd(\\
\quad input(type(int), n), \\
\quad returnType(void), \\
\quad =(type(float), n, 0.0), \\
\quad =(type(float), prod, 1.0), \\
\quad for(=(type(int), i, 1), \\
\qquad \leq (i, n), {+}{+}(i), \\
\qquad =(sum, +(sum, *(\mathbf{i}, \mathbf{i}))), \\
\qquad =(prod, *(prod, *(\mathbf{i}, \mathbf{i}))), \\
\qquad foo(sum, prod)))
\end{array}
$$

The term r_1.

$$
\begin{array}{l}
sumProd(\\
\quad input(type(int), n), \\
\quad returnType(void), \\
\quad =(type(float), n, 0.0), \\
\quad =(type(float), prod, 1.0), \\
\quad for(=(type(int), i, 1), \\
\qquad \leq (i, n), {+}{+}(i), \\
\qquad =(sum, +(sum, i)), \\
\qquad =(prod, *(prod, i)), \\
\qquad foo(sum, prod, \mathbf{n})))
\end{array}
$$

The term r_2.

$$
\begin{array}{l}
sumProd(\\
\quad input(type(int), n), \\
\quad returnType(void), \\
\quad =(type(float), n, 0.0), \\
\quad =(type(float), prod, 1.0), \\
\quad for(=(type(int), i, 1), \\
\qquad \leq (i, n), {+}{+}(i), \\
\qquad =(sum, +(sum, i)), \\
\\
\qquad foo(sum, prod)))
\end{array}
$$

The term r_3.

Fig. 2 Representing the program and its clones as unranked terms

a better generalization, because it preserves the common structure better than the other.

The standard anti-unification [31, 32] has already been considered for computing software clones by Bulychev and Minea [8], Bulychev et al. [9], Li and Thompson [27], detecting mostly clones of types I and II. However, we think that parametrized anti-unification over unranked terms offers more flexibility in finding clone

Fig. 3 $mcg_{\mathscr{R}_\mathsf{T}}(t \triangleq r_1)$ and $mcg_{\mathscr{R}_\mathsf{T}}(t \triangleq r_2)$

$$
\begin{array}{l}
mcg_{\mathscr{R}_\mathsf{T}}(t \triangleq r_1) = \\
\quad \{sumProd(\\
\qquad input(type(int), n), \\
\qquad returnType(void), \\
\qquad =(type(float), n, 0.0), \\
\qquad =(type(float), prod, 1.0), \\
\qquad for(=(type(int), i, 1), \\
\qquad\quad \leq (i, n), {+}{+}(i), \\
\qquad\quad =(sum, +(sum, \mathbf{x})), \\
\qquad\quad =(prod, *(prod, \mathbf{x})), \\
\qquad\quad foo(sum, prod)))\}
\end{array}
$$

$$
\begin{array}{l}
mcg_{\mathscr{R}_\mathsf{T}}(t \triangleq r_2) = \\
\quad \{sumProd(\\
\qquad input(type(int), n), \\
\qquad returnType(void), \\
\qquad =(type(float), n, 0.0), \\
\qquad =(type(float), prod, 1.0), \\
\qquad for(=(type(int), i, 1), \\
\qquad\quad \leq (i, n), {+}{+}(i), \\
\qquad\quad =(sum, +(sum, i)), \\
\qquad\quad =(prod, *(prod, i)), \\
\qquad\quad foo(sum, prod, \mathbf{X})))\}
\end{array}
$$

$$
\begin{array}{lcll}
mcg_{\mathscr{R}_{\mathsf{T}}}(t \triangleq r_3) & = & \{sumProd(& , \quad sumProd(\\
 & & \quad input(type(int),n), & \quad input(type(int),n), \\
 & & \quad returnType(void), & \quad returnType(void), \\
 & & \quad =(type(float),n,0.0), & \quad =(type(float),n,0.0), \\
 & & \quad =(type(float),prod,1.0), & \quad =(type(float),prod,1.0), \\
 & & \quad for(=(type(int),i,1), & \quad for(=(type(int),i,1), \\
 & & \qquad \leq(i,n),++(i), & \qquad \leq(i,n),++(i), \\
 & & \quad =(sum,+(sum,i)), & \quad \mathbf{X}, \\
 & & \quad \mathbf{X}, & \quad =(\mathbf{x},\mathbf{y}), \\
 & & \quad foo(sum,prod))) & \quad foo(sum,prod)))\}
\end{array}
$$

Fig. 4 $mcg_{\mathscr{R}_{\mathsf{T}}}(t \triangleq r_3)$

candidates. It helps to detect inserted or deleted pieces of code, which is necessary for clones of type III. Another advantage of this approach is that it is modular, where most of the computations are performed on strings. It may combine advantages of fast textual and precise structural techniques. For many interesting string relations (e.g., longest common subsequence, longest common substring, sequence alignment, etc.), there exist efficient algorithms that also scale well for large data [15]. Hence, one can take advantage using these off-the-shelf methods when computing clone candidates by $\mathscr{R}$-generalization.

Yet another advantage of using $\mathscr{R}$-generalizations in clone detection is that it works on unranked terms that are natural abstractions of XML documents. How to detect clones well in generated XML/HTML is mentioned as one of the open problems in clone detection research in Roy and Cordy [33]. A detection technique that uses $\mathscr{R}$-generalization would be an interesting approach to this problem.

Moreover, from the clones computed by $\mathscr{R}$-generalization (anti-unification, in general) one can extract a procedure. This process has a use in code refactoring. The clones can be replaced by the procedure calls, properly instantiated by the substitution that gives, from the computed $\mathscr{R}$-generalization, the clone it generalizes. As we saw in Example 4.7, these substitutions are easily extracted from the store. In general, while anti-unifiers reflect similarities between two inputs, the data in the store can be used to identify differences between them (i.e., between inputs). The latter provides for unranked trees a functionality similar, for instance, to one of the well-known comparison utilities (e.g., diff, cmp, fc) that compare the contents of files, finding common contents and differences in them.

We proved properties of $\mathscr{R}$-generalization for a generic $\mathscr{R}$, i.e., for the entire class of rigidity functions. Specializing $\mathscr{R}$ with a particular function, we obtain a specific instance of $\mathscr{R}$-generalization. We saw how certain known generalization problems fall into the class of specific instances of $\mathscr{R}$-generalization in this way.

Some applications (e.g., structural clustering of XML documents) may require computation of lggs for more than two hedges, i.e., given the hedges $\tilde{s}_1, \ldots, \tilde{s}_n$, compute a hedge $\tilde{s}$ that is more general than any of the given hedges and less general among those with the same property. For such cases, definition of rigid generalizations should be made more generic, permitting alignments and rigidity functions to be defined for any number ($\geq$2) of arguments. Furthermore, we have

to modify the rules of the algorithms $\mathfrak{G}(\mathscr{R})$ and $\mathfrak{G}(\mathscr{R}_\mathsf{T})$, permitting there tuples $X : \tilde{s}_1 \triangleq \tilde{s}_2 \triangleq \cdots \triangleq \tilde{s}_n$ instead of triples $X : \tilde{s} \triangleq \tilde{q}$. We do not intend to go into the details of such extended algorithms. (Their properties are not difficult to prove along the lines of proofs for the case with two hedges.) But we would like to emphasize that it is indeed necessary to have these extensions of $\mathfrak{G}(\mathscr{R})$ and $\mathfrak{G}(\mathscr{R}_\mathsf{T})$ for tuples because, in general, they can not be replaced by iterative applications of $\mathfrak{G}(\mathscr{R})$ or $\mathfrak{G}(\mathscr{R}_\mathsf{T})$. There exist rigidity functions for which rigid generalizations of several terms computed at once might differ from the rigid generalizations computed iteratively. The following example illustrates this:

Example 7.1 Let $\mathscr{R}$ stand for longest common subsequence. Then $f(X, c, Y)$ is an $\mathscr{R}$-lgg for three terms $f(a, b, c)$, $f(c, a, b)$ and $f(c)$. However, if we proceed step by step with the $\mathfrak{G}(\mathscr{R})$ algorithm, we obtain $f(Z)$ as their generalization: The only (modulo $\simeq$) $\mathscr{R}$-lgg of $f(a, b, c)$ and $f(c, a, b)$ is $f(X, a, b, Y)$, and then for this term and $f(c)$ we get $f(Z)$.

On the other hand, there exist also rigidity functions for which "computation of generalizations at once" can be replaced by their iterative computation. For instance, the rigidity function used for modeling the standard anti-unification has such a property. Also, the algorithm $\mathfrak{G}$ can be used iteratively to compute generalizations of several terms.

8 Final Comments

We have presented anti-unification algorithms for unranked terms and hedges, starting from a minimal complete one and then designing a more efficient and flexible version for computing only rigid anti-unifiers. Furthermore, we made rigid generalizations more precise by generalizing certain hedges with sequences of term variables instead of abstracting them by a single hedge variable. We also indicated possible applications in code clone detection.

The rigid generalization algorithm $\mathfrak{G}(\mathscr{R})$ has been implemented (without the minimization step) in Java and is available from http://www.risc.jku.at/projects/stout/. One can use it online via a Web interface or download sources freely from the same location. The provided rigidity functions are those for computing longest common subsequences and longest common substrings (both with and without minimal length restriction).

There are a couple of possible directions in future work. One option is to bring in certain higher-order features that can help to further improve the precision of generalizations and detect similarities in different levels. Other interesting directions would be to perform unranked anti-unification in a sorted setting or on compressed terms.

Acknowledgements We thank Santiago Escobar for a useful discussion and Alexander Baumgartner for implementing the $\mathfrak{G}(\mathscr{R})$ algorithm.

References

1. Aït-Kaci, H., Sasaki, Y.: An axiomatic approach to feature term generalization. In: Raedt, L.D., Flach, P.A. (eds.) ECML. Lecture Notes in Computer Science, vol. 2167, pp. 1–12. Springer, Heidelberg (2001)
2. Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P.: A modular equational generalization algorithm. In: Hanus, M. (ed.) LOPSTR. Lecture Notes in Computer Science, vol. 5438, pp. 24–39. Springer, Heidelberg (2008)
3. Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P.: Order-sorted generalization. Electr. Notes Theor. Comput. Sci. **246**, 27–38 (2009)
4. Armengol, E., Plaza, E.: Bottom-up induction of feature terms. Mach. Learn. **41**(3), 259–294 (2000)
5. Baader, F.: Unification, weak unification, upper bound, lower bound, and generalization problems. In: Book, R.V. (ed.) RTA. Lecture Notes in Computer Science, vol. 488, pp. 86–97. Springer, Heidelberg (1991)
6. Baxter, I.D., Yahin, A., de Moura, L.M., Sant'Anna, M., Bier, L.: Clone detection using abstract syntax trees. In: ICSM, pp. 368–377 (1998)
7. Biere, A.: Normalisation, unification and generalisation in free monoids. Master's thesis, University of Karlsruhe (in German, 1993)
8. Bulychev, P., Minea, M.: An evaluation of duplicate code detection using anti-unification. In: Proc. 3rd International Workshop on Software Clones (2009)
9. Bulychev, P.E., Kostylev, E.V., Zakharov, V.A.: Anti-unification algorithms and their applications in program analysis. In: Pnueli, A., Virbitskaite, I., Voronkov, A. (eds.) Ershov Memorial Conference. Lecture Notes in Computer Science, vol. 5947, pp. 413–423. Springer, Heidelberg (2009)
10. Burghardt, J.: E-generalization using grammars. Artif. Intell. **165**(1), 1–35 (2005)
11. Cicekli, I., Cicekli, N.K.: Generalizing predicates with string arguments. Appl. Intell. **25**(1):23–36 (2006)
12. Cirstea, H., Kirchner, C., Kopetz, R., Moreau, P.E.: Anti-patterns for rule-based languages. J. Symb. Comput. **45**(5), 523–550 (2010)
13. Delcher, A.L., Kasif, S.: Efficient parallel term matching and anti-unification. J. Autom. Reasoning **9**(3), 391–406 (1992)
14. Evans, W.S., Fraser, C.W., Ma, F.: Clone detection via structural abstraction. Softw. Qual. J. **17**(4), 309–330 (2009)
15. Gusfield, D.: Algorithms on Strings, Trees, and Sequences—Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
16. Huet, G.: Résolution d'équations dans des langages d'ordre $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII (1976)
17. Kitzelmann, E., Schmid, U.: Inductive synthesis of functional programs: an explanation based generalization approach. J. Mach. Learn. Res. **7**, 429–454 (2006)
18. Koschke, R., Falke, R., Frenzel, P.: Clone detection using abstract syntax suffix trees. In: WCRE, pp. 253–262. IEEE Computer Society (2006)
19. Krumnack, U., Schwering, A., Gust, H., Kühnberger, K.U.: Restricted higher-order anti-unification for analogy making. In: Orgun, M.A., Thornton, J. (eds.) Australian Conference on Artificial Intelligence. Lecture Notes in Computer Science, vol. 4830, pp. 273–282. Springer, Heidelberg (2007)
20. Kutsia, T.: Solving equations with sequence variables and sequence functions. J. Symb. Comput. **42**(3), 352–388 (2007)
21. Kutsia, T.: Flat matching. J. Symb. Comput. **43**(12), 858–873 (2008)
22. Kutsia, T., Marin, M.: Matching with regular constraints. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR. Lecture Notes in Computer Science, vol. 3835, pp. 215–229. Springer, Heidelberg (2005)
23. Kutsia, T., Marin, M.: Order-sorted unification with regular expression sorts. In: Lynch, C. (ed.) RTA, Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, LIPIcs, vol. 6, pp. 193–208 (2010)
24. Kutsia, T., Levy, J., Villaret, M.: Sequence unification through currying. In: Baader, F. (ed.) RTA. Lecture Notes in Computer Science, vol. 4533, pp. 288–302. Springer, Heidelberg (2007)
25. Kutsia, T., Levy, J., Villaret, M.: On the relation between context and sequence unification. J. Symb. Comput. **45**(1), 74–95 (2010)
26. Kutsia, T., Levy, J., Villaret, M.: Anti-unification for unranked terms and hedges. In: Schmidt-Schauß, M. (ed.) RTA. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, LIPIcs, vol. 10, pp. 219–234 (2011)

27. Li, H., Thompson, S.J.: Similar code detection and elimination for erlang programs. In: Carro, M., Peña, R. (eds.) PADL. Lecture Notes in Computer Science, vol. 5937, pp. 104–118. Springer, Heidelberg (2010)
28. Lu, J., Mylopoulos, J., Harao, M., Hagiya, M.: Higher order generalization and its application in program verification. Ann. Math. Artif. Intell. **28**(1–4), 107–126 (2000)
29. Pfenning, F.: Unification and anti-unification in the calculus of constructions. In: LICS, pp. 74–85. IEEE Computer Society (1991)
30. Plaza, E.: Cases as terms: a feature term approach to the structured representation of cases. In: Veloso, M.M., Aamodt, A. (eds.) ICCBR. Lecture Notes in Computer Science, vol. 1010, pp. 265–276. Springer, Heidelberg (1995)
31. Plotkin, G.D.: A note on inductive generalization. Mach. Intell. **5**(1), 153–163 (1970)
32. Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. Mach. Intell. **5**(1), 135–151 (1970)
33. Roy, C.K., Cordy, J.R.: A Survey of Software Clone Detection Research. Tech. rep., School of Computing, Queen's University at Kingston, ON, Canada (2007)
34. Roy, C.K., Cordy, J.R., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: a qualitative approach. Sci. Comput. Program **74**(7), 470–495 (2009)
35. Schmid, U.: Inductive synthesis of functional programs, universal planning, folding of finite programs and schema abstraction by analogical reasoning. In: Lecture Notes in Computer Science, vol. 2654. Springer, Heidelberg (2003)
36. Wahler, V., Seipel, D., von Gudenberg, J.W., Fischer, G.: Clone detection in source code by frequent itemset techniques. In: SCAM, pp. 128–135. IEEE Computer Society Press, Los Alamitos (2004)
37. Yamamoto, A., Ito, K., Ishino, A., Arimura, H.: Modelling semi-structured documents with hedges for deduction and induction. In: Rouveirol, C., Sebag, M. (eds.) ILP. Lecture Notes in Computer Science, vol. 2157, pp. 240–247. Springer, Heidelberg (2001)
38. Yang, W.: Identifying syntactic differences between two programs. Softw. Pract. Exper. **21**(7), 739–755 (1991)