Unranked Second-Order Anti-Unification

Alexander Baumgartner and Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria
{abaumgar,kutsia}@risc.jku.at

Abstract. In this work we study anti-unification for unranked hedges, permitting context and hedge variables. Hedges are sequences of unranked terms. The anti-unification problem of two hedges $\tilde{s}$ and $\tilde{q}$ is concerned with finding their generalization, a hedge $\tilde{g}$ such that both $\tilde{s}$ and $\tilde{q}$ are substitution instances of $\tilde{g}$. Second-order power is gained by using context variables to generalize vertical differences at the input hedges. Hedge variables are used to generalize horizontal differences. An anti-unification algorithm is presented, which computes a generalization of input hedges and records all the differences. The computed generalizations are least general among a certain class of generalizations.

1 Introduction

The anti-unification problem for two terms t_1 and t_2 requires finding their generalization: A term such that both t_1 and t_2 are instances of it under some substitutions. The interesting generalizations are least general ones (lggs). Anti-unification algorithms are supposed to compute lggs.

In 1970, Plotkin [20] and Reynolds [21] independently came up with essentially the same anti-unification algorithm. It was designed for first-order ranked terms (i.e., 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., [1, 2, 5, 7, 10, 13, 19]) and from the point of view of different applications (e.g., [4, 9, 11, 15, 18, 17, 22]).

In this paper, we consider anti-unification for hedges, which are finite sequences of unranked terms. Such terms are constructed from function symbols that do not have a fixed arity. We permit two kinds of variables: first-order, for hedges, and second-order, for contexts. Contexts that we consider here are hedges with a single occurrence of the distinguished symbol "hole". They are functions which can apply to another context or to a hedge, which are then "plugged" in the place of the hole.

Some applications of anti-unification indeed require higher-order features. For instance, reuse of proofs in program verification needs anti-unification with higher-order variables [18]. A restricted use of higher-order variables in generalizations turned out to be helpful for analogy making with Heuristic-Driven Theory Projection [15]. Anti-unification with combinator terms plays a role in replaying program derivations [12].

First-order anti-unification for ranked terms has been used to detect software code clones [9, 17]. It helps to achieve high-precision for clones obtained, essentially, by renaming and reformatting, but ranked anti-unification is not strong enough to detect

clones obtained by omitting/inserting pieces of statements in the code. Unranked anti-unification can detect similarities not only between renamed parts of a hedge, but also between parts which differ from each other by inserting or omitting subparts, as was indicated in [16]. These features can be useful also for comparison of XML documents, which can be abstracted by unranked trees.

However, one important restriction of existing hedge anti-unification algorithms, such as, e.g., [16, 8, 23], is that the languages used in these algorithms do not permit higher-order variables. This imposes a natural restriction on solutions: The computed lggs do not reflect similarities between input hedges, if those similar pieces are located under distinct heads or at different depths. For instance, $f(a, b)$ and $g(h(a, b))$ are generalized by a single variable, although both terms contain a and b and a more natural generalization could be, e.g. $X(a, b)$, where X is a higher-order variable. In applications, it is often desirable to detect these similarities.

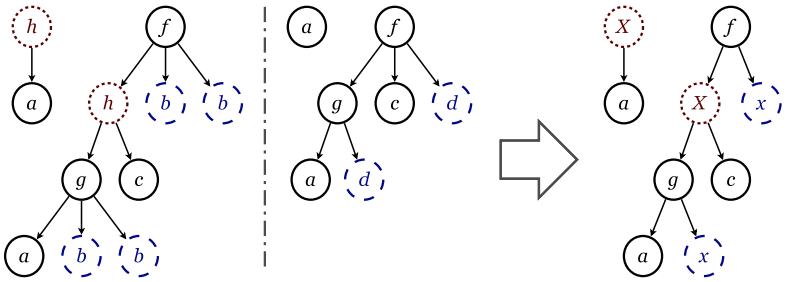
This is the problem we address here, permitting the use of context variables to abstract vertical differences between trees, and hedge variables used to abstract horizontal differences. The algorithm described in this paper first constructs a "skeleton" of a generalization of the input hedges, which corresponds to a hedge embedded into each of the input hedges. Next, it inserts context and/or hedge variables into the skeleton, which are supposed to uniformly generalize (vertical and horizontal) differences between input hedges, to obtain an lgg (with respect to the given skeleton). The skeleton computation function is the parameter of the algorithm: One can compute an lgg which contains, for instance, a constrained longest common subforest [3], or an agreement subhedge/subtree [14] of the input hedges.

In this paper we focus on the step of computing an lgg of two hedges, when the skeleton is already constructed. We assume that the latter is given in the form of an admissible alignment, which is a certain sequence of symbols occurring in both hedges, together with the positions these symbols occur in. We need to restrict variable occurrences in the generalization to guarantee that for each admissible alignment a unique lgg is computed, leading to the notion of rigid generalizations. We develop an algorithm which takes two hedges and an admissible alignment and computes a rigid lgg of the hedges with respect to that alignment. The computed lgg is unique modulo variable renaming. Moreover, we can return not only the generalization, but also the differences between the input hedges, which tells us how one can obtain the original hedges from the generalization. The algorithm runs in quadratic time and requires linear space with respect to the size of the input. This result means that, for instance, if the skeleton is a constrained longest common subhedge of the input hedges in the sense of [24], then both skeleton and generalization computation can be done in quadratic time, because the time complexity of computing a constrained lcs is quadratic.

In some cases, the skeleton can be constructed in multiple ways, giving rise to several admissible alignments. It requires that the generalizations computed for each alignment should be compared to each other, to make the obtained set minimal. This problem requires matching with context and hedge variables in the minimization step and goes beyond the scope of this paper.

A prototype implementation of the algorithm is available from `http://www.risc.jku.at/projects/stout/software/urauc.php`.

Example 1. The hedge $(X(a), f(X(g(a,x),c),x))$ is a generalization of two hedges $(h(a), f(h(g(a,b,b),c),b,b))$ and $(a, f(g(a,d),c,d))$. Dotted and dashed nodes indicate differences, while the solid ones form the admissible alignment. The first hedge can be obtained from the generalization by replacing the context variable X with the context $h(\circ)$ and the hedge variable x with the hedge (a,b). To obtain the second hedge, we need to replace X with the hole (i.e., to eliminate X) and to replace x by d.



2 Preliminaries

Given pairwise disjoint countable sets of unranked function symbols $\mathcal{F}$ (symbols without fixed arity), hedge variables $\mathcal{V}_{\mathsf{H}}$, unranked context variables $\mathcal{V}_{\mathsf{C}}$, and a special symbol $\circ$ (the hole), we define *terms*, *hedges*, and *contexts* by the following grammar:

$$\begin{array}{lll} t & := & x \mid f(\tilde{s}) \mid X(\tilde{s}) \qquad \text{(terms)} \\ \tilde{s} & := & t_1, \ldots, t_n \qquad \text{(hedges)} \\ \tilde{c} & := & \tilde{s}_1, \circ, \tilde{s}_2 \mid \tilde{s}_1, f(\tilde{c}), \tilde{s}_2 \mid \tilde{s}_1, X(\tilde{c}), \tilde{s}_2 \qquad \text{(contexts)} \end{array}$$

where $x \in \mathcal{V}_{\mathsf{H}}$, $f \in \mathcal{F}$, $X \in \mathcal{V}_{\mathsf{C}}$, and $n \geqslant 0$.

Hedges are finite sequences of terms, constructed over $\mathcal{F}$ and $\mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$. A term can be seen as a singleton hedge. A context can be seen as a hedge over $\mathcal{F} \cup \{\circ\}$ and $\mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$, where the hole occurs exactly once. A singleton context is then a term over $\mathcal{F} \cup \{\circ\}$ and $\mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$ with a single hole in it. To improve readability, we put non-singleton hedges and contexts between parenthesis.

We use the letters x, y, z for hedge variables and X, Y, Z for context variables. By f, g, h, a, b, c, d, e we denote function symbols, by $\tilde{s}, \tilde{q}, \tilde{r}, \tilde{g}, \tilde{h}$ hedges, by $\tilde{c}, \tilde{d}$ arbitrary contexts and by $\dot{c}, \dot{d}$ singleton contexts. We use ϕ, ψ for a context variable or a function symbol. The empty hedge is denoted by ϵ. Terms of the form $a(\epsilon)$ are written as just a. Examples of a term, a hedge, and a context are, respectively, $f(f(a),b)$, $(x, X(a,x), f(f(a),b))$, and $(x, X(a,x), f(f(\circ),b))$.

A context $\tilde{c}$ can apply to a hedge $\tilde{s}$, denoted by $\tilde{c}[\tilde{s}]$, obtaining a hedge by replacing the hole in $\tilde{c}$ with $\tilde{s}$. For example, $(x, X(a,x), f(f(\circ),b))[a, X(a)] = (x, X(a,x), f(f(a, X(a)),b))$. Application of a context to a context is defined similarly.

The length of a hedge $\tilde{s}$, denoted $|\tilde{s}|$, is the number of elements in it. We denote by $\tilde{s}|_i$ the ith element of $\tilde{s}$ and by $\tilde{s}|_i^j$ the subhedge $(\tilde{s}|_i, \ldots, \tilde{s}|_j)$. If $i > j$ then $\tilde{s}|_i^j$ is the empty hedge. The set of all function symbols which appear in a hedge $\tilde{s}$ (resp., in a context $\tilde{c}$) is denoted by $\mathcal{F}(\tilde{s})$ (resp., by $\mathcal{F}(\tilde{c})$). We overload the notation $\mathcal{F}(A)$ for the set of all function symbols which appear in a set of hedges and contexts A.

A *substitution* is a mapping from hedge variables to hedges and from context variables to contexts, which is identity almost everywhere. When substituting a context variable X by a context, the context will be applied to the argument hedge of X. The symbols σ, ϑ are used to denote a substitution. Substitutions can be applied to hedges and contexts in the usual way. We use postfix notation for application, writing, e.g., $\tilde{s}\sigma$ for the application of σ to $\tilde{s}$. For example, if $\sigma = \{x \mapsto \epsilon, y \mapsto (a, x), X \mapsto g(\circ)\}$ is a substitution, then $(X(x), y, f(X(y), c))\sigma = (g, a, x, f(g(a, x), c))$. The notion *range* of a substitution σ is standard and denoted by $\mathrm{Ran}(\sigma)$.

A hedge $\tilde{s}$ is the *instance* of a hedge $\tilde{q}$ if there exists a substitution σ with $\tilde{q}\sigma = \tilde{s}$. We say that $\tilde{q}$ is more general than $\tilde{s}$ if $\tilde{s}$ is an instance of $\tilde{q}$ and denote this by $\tilde{q} \preceq \tilde{s}$. If $\tilde{q} \preceq \tilde{s}$ and $\tilde{s} \preceq \tilde{q}$, then we write $\tilde{q} \simeq \tilde{s}$. If $\tilde{q} \preceq \tilde{s}$ and $\tilde{q} \not\simeq \tilde{s}$, then we say that $\tilde{q}$ is strictly more general than $\tilde{s}$ and write $\tilde{q} \prec \tilde{s}$. A hedge $\tilde{g}$ is a *generalization* of the hedges $\tilde{s}$ and $\tilde{q}$ if $\tilde{s}$ and $\tilde{q}$ are instances of $\tilde{g}$.

The word representation $\omega(\tilde{s})$ of a hedge $\tilde{s}$ is defined by the concatenation of the depth-first pre-order traversal of the constituent terms. For instance, $afgagbbc$ is the word representation of $(a, f(g(a, g(b, b)), c))$. Generalizations contain a common subsequence of the word representation of the input hedges. We will use this property in the formulation of our anti-unification algorithm. Observe, e.g., the hedges from example 1:

$$\begin{array}{l} (\ h(\mathbf{a}), \mathbf{f}(\ h(\mathbf{g}(\mathbf{a}, b, b), \mathbf{c}), b, b)) \\ (\quad \mathbf{a}\ , \mathbf{f}(\quad \mathbf{g}(\mathbf{a}, d\ \ ), \mathbf{c}\ , d\ \ )) \\ \hline (X(\mathbf{a}), \mathbf{f}(X(\mathbf{g}(\mathbf{a}, x\ \ ), \mathbf{c}), x\ \ )) \end{array}$$

The set of *positions* of a hedge $\tilde{s} = (t_1, \ldots t_n)$, denoted $pos(\tilde{s})$, is a set of strings of positive integers. It is defined as $pos(\tilde{s}) := \bigcup_{i=1}^{n} \{i{\cdot}p \mid p \in pos_{\mathsf{T}}(t_i)\}$, where $\cdot$ stands for concatenation. $pos_{\mathsf{T}}(t)$ is defined as $pos_{\mathsf{T}}(x) := \{\lambda\}$ and $pos_{\mathsf{T}}(\phi(\tilde{q})) := \{\lambda\} \cup pos(\tilde{q})$, where λ is the empty string. For example, $pos(f(a, g(b, c))) = \{1, 1{\cdot}1, 1{\cdot}2, 1{\cdot}2{\cdot}1, 1{\cdot}2{\cdot}2\}$ and $pos(a, f(b, g(c)), d) = \{1, 2, 2{\cdot}1, 2{\cdot}2, 2{\cdot}2{\cdot}1, 3\}$. In the latter hedge, the symbol g stands at the position $2{\cdot}2$ and c occurs at the position $2{\cdot}2{\cdot}1$.

- Two symbols $\mathsf{s}_1, \mathsf{s}_2 \in \mathcal{F} \cup \mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$ of a hedge are *horizontal consecutive* if the corresponding positions $I_{\mathsf{s}_1}{\cdot}i_{\mathsf{s}_1}$ and $I_{\mathsf{s}_2}{\cdot}i_{\mathsf{s}_2}$ are in the relation $I_{\mathsf{s}_1} = I_{\mathsf{s}_1}$ and $i_{\mathsf{s}_1} + 1 = i_{\mathsf{s}_2}$.
- Two symbols $\mathsf{s}_1, \mathsf{s}_2 \in \mathcal{F} \cup \mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$ of a hedge $\tilde{s}$ are in a *vertical chain* if their positions I_{s_1} and I_{s_2} are in the relation $I_{\mathsf{s}_1}{\cdot}1 = I_{\mathsf{s}_2}$ and $I_{\mathsf{s}_1}{\cdot}2 \notin pos(\tilde{s})$.

For example, in $(a, f(X(a, b)))$, the occurrence of a at position 1 and the occurrence of f at 2 are horizontal consecutive, as well as a at $2{\cdot}1{\cdot}1$ and b at $2{\cdot}1{\cdot}2$. The occurrence of f at 2 and the occurrence of X at $2{\cdot}1$ are in vertical chain.

With $<$ we denote the (strict) *lexicographic ordering* and with $\sqsubset$ the (strict) *ancestor relation* on positions, e.g., $1{\cdot}2{\cdot}1 < 1{\cdot}2{\cdot}2$, $1{\cdot}2{\cdot}1 < 1{\cdot}2{\cdot}1{\cdot}2$, and $1{\cdot}2{\cdot}1 \sqsubset 1{\cdot}2{\cdot}1{\cdot}2$. The relation $\sqsubseteq$ is defined as $\sqsubset \cup =$.

Given three positions I_1, I_2 and I_3, the ternary relation $\bowtie$ is defined as

$$I_1 \bowtie_{I_3} I_2 :\Longleftrightarrow \text{there is } I_4 \neq \lambda \text{ such that } I_4 \sqsubset I_1 \text{ and } I_4 \sqsubset I_2 \text{ and } I_4 \not\sqsubset I_3 \text{ and } I_1, I_2, I_3 \text{ are pairwise not in } \sqsubseteq .$$

This relation tests whether I_1 and I_2 have a common ancestor which is not an ancestor of I_3. None of these positions should be an ancestor of another. For instance, $1{\cdot}1 \bowtie_2 1{\cdot}2$, but *not* $1 \bowtie_3 2$, $1{\cdot}1 \bowtie_2 1{\cdot}1{\cdot}2$, $1{\cdot}1 \bowtie_2 1{\cdot}1$, and $1{\cdot}1 \bowtie_{1{\cdot}3} 1{\cdot}2$. A real world example of this relation would be two sisters and one of their uncles.

3 The Skeletons: Admissible Alignments

In this section we introduce the concept of admissible alignments, which are used later as skeletons to compute corresponding generalizations. For simplicity, we formulate all the notions and the algorithm for two hedges. The extension to more hedges is straightforward. Hedges to be generalized are assumed to be variable disjoint.

Given two hedges $\tilde{s}$ and $\tilde{q}$, an *alignment* is a sequence of the form $a_1\langle I_1, J_1\rangle \dots a_m\langle I_m, J_m\rangle$ such that $I_1 < \dots < I_m$, $J_1 < \dots < J_m$, and a_k is the symbol at position I_k in $\tilde{s}$ and at position J_k in $\tilde{q}$ for all $1 \leqslant k \leqslant m$.

An alignment represents common function symbols inside of two hedges with the corresponding positions, respecting the ordering $<$. It is a common subsequence of the word representation of those hedges with some additional information about the positions. The empty alignment is denoted by $\mathfrak{e}$.

Collisions in an alignment $\mathfrak{a}$ of two hedges are defined as follows:

- A collision appears at two elements $a_k\langle I_k, J_k\rangle$, $a_l\langle I_l, J_l\rangle$ of $\mathfrak{a}$ if either ($I_k \sqsubset I_l$ and $J_k \sqsupset J_l$) or ($I_k \sqsupset I_l$ and $J_k \sqsubset J_l$).
- A collision appears at three elements $a_k\langle I_k, J_k\rangle$, $a_l\langle I_l, J_l\rangle$, $a_n\langle I_n, J_n\rangle$ of $\mathfrak{a}$ if $I_k \bowtie_{I_n} I_l$ and $J_l \bowtie_{J_k} J_n$.

For instance, the alignment $f\langle 2, 1\rangle b\langle 2{\cdot}1, 1{\cdot}2\rangle c\langle 2{\cdot}2, 2\rangle$ of the hedges $(a, f(b, c))$ and $(f(a, b), c)$ contains a collision at the two elements $f\langle 2, 1\rangle$ and $c\langle 2{\cdot}2, 2\rangle$. The alignment $a\langle 1, 1{\cdot}1\rangle b\langle 2{\cdot}1, 1{\cdot}2\rangle c\langle 2{\cdot}2, 2\rangle$ of the same hedges has a collision at its three elements.

An alignment $\mathfrak{a}$ is called *admissible* if there are no collisions in it. Note that for any two elements $a_k\langle I_k, J_k\rangle$, $a_l\langle I_l, J_l\rangle$ of an admissible alignment $\mathfrak{a}$, $I_k < I_l$ iff $J_k < J_l$ and $I_k \sqsubset I_l$ iff $J_k \sqsubset J_l$.

Admissible alignments are related to generalization by the following theorem:

Theorem 1. *Let $\mathfrak{a} = a_1\langle I_1, J_1\rangle \dots a_m\langle I_m, J_m\rangle$ be an alignment of $\tilde{s}$ and $\tilde{q}$ such that for all $1 \leqslant k \leqslant m$ the function symbol a_k is unique in $\tilde{s}$ and unique in $\tilde{q}$. $\mathfrak{a}$ is admissible iff there exists a generalization $\tilde{g}$ of $\tilde{s}$ and $\tilde{q}$ with $\mathcal{F}(\tilde{g}) = \{a_1, \dots, a_m\}$.*

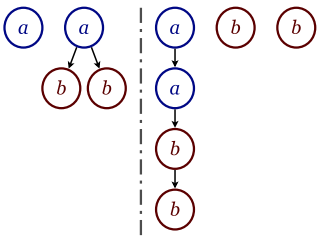
Notice that requiring uniqueness of the function symbols $a_1, \dots, a_m$ in theorem 1 does not impose a loss of generality. One can simply rename those symbols with fresh ones and restore the original function symbols afterwards.

From this theorem, we get that for any admissible alignment $\mathfrak{a}$ of two hedges there exists a generalization $\tilde{g}$ of those hedges which contains all the corresponding function symbols. The other direction is also true: For any generalization $\tilde{g}$ of two hedges there exists their admissible alignment containing all the function symbols which appear in $\tilde{g}$.

We call such a $\tilde{g}$ a *supporting generalization* of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$.

Example 2. Let $\tilde{s} = (a, a(b, b))$ and $\tilde{q} = (a(a(b(b))), b, b)$.

- $b\langle 2{\cdot}2, 1{\cdot}1{\cdot}1\rangle\, a\langle 1, 1\rangle$ is not an alignment of $\tilde{s}, \tilde{q}$.
- $a\langle 1, 1\rangle\, a\langle 2, 1{\cdot}1\rangle\, b\langle 2{\cdot}1, 1{\cdot}1{\cdot}1\rangle\, b\langle 2{\cdot}2, 3\rangle$ is a non-admissible alignment of $\tilde{s}, \tilde{q}$.
- $a\langle 1, 1{\cdot}1\rangle\, b\langle 2{\cdot}1, 2\rangle\, b\langle 2{\cdot}2, 3\rangle$ is an admissible alignment of $\tilde{s}, \tilde{q}$, with $(X(a(x)), Y(b, b))$ being a corresponding supporting generalization.
- $(x, a(y, Y(b)), z)$ is a supporting generalization of $\tilde{s}, \tilde{q}$, with respect to $a\langle 2, 1\rangle\, b\langle 2{\cdot}2, 1{\cdot}1{\cdot}1{\cdot}1\rangle$.



4 Computing Least General Rigid Generalizations

We aim at solving the following problem: Given two hedges $\tilde{s}$ and $\tilde{q}$ and their admissible alignment $\mathfrak{a}$, compute a least general supporting generalization $\tilde{g}$ of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$. However, least general supporting generalizations might not be unique. For instance, for (a, b, a) and (b, c) with the admissible alignment $b\langle 2, 1\rangle$, we have two supporting lggs (x, b, x, y) and (x, b, y, x).

Therefore, we are interested in a special class of supporting generalizations, which we call rigid generalizations. Given two hedges $\tilde{s}, \tilde{q}$ and their admissible alignment $\mathfrak{a}$, a hedge $\tilde{g}$ is called a *rigid generalization* of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$, if $\tilde{g}$ is a supporting generalization of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$ such that the following conditions hold:

- There exist substitutions σ, ϑ with $\tilde{g}\sigma = \tilde{s}$ and $\tilde{g}\vartheta = \tilde{q}$ such that all the contexts in σ and ϑ are singleton contexts.
- No context variable in $\tilde{g}$ applies to the empty hedge.
- $\tilde{g}$ doesn't contain horizontal consecutive hedge variables.
- $\tilde{g}$ doesn't contain vertical chains of variables.
- $\tilde{g}$ doesn't contain context variables with a hedge variable as the first or the last argument (i.e., no subterms of the form $X(x, \dots)$ and $X(\dots, x)$).

This definition puts some restrictions on the usage of the variables. Especially, our very general concept of context variables demands for some restrictions. For instance, $X(a, b)$ is a rigid generalization of $f(g(a, b, c))$ and (a, b) with respect to $a\langle 1{\cdot}1{\cdot}1, 1\rangle$ $b\langle 1{\cdot}1{\cdot}2, 2\rangle$, while $X(a, b, x)$ and $X(Y(a, b))$ are not rigid generalizations.

A rigid generalization $\tilde{g}$ of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$ is called a *rigid lgg* of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$, if there is no rigid generalization $\tilde{h}$ of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$ which satisfies $\tilde{g} \prec \tilde{h}$.

Note that two hedges might have a supporting generalization which is less general than their rigid lgg with respect to the same admissible alignment. For instance, $X(a) \prec X(X(a))$ and both of them are generalizations of $f(f(a))$ and $g(g(g(g(a))))$ with respect to $a\langle 1{\cdot}1{\cdot}1, 1{\cdot}1{\cdot}1{\cdot}1{\cdot}1\rangle$, but only $X(a)$ is a rigid generalization.

From now on, we concentrate on computing least general rigid generalizations of two variable-disjoint hedges with respect to an admissible alignment.

An *anti-unification problem* (AUP) is a triple of the form $x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ \mathfrak{a}$, where

- x is a hedge variable and $\tilde{s}, \tilde{q}$ are hedges,
- X is a context variable and $\tilde{c}, \tilde{d}$ are contexts,
- $\mathfrak{a}$ is an admissible alignment of $\tilde{s}$ and $\tilde{q}$.

We present our anti-unification algorithm as a rule-based algorithm that works on triples $P;\ S;\ \sigma$, where the problem set P is a set of AUPs, the store S is a set of AUPs with empty alignments, σ is a substitution which keeps track of the generalization computed so far, and for all pairs of AUPs $\{x\colon \tilde{s}_1 \triangleq \tilde{q}_1;\ X\colon \tilde{c}_1 \triangleq \tilde{d}_1;\ \mathfrak{a}_1,\ \ y\colon \tilde{s}_2 \triangleq \tilde{q}_2;\ Y\colon \tilde{c}_2 \triangleq \tilde{d}_2;\ \mathfrak{a}_2\} \subseteq P \cup S$ holds $x \neq y$ and $X \neq Y$.

As all the AUPs in S have the empty alignment, we write $x\colon\ \tilde{s} \triangleq \tilde{q};\ X\colon\ \tilde{c} \triangleq \tilde{d}$ instead of $x\colon\ \tilde{s} \triangleq \tilde{q};\ X\colon\ \tilde{c} \triangleq \tilde{d};\ \mathfrak{e}$ for an AUP of S. In the rules below, we use the symbols Y, Z for fresh context variables and y, z for fresh hedge variables. The brackets $[\,]$, as before, are used for context application. The symbol $\cup$ stands for disjoint union. Furthermore, i^{--} denotes $i - 1$ and i^{++} denotes $i + 1$.

Spl-H: **Split Hedge**

$\{x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ a_1\langle i_1{\cdot}I_1, j_1{\cdot}J_1\rangle \dots a_k\langle i_k{\cdot}I_k, j_k{\cdot}J_k\rangle$
$a_{k+1}\langle i_{k+1}{\cdot}I_{k+1}, j_{k+1}{\cdot}J_{k+1}\rangle \dots a_m\langle i_m{\cdot}I_m, j_m{\cdot}J_m\rangle\} \cup P;\ S;\ \sigma \implies$
$\{y\colon \tilde{s}|_{i_1}^{i_k} \triangleq \tilde{q}|_{j_1}^{j_k};\ Y\colon \circ \triangleq \circ;\ a_1\langle (i_1 - i_1^{--}){\cdot}I_1, (j_1 - j_1^{--}){\cdot}J_1\rangle \dots$
$a_k\langle (i_k - i_1^{--}){\cdot}I_k, (j_k - j_1^{--}){\cdot}J_k\rangle\} \cup$
$\{z\colon \tilde{s}|_{i_k^{++}}^{i_m} \triangleq \tilde{q}|_{j_k^{++}}^{j_m};\ Z\colon \circ \triangleq \circ;\ a_{k+1}\langle (i_{k+1} - i_k){\cdot}I_{k+1}, (j_{k+1} - j_k){\cdot}J_{k+1}\rangle \dots$
$a_m\langle (i_m - i_k){\cdot}I_m, (j_m - j_k){\cdot}J_m\rangle\} \cup P;$
$\{x\colon \epsilon \triangleq \epsilon;\ X\colon \tilde{c}[\tilde{s}|_1^{i_1^{--}}, \circ, \tilde{s}|_{i_m^{++}}^{|\tilde{s}|}] \triangleq \tilde{d}[\tilde{q}|_1^{j_1^{--}}, \circ, \tilde{q}|_{j_m^{++}}^{|\tilde{q}|}]\} \cup S;\ \sigma\{x \mapsto (Y(y), Z(z))\},$

If $i_1 \neq i_{k+1}$ and $j_1 \neq j_{k+1}$, and, moreover, $i_1 = i_k$ or $j_1 = j_k$, for $1 \leqslant k < m$.

Abs-L: **Abstract Left Context**

$\{x\colon (\tilde{s}_l, \phi(\tilde{s}), \tilde{s}_r) \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ a_1\langle i{\cdot}I_1, J_1\rangle \dots a_m\langle i{\cdot}I_m, J_m\rangle\} \cup P;\ S;\ \sigma \implies$
$\{x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c}[\tilde{s}_l, \phi(\circ), \tilde{s}_r] \triangleq \tilde{d};\ a_1\langle I_1, J_1\rangle \dots a_m\langle I_m, J_m\rangle\} \cup P;\ S;\ \sigma,$

where $I_1 \neq \lambda$, $\phi(\tilde{s})$ is the term at position i in $(\tilde{s}_l, \phi(\tilde{s}), \tilde{s}_r)$, and $\tilde{s}_l, \tilde{s}_r$ are hedges.

Abs-R: **Abstract Right Context**

$\{x\colon \tilde{s} \triangleq (\tilde{q}_l, \phi(\tilde{q}), \tilde{q}_r);\ X\colon \tilde{c} \triangleq \tilde{d};\ a_1\langle I_1, j{\cdot}J_1\rangle \dots a_m\langle I_m, j{\cdot}J_m\rangle\} \cup P;\ S;\ \sigma \implies$
$\{x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d}[\tilde{q}_l, \phi(\circ), \tilde{q}_r];\ a_1\langle I_1, J_1\rangle \dots a_m\langle I_m, J_m\rangle\} \cup P;\ S;\ \sigma,$

where $J_1 \neq \lambda$, $\phi(\tilde{q})$ is the term at position j in $(\tilde{q}_l, \phi(\tilde{q}), \tilde{q}_r)$, and $\tilde{q}_l, \tilde{q}_r$ are hedges.

App-A: **Apply Alignment**

$\{x\colon (\tilde{s}_l, a_1(\tilde{s}), \tilde{s}_r) \triangleq (\tilde{q}_l, a_1(\tilde{q}), \tilde{q}_r);\ X\colon \tilde{c} \triangleq \tilde{d};$
$a_1\langle i, j\rangle a_2\langle i{\cdot}I_2, j{\cdot}J_2\rangle \dots a_m\langle i{\cdot}I_m, j{\cdot}J_m\rangle\} \cup P;\ S;\ \sigma \implies$
$\{y\colon \tilde{s} \triangleq \tilde{q};\ Y\colon \circ \triangleq \circ;\ a_2\langle I_2, J_2\rangle \dots a_m\langle I_m, J_m\rangle\} \cup P;$
$\{x\colon \epsilon \triangleq \epsilon;\ X\colon \tilde{c}[\tilde{s}_l, \circ, \tilde{s}_r] \triangleq \tilde{d}[\tilde{q}_l, \circ, \tilde{q}_r]\} \cup S;\ \sigma\{x \mapsto a_1(Y(y))\},$

where $a_1(\tilde{s}), a_1(\tilde{q})$ are the terms at the positions i, j and $\tilde{s}_l, \tilde{s}_r, \tilde{q}_l, \tilde{q}_r$ are hedges.

Sol-H: **Solve Hedge**

$\{x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \circ \triangleq \circ;\ \mathfrak{e}\} \cup P;\ S;\ \sigma \Longrightarrow P;\ \{x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \circ \triangleq \circ\} \cup S;\ \sigma\{X \mapsto \circ\}.$

Res-C: **Restore Context**

$P;\ \{x\colon \epsilon \triangleq \epsilon;\ X\colon (\tilde{s}_l, \dot{c}, \tilde{s}_r) \triangleq (\tilde{q}_l, \dot{d}, \tilde{q}_r)\} \cup S;\ \sigma \Longrightarrow$
$P;\ \{x\colon \epsilon \triangleq \epsilon;\ X\colon \dot{c} \triangleq \dot{d},\ y\colon \tilde{s}_l \triangleq \tilde{q}_l;\ Y\colon \circ \triangleq \circ,\ z\colon \tilde{s}_r \triangleq \tilde{q}_r;\ Z\colon \circ \triangleq \circ\} \cup S;$
$\sigma\{X \mapsto (y, X(\circ), z)\},$

if not $\epsilon = \tilde{s}_l = \tilde{s}_r = \tilde{q}_l = \tilde{q}_r$. $\dot{c}, \dot{d}$ are singleton contexts.

Mer-S: **Merge Store**

$P;\ \{x_1\colon \tilde{s} \triangleq \tilde{q};\ X_1\colon \tilde{c} \triangleq \tilde{d},\ x_2\colon \tilde{s} \triangleq \tilde{q};\ X_2\colon \tilde{c} \triangleq \tilde{d}\} \cup S;\ \sigma \Longrightarrow$
$P;\ \{x_1\colon \tilde{s} \triangleq \tilde{q};\ X_1\colon \tilde{c} \triangleq \tilde{d}\} \cup S;\ \sigma\{x_2 \mapsto x_1, X_2 \mapsto X_1\}.$

Clr-S: **Clear Store**

$P;\ \{x\colon \epsilon \triangleq \epsilon;\ X\colon \circ \triangleq \circ\} \cup S;\ \sigma \implies P;\ S;\ \sigma\{x \mapsto \epsilon, X \mapsto \circ\}.$

The idea of the store is to keep track of already solved AUPs in order to generalize the same AUPs in the same way, as it is illustrated in the Mer-S rule.

To compute generalizations of $\tilde{s}$ and $\tilde{q}$ with respect to an admissible alignment $\mathfrak{a}$, the procedure starts with $\{x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \circ \triangleq \circ;\ \mathfrak{a}\};\ \varnothing;\ \varepsilon$, where x and X are fresh variables, and applies the rules exhaustively. We denote this procedure by $\mathfrak{G}$. The intuition is that at i's step of such a derivation, $X(x)\sigma_i$ is supposed to be a generalization of $\tilde{s}$ and $\tilde{q}$, with the idea that when the process stops with σ in the last step, then $X(x)\sigma$ is a rigid lgg of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$.

Before discussing the properties of $\mathfrak{G}$, we briefly explain informally what the rules do. At each step, each AUP $x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ \mathfrak{a}$ in P represents the hedges $\tilde{c}[\tilde{s}]$ and $\tilde{d}[\tilde{q}]$ which are to be generalized, such that the generalization contains the function symbols from $\mathfrak{a}$. They are split according to the occurrences of alignment elements: All symbols from $\mathfrak{a}$ are in $\tilde{s}$ and $\tilde{q}$. None of them appear in $\tilde{c}$ and $\tilde{d}$.

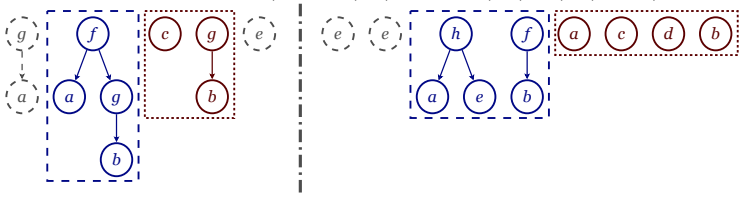
Such an AUP can be transformed by one of the first four rules: Spl-H, Abs-L, Abs-R, or App-A. The eventual goal of these transformations is to reach the occurrences of the first alignment element in $\tilde{s}$ and $\tilde{q}$. In the course of the transformation, $\tilde{c}$ and $\tilde{d}$ are getting extended with contexts above those occurrences.

When the symbols in $\mathfrak{a}$ are distributed in more than one term both in $\tilde{s}$ and in $\tilde{q}$, then we use the Spl-H rule to select subhedges of $\tilde{s}$ and $\tilde{q}$ which contain all the alignment elements. (The other parts of $\tilde{s}$ and $\tilde{q}$ are moved to the store, since they will not contribute a symbol to the generalization.) Furthermore, by this rule, each of these subhedges are split into two smaller subhedges: From the $\tilde{s}$ side these are $\tilde{s}|_{i_1}^{i_k}$ and $\tilde{s}|_{i_k^{++}}^{i_m}$, and from the $\tilde{q}$ side they are $\tilde{q}|_{j_1}^{j_k}$ and $\tilde{q}|_{j_k^{++}}^{j_m}$. The split point k is decided by the following criteria:

- $\tilde{s}|_{i_1}^{i_k}$ and $\tilde{q}|_{j_1}^{j_k}$ contain the first $k > 0$ elements of $\mathfrak{a}$.
- $\tilde{s}|_{i_k^{++}}^{i_m}$ and $\tilde{q}|_{j_k^{++}}^{j_m}$ contain the elements of $\mathfrak{a}$ starting from $k+1$. There exists at least one such element.
- $\tilde{s}|_{i_1}^{i_k}$ or $\tilde{q}|_{j_1}^{j_k}$ is a term (a singleton hedge), and the $k+1$'st element of $\mathfrak{a}$ does not belong to it.

The process will continue by generalizing $\tilde{s}|_{i_1}^{i_k}$ and $\tilde{q}|_{j_1}^{j_k}$ with respect to the first k-element prefix of $\mathfrak{a}$, and generalizing $\tilde{s}|_{i_k^{++}}^{i_m}$ and $\tilde{q}|_{j_k^{++}}^{j_m}$ with respect to the elements of $\mathfrak{a}$ starting from $k+1$. Note that in the next step Spl-H is not applicable to the AUP with $\tilde{s}|_{i_1}^{i_k}$ and $\tilde{q}|_{j_1}^{j_k}$. This is because at least one of them is a single term which completely contains the alignment elements. Therefore either Abs-L, Abs-R, or App-A applies.

Consider the hedges $(g(a), f(a, g(b)), c, g(b), e)$ and $(e, e, h(a, e), f(b), a, c, d, b)$ and the admissible alignment $a\langle 2{\cdot}1, 3{\cdot}1\rangle b\langle 2{\cdot}2{\cdot}1, 4{\cdot}1\rangle c\langle 3, 6\rangle b\langle 4{\cdot}1, 8\rangle$ of them.



The dashed nodes in the figure above denote the parts which are moved into the store. The dashed rectangle denotes $\tilde{s}|_{i_1}^{i_k}$ and $\tilde{q}|_{j_1}^{j_k}$ and the dotted one denotes $\tilde{s}|_{i_k^{++}}^{i_m}$ and $\tilde{q}|_{j_k^{++}}^{j_m}$.

When all symbols in $\mathfrak{a}$ belong to one term in $\tilde{s}$ or in $\tilde{q}$ (or maybe both), but the root of that term is *not* the symbol a_1 from the first element of $\mathfrak{a}$, then an attempt is made to

get deeper to that term, to reach the subterm whose top symbol is the a_1 from $\mathfrak{a}$. This descent is carried out by Abs-L or Abs-R, depending whether we are searching for the subterm with a_1 in the top in $\tilde{s}$ or in $\tilde{q}$.

When all symbols in $\mathfrak{a}$ belong to one term in $\tilde{s}$ and one term in $\tilde{q}$, and these terms have the same root symbol which is exactly the a_1 from the first element of $\mathfrak{a}$, then a_1 is moved to the generalization. This is what the App-A rule does. The process will continue with generalizing the hedges under the occurrences of a_1 in $\tilde{s}$ and $\tilde{q}$.

When the alignment is empty in $x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ \mathfrak{e}$ in P, then the hedge there will not contribute a symbol in the generalization. Moreover, both $\tilde{c}$ and $\tilde{d}$ are holes, because only App-A can make the alignment empty, and it makes the contexts in the obtained AUP the hole. Such AUPs are considered solved, as their generalization is just x they contain. They should be put in the store, which keeps information about the differences between the hedges to be generalized. At the same time, the context variable X can be deleted, as it just stand for the hole. This is what the Sol-H rule does.

The other three rules work on the store. Clr-S removes the empty AUP from the store and eliminates the corresponding variables form the generalization. Mer-S guarantees that the same AUPs are generalized with the same variables, making sure that the same differences in the input hedges are generalized uniformly. Finally, the Res-C rule guarantees that each context variable in the generalization generalizes singleton contexts in the input hedges: A property required for rigid generalizations.

We define two substitutions obtained by a set S of AUPs:

$$\sigma_{\text{L}}(S) ::= \{x \mapsto \tilde{s}, X \mapsto \tilde{c} \mid x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ \mathfrak{a} \in S\}$$
$$\sigma_{\text{R}}(S) ::= \{x \mapsto \tilde{q}, X \mapsto \tilde{d} \mid x\colon \tilde{s} \triangleq \tilde{q};\ X\colon \tilde{c} \triangleq \tilde{d};\ \mathfrak{a} \in S\}$$

Example 3. Let $\tilde{s} = f(a, f(b, b))$ and $\tilde{q} = (b, f(a, b), b)$ be the input hedges with the admissible alignment $\mathfrak{a} = f\langle 1, 2\rangle a\langle 1{\cdot}1, 2{\cdot}1\rangle b\langle 1{\cdot}2{\cdot}1, 2{\cdot}2\rangle$. Then the algorithm $\mathfrak{G}$ starts with the initial system

$$\{x\colon f(a, f(b, b)) \triangleq (b, f(a, b), b);\ X\colon \circ \triangleq \circ;\ f\langle 1, 2\rangle a\langle 1{\cdot}1, 2{\cdot}1\rangle b\langle 1{\cdot}2{\cdot}1, 2{\cdot}2\rangle\};\ \varnothing;\ \varepsilon$$

and computes $\varnothing; S; \sigma$, where

$$S = \{y\colon \epsilon \triangleq \epsilon;\ Y\colon f(\circ, b) \triangleq \circ,\ z\colon \epsilon \triangleq b;\ Z\colon \circ \triangleq \circ\},$$
$$\sigma = \{x \mapsto (z, f(a, Y(b)), z), X \mapsto \circ, \ldots\}.$$

$X(x)\sigma = (z, f(a, Y(b)), z)$ generalizes $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$. From the store S we can read $\sigma_{\text{L}}(S) = \{z \mapsto \epsilon, Y \mapsto f(\circ, b), \ldots\}$ and $\sigma_{\text{R}}(S) = \{z \mapsto b, Y \mapsto \circ, \ldots\}$. Then we have $X(x)\sigma\sigma_{\text{L}}(S) = \tilde{s}$ and $X(x)\sigma\sigma_{\text{R}}(S) = \tilde{q}$.

Now we turn to discussing the properties of $\mathfrak{G}$. Termination is the first of them:

Theorem 2 (Termination of $\mathfrak{G}$). *The system $\mathfrak{G}$ terminates on any input.*

The substitutions $\sigma_{\text{L}}(S)$ and $\sigma_{\text{R}}(S)$ are used to characterize the invariant of $\mathfrak{G}$:

Lemma 1 (Generalization Invariant). *Let $P_0;\ S_0;\ \sigma_0$ such that for all $x_0\colon \tilde{s}_0 \triangleq \tilde{q}_0;\ X_0\colon \tilde{c}_0 \triangleq \tilde{d}_0;\ \mathfrak{a}_0 \in P_0$ the variables x_0, X_0 only appear together as term $X_0(x_0)$ in σ_0. If $P_0;\ S_0;\ \sigma_0 \Longrightarrow^* P_n;\ S_n;\ \sigma_n$ is a derivation in $\mathfrak{G}$ then for all $x_0\colon \tilde{s}_0 \triangleq \tilde{q}_0;\ X_0\colon \tilde{c}_0 \triangleq \tilde{d}_0;\ \mathfrak{a}_0 \in P_0 \cup S_0$ holds*

– $X_0(x_0)\sigma_0\sigma_{\scriptscriptstyle L}(P_0 \cup S_0) = X_0(x_0)\sigma_n\sigma_{\scriptscriptstyle L}(P_n \cup S_n)$,
– $X_0(x_0)\sigma_0\sigma_{\scriptscriptstyle R}(P_0 \cup S_0) = X_0(x_0)\sigma_n\sigma_{\scriptscriptstyle R}(P_n \cup S_n)$.

This lemma has a corollary which states that for the invariant, the initial substitution is irrelevant:

Corollary 1. *If* $P_0;\ S_0;\ \vartheta_0 \Longrightarrow^*\ P_n;\ S_n;\ \vartheta_0\vartheta_1 \ldots \vartheta_n$ *is a derivation in* $\mathfrak{G}$ *then for all* $x_0 \colon \tilde{s}_0 \triangleq \tilde{q}_0;\ X_0 \colon \tilde{c}_0 \triangleq \tilde{d}_0;\ \mathfrak{a}_0 \in P_0 \cup S_0$ *holds*

– $X_0(x_0)\sigma_{\scriptscriptstyle L}(P_0 \cup S_0) = X_0(x_0)\vartheta_1 \ldots \vartheta_n\sigma_{\scriptscriptstyle L}(P_n \cup S_n)$,
– $X_0(x_0)\sigma_{\scriptscriptstyle R}(P_0 \cup S_0) = X_0(x_0)\vartheta_1 \ldots \vartheta_n\sigma_{\scriptscriptstyle R}(P_n \cup S_n)$.

The soundness theorem shows that $\mathfrak{G}$ indeed computes rigid generalizations. Besides, the store keeps the information which indicates how to obtain the initial hedges from the generalization:

Theorem 3 (Soundness of $\mathfrak{G}$). *Let* P *be a set of AUPs of the form* $\{x \colon \tilde{s} \triangleq \tilde{q};\ X : \circ \triangleq \circ;\ \mathfrak{a}\}$. *Every exhaustive rule application in* $\mathfrak{G}$ *yields a derivation* $P;\ \varnothing;\ \varepsilon \Longrightarrow^+ \varnothing;\ S;\ \sigma$ *where* $\tilde{g} = X(x)\sigma$ *is a rigid generalization of* $\tilde{s}$ *and* $\tilde{q}$ *with respect to* $\mathfrak{a}$ *and the store* S *records all the differences such that* $\tilde{g}\sigma_{\scriptscriptstyle L}(S) = \tilde{s}$ *and* $\tilde{g}\sigma_{\scriptscriptstyle R}(S) = \tilde{q}$.

The next theorem is the Completeness Theorem. It, essentially, says that for the given alignment, a rigid generalization $\mathfrak{G}$ computes is least general among all rigid generalizations of the input hedges.

Theorem 4 (Completeness of $\mathfrak{G}$). *Let* $\tilde{g}$ *be a rigid generalization of* $\tilde{s}$ *and* $\tilde{q}$ *with respect to* $\mathfrak{a}$. *Then there exists a derivation* $\{x \colon \tilde{s} \triangleq \tilde{q};\ X : \circ \triangleq \circ;\ \mathfrak{a}\};\ \varnothing;\ \varepsilon \Longrightarrow^+ \varnothing;\ S;\ \sigma$ *obtained by* $\mathfrak{G}$ *such that* $\tilde{g} \preceq X(x)\sigma$.

There is a nondeterminism in the algorithm. The Uniqueness Theorem says that different transformations compute generalizations which are equivalent modulo $\simeq$, i.e., differ from each other only by variable renaming:

Theorem 5 (Uniqueness modulo $\simeq$). *Let* $\mathfrak{a}$ *be an admissible alignment of* $\tilde{s}$ *and* $\tilde{q}$. *If* $\{x_1 \colon \tilde{s} \triangleq \tilde{q};\ X_1 : \circ \triangleq \circ;\ \mathfrak{a}\};\ \varnothing;\ \varepsilon \Longrightarrow^+ \varnothing;\ S_1;\ \sigma_1$ *and* $\{x_2 \colon \tilde{s} \triangleq \tilde{q};\ X_2 : \circ \triangleq \circ;\ \mathfrak{a}\};\ \varnothing;\ \varepsilon \Longrightarrow^+ \varnothing;\ S_2;\ \sigma_2$ *are two exhaustive derivations in* $\mathfrak{G}$, *then* $X_1(x_1)\sigma_1 \simeq X_2(x_2)\sigma_2$.

Finally, the complexity analysis reveals that the algorithm runs in quadratic time and requires linear space:

Theorem 6 (Complexity of $\mathfrak{G}$). *The anti-unification algorithm* $\mathfrak{G}$ *has* $O(n^2)$ *time complexity and* $O(n)$ *space complexity, where* n *is the number of symbols in the input.*

Acknowledgments

This research has been supported by the Austrian Science Fund (FWF) under the project SToUT (P 24087-N18).

Bibliography

[1] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. A modular equational generalization algorithm. In M. Hanus, editor, *LOPSTR*, volume 5438 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2008. ISBN 978-3-642-00514-5.

[2] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. Order-sorted generalization. *Electr. Notes Theor. Comput. Sci.*, 246:27–38, 2009.

[3] A. Amir, T. Hartman, O. Kapah, B. R. Shalom, and D. Tsur. Generalized LCS. *Theor. Comput. Sci.*, 409(3):438–449, 2008.

[4] E. Armengol and E. Plaza. Bottom-up induction of feature terms. *Machine Learning*, 41(3):259–294, 2000.

[5] F. Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In R. V. Book, editor, *RTA*, volume 488 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1991. ISBN 3-540-53904-2.

[6] A. Baumgartner and T. Kutsia. Unranked Second-Order Anti-Unification. Technical report, RISC, JKU Linz, March 2014. URL `http://www.risc.jku.at/publications/download/risc_4966/Baumgartner_Kutsia_2014.pdf`.

[7] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. A variant of higher-order anti-unification. In F. van Raamsdonk, editor, *RTA*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. ISBN 978-3-939897-53-8.

[8] H. Boley. Finite domains and exclusions as first-class citizens. In R. Dyckhoff, editor, *ELP*, volume 798 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 1993. ISBN 3-540-58025-5.

[9] P. E. Bulychev, E. V. Kostylev, and V. A. Zakharov. Anti-unification algorithms and their applications in program analysis. In A. Pnueli, I. Virbitskaite, and A. Voronkov, editors, *Ershov Memorial Conference*, volume 5947 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2009. ISBN 978-3-642-11485-4.

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

[11] A. L. Delcher and S. Kasif. Efficient parallel term matching and anti-unification. *J. Autom. Reasoning*, 9(3):391–406, 1992.

[12] R. W. Hasker. *The Replay of Program Derivations*. PhD thesis, University of Illionois at Urbana-Champaign, 1995.

[13] G. Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII, September 1976.

[14] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. Algorithms*, 40(2):212–233, 2001.

[15] U. Krumnack, A. Schwering, H. Gust, and K.-U. Kühnberger. Restricted higher-order anti-unification for analogy making. In M. A. Orgun and J. Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture*

Notes in Computer Science, pages 273–282. Springer, 2007. ISBN 978-3-540-76926-2.

[16] T. Kutsia, J. Levy, and M. Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014.

[17] H. Li and S. J. Thompson. Similar code detection and elimination for Erlang programs. In M. Carro and R. Peña, editors, *PADL*, volume 5937 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2010. ISBN 978-3-642-11502-8.

[18] J. Lu, J. Mylopoulos, M. Harao, and M. Hagiya. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

[19] F. Pfenning. Unification and anti-unification in the calculus of constructions. In *LICS*, pages 74–85. IEEE Computer Society, 1991.

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

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

[22] U. Schmid. *Inductive Synthesis of Functional Programs, Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning*, volume 2654 of *Lecture Notes in Computer Science*. Springer, 2003. ISBN 3-540-40174-1.

[23] A. Yamamoto, K. Ito, A. Ishino, and H. Arimura. Modelling semi-structured documents with hedges for deduction and induction. In C. Rouveirol and M. Sebag, editors, *ILP*, volume 2157 of *Lecture Notes in Computer Science*, pages 240–247. Springer, 2001. ISBN 3-540-42538-1.

[24] K. Zhang. Algorithms for the constrained editing problem between ordered labeled trees and related problems. *Pattern Recognition*, 28:463–474, 1995.

A Appendix

We first illustrate, step by step, how the algorithm $\mathfrak{G}$ computes the rigid lgg for the anti-unification problem in Example 3:

$$
\begin{aligned}
& \{x\colon f(a, f(b,b)) \triangleq (b, f(a,b), b);\ X\colon \circ \triangleq \circ;\ f\langle 1,2\rangle a\langle 1{\cdot}1, 2{\cdot}1\rangle b\langle 1{\cdot}2{\cdot}1, 2{\cdot}2\rangle\};\ \varnothing;\ \varepsilon \\
\Rightarrow_{\text{App-A}}\ & \{y_1\colon (a, f(b,b)) \triangleq (a,b);\ Y_1\colon \circ \triangleq \circ;\ a\langle 1,1\rangle b\langle 2{\cdot}1, 2\rangle\}; \\
& \{x\colon \epsilon \triangleq \epsilon;\ X\colon \circ \triangleq (b, \circ, b)\};\ \{x \mapsto f(Y_1(y_1))\} \\
\Rightarrow_{\text{Res-C}}\ & \{y_1\colon (a, f(b,b)) \triangleq (a,b);\ Y_1\colon \circ \triangleq \circ;\ a\langle 1,1\rangle b\langle 2{\cdot}1, 2\rangle\}; \\
& \{z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ, z_2\colon \epsilon \triangleq b;\ Z_2\colon \circ \triangleq \circ\};\ \{x \mapsto (z_1, f(Y_1(y_1)), z_2), X \mapsto \circ\} \\
\Rightarrow_{\text{Mer-S}}\ & \{y_1\colon (a, f(b,b)) \triangleq (a,b);\ Y_1\colon \circ \triangleq \circ;\ a\langle 1,1\rangle b\langle 2{\cdot}1, 2\rangle\}; \\
& \{z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\};\ \{x \mapsto (z_1, f(Y_1(y_1)), z_1), X \mapsto \circ\} \\
\Rightarrow^{\text{Spl-H}}_{\text{Clr-S}}\ & \{y_2\colon a \triangleq a;\ Y_2\colon \circ \triangleq \circ;\ a\langle 1,1\rangle,\ y_3\colon (f(b,b)) \triangleq b;\ Y_3\colon \circ \triangleq \circ;\ b\langle 1{\cdot}1, 1\rangle\}; \\
& \{z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\};\ \{x \mapsto (z_1, f(Y_2(y_2), Y_3(y_3)), z_1), X \mapsto \circ\} \\
\Rightarrow^{\text{App-A}}_{\text{Clr-S}}\ & \{y_4\colon \epsilon \triangleq \epsilon;\ Y_4\colon \circ \triangleq \circ;\ \mathfrak{e},\ y_3\colon (f(b,b)) \triangleq (b);\ Y_3\colon \circ \triangleq \circ;\ b\langle 1{\cdot}1, 1\rangle\}; \\
& \{z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\};\ \{x \mapsto (z_1, f(a(Y_4(y_4)), Y_3(y_3)), z_1), X \mapsto \circ\} \\
\Rightarrow^{\text{Sol-H}}_{\text{Clr-S}}\ & \{y_3\colon f(b,b) \triangleq b;\ Y_3\colon \circ \triangleq \circ;\ b\langle 1{\cdot}1, 1\rangle\}; \\
& \{z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\};\ \{x \mapsto (z_1, f(a, Y_3(y_3)), z_1), X \mapsto \circ\} \\
\Rightarrow_{\text{Abs-L}}\ & \{y_3\colon (b,b) \triangleq b;\ Y_3\colon f(\circ) \triangleq \circ;\ b\langle 1,1\rangle\}; \\
& \{z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\};\ \{x \mapsto (z_1, f(a, Y_3(y_3)), z_1), X \mapsto \circ\} \\
\Rightarrow_{\text{App-A}}\ & \{y_5\colon \epsilon \triangleq \epsilon;\ Y_5\colon \circ \triangleq \circ;\ \mathfrak{e}\};\ \{y_3\colon \epsilon \triangleq \epsilon;\ Y_3\colon f(\circ, b) \triangleq \circ,\ z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\}; \\
& \{x \mapsto (z_1, f(a, Y_3(b(Y_5(y_5)))), z_1), X \mapsto \circ\} \\
\Rightarrow^{\text{Sol-H}}_{\text{Clr-S}}\ & \varnothing;\ \{y_3\colon \epsilon \triangleq \epsilon;\ Y_3\colon f(\circ, b) \triangleq \circ,\ z_1\colon \epsilon \triangleq b;\ Z_1\colon \circ \triangleq \circ\}; \\
& \{x \mapsto (z_1, f(a, Y_3(b)), z_1), X \mapsto \circ\}.
\end{aligned}
$$

The proofs of the properties (e.g. termination, soundness, completeness and uniqueness) of $\mathfrak{G}$ can be found in the technical report [6]. Here we give only the proof of theorem 1 and the complexity result.

Theorem 1. *Let $\mathfrak{a} = a_1\langle I_1, J_1\rangle \ldots a_m\langle I_m, J_m\rangle$ be an alignment of $\tilde{s}$ and $\tilde{q}$ such that for all $1 \leqslant k \leqslant m$ the function symbol a_k is unique in $\tilde{s}$ and unique in $\tilde{q}$. $\mathfrak{a}$ is admissible iff there exists a generalization $\tilde{g}$ of $\tilde{s}$ and $\tilde{q}$ with $\mathcal{F}(\tilde{g}) = \{a_1, \ldots, a_m\}$.*

Proof. Let $\mathfrak{a} = a_1\langle I_1, J_1\rangle \ldots a_m\langle I_m, J_m\rangle$ be an alignment of $\tilde{s}$ and $\tilde{q}$ such that for all $1 \leqslant k \leqslant m$ the function symbol a_k is unique in $\tilde{s}$ and unique in $\tilde{q}$.

($\Leftarrow$) Assume $\tilde{g}$ is a generalization of $\tilde{s}$ and $\tilde{q}$ with $\mathcal{F}(\tilde{g}) = \{a_1, \ldots, a_m\}$. We will prove by contradiction that there are no collisions in $\mathfrak{a}$ (see definition of admissible alignment). Furthermore we assume that there are at least two elements in $\mathfrak{a}$ because the other cases are trivial by definition.

Case 1: Assume there is a collision at two elements of $\mathfrak{a}$. Then there exists $a_i, a_j \in \{a_1, \ldots, a_m\}$ such that a_i is an ancestor of a_j in $\tilde{s}$ while it is not an ancestor of a_j in $\tilde{q}$. We know that $\tilde{g}$ contains both symbols a_i and a_j.

Case 1.1: a_i is an ancestor of a_j in $\tilde{g}$. Then we have $a_i(\tilde{r}_1, t, \tilde{r}_2)$ being a subterm of $\tilde{g}$ where t is the term which contains a_j and $\tilde{r}_1, \tilde{r}_2$ are arbitrary hedges. By assumption there exists a substitution σ with $a_i, a_j \notin \mathcal{F}(\text{Ran}(\sigma))$ such that a_i is not an ancestor of a_j in $\tilde{g}\sigma$ but by the rule of substitution application $a_i(\tilde{r}_1, t, \tilde{r}_2)\sigma = a_i(\tilde{r}_1\sigma, t\sigma, \tilde{r}_2\sigma)$ the ancestor-descendant relation is preserved which is a contradiction.

Case 1.2: a_i is not an ancestor of a_j in $\tilde{g}$. Then we have $(\tilde{r}_1, t_1, \tilde{r}_2, t_2, \tilde{r}_3)$ being a subhedge of $\tilde{g}$ where t_1 is the term which contains a_i, t_2 is the term which contains a_j and $\tilde{r}_1, \tilde{r}_2, \tilde{r}_3$ are arbitrary hedges. By assumption there exists a substitution σ with $a_i, a_j \notin \mathcal{F}(\text{Ran}(\sigma))$ such that a_i is an ancestor of a_j in $\tilde{g}\sigma$ but this contradicts the rule of substitution application $(\tilde{r}_1, t_1, \tilde{r}_2, t_2, \tilde{r}_3)\sigma = (\tilde{r}_1\sigma, t_1\sigma, \tilde{r}_2\sigma, t_2\sigma, \tilde{r}_3\sigma)$ again.

Case 2: A collision appears at three elements. Let a_i, a_j, a_k be those elements. Without loss of generality, assume that a_i, a_j have a common ancestor ϕ which is not an ancestor of a_k in $\tilde{s}$ and let a_j, a_k have a common ancestor ψ which is not an ancestor of a_i in $\tilde{q}$. By assumption, $\tilde{g}$ contains all three symbols exactly once. It follows that there are substitutions σ_1, σ_2 with $a_i, a_j, a_k \notin \mathcal{F}(\text{Ran}(\sigma_1) \cup \text{Ran}(\sigma_2))$ where $\tilde{g}\sigma_1 = \tilde{s}$ and $\tilde{g}\sigma_2 = \tilde{q}$. By assumption, we know that $\tilde{g}\sigma_1$ contains a subhedge $(t_{ij}, \tilde{s}_k)$, with t_{ij} being the term which contains the symbols ϕ, a_i, a_j, and $\tilde{s}_k$ being a hedge which contains the symbol a_k. This implies that $\tilde{g}$ contains either ϕ or a context variable which can be instantiated to introduce ϕ. It follows that $\tilde{g}$ also contains a subhedge $(t'_{ij}, \tilde{s}'_k)$, with t'_{ij} being the term which contains the symbols a_i, a_j and $\tilde{s}'_k$ being a hedge which contains the symbol a_k. Similarly $\tilde{g}\sigma_2$ contains a subhedge $(\tilde{q}_i, t_{jk})$, with $\tilde{q}_i$ being a hedge which contains the symbol a_i and t_{jk} being the term which contains the symbols ψ, a_j, a_k. Further on $\tilde{g}$ either contains ψ or a context variable, say X, which can be instantiated to introduce ψ. Let us call this metavariable χ. As ψ is an ancestor of both, a_j and a_k in $\tilde{q}$, χ has to be above t'_{ij}. This is a contradiction to the assumption that ψ is not an ancestor of a_i in $\tilde{q}$.

($\Rightarrow$) Proof by construction of an algorithm which computes such a generalization for a given admissible alignment of two hedges. In section 4 we described this algorithm and proved its properties. □

Theorem 6 (Complexity of $\mathfrak{G}$). *The anti-unification algorithm $\mathfrak{G}$ has $O(n^2)$ time complexity and $O(n)$ space complexity, where n is the number of symbols in the input.*

Proof. Let $P_0;\ S_0;\ \sigma_0 = \{x : \tilde{s} \triangleq \tilde{q};\ X : \circ \triangleq \circ;\ \mathfrak{a}\};\ \varnothing;\ \varepsilon$ be the initial state of $\mathfrak{G}$ and $P_{i-1};\ S_{i-1};\ \sigma_{i-1} \Longrightarrow P_i;\ S_i;\ \sigma_i$ an arbitrary rule application. By theorem 5 we can arrange the rule applications as we like to obtain a maximal derivation. First the rules Spl-H, Abs-L/R, App-A and Sol-H are applied exhaustively. This are the only rules that operate on P_{i-1} and furthermore they do not have conditions on S_{i-1} or σ_{i-1} such that $P_0;\ S_0;\ \sigma_0 \Longrightarrow^+ \varnothing;\ S_j;\ \sigma_j$, for some j. Afterwards they are not applicable again and Res-C is applied exhaustively $\varnothing;\ S_j;\ \sigma_j \Longrightarrow^*_{\text{Res-C}} \varnothing;\ S_k;\ \sigma_k$. It transforms all the contexts in the store to terms. The rules Clr-S and Mer-S operate on S_k but they only remove AUPs from there, such that Res-C will not be applicable again. Finally we postpone the application of Mer-S to the very end, leading to a partial derivation $\varnothing;\ S_k;\ \sigma_k \Longrightarrow^*_{\text{Clr-S}} \varnothing;\ S_l;\ \sigma_l \Longrightarrow^*_{\text{Mer-S}} \varnothing;\ S_n;\ \sigma_n$ where no more rule is applicable because Mer-S does not introduce any AUPs, to which another rule could apply.

Now we analyze the first phase $P_0;\ S_0;\ \sigma_0 \Longrightarrow^+ \varnothing;\ S_j;\ \sigma_j$. The rule Spl-H splits an AUP into two AUPs and moves some parts into the store. The space overhead for

one application is constant because the two new AUPs in P_i and the one in S_i together exactly cover the original one from P_{i-1}, and four new variables are introduced. It can be applied $O(n)$ many times because both of the new AUPs are nonempty. It needs linear time (by the length of the alignment) to check for applicability and find the position for splitting the AUP. Also the context application needs linear time. The rules Abs-L/R are also applicable $O(n)$ many times. They strictly reduce the size of a hedge in P_i. The space overhead is zero. The test for applicability, the context application as well as the operations on the alignment need linear time. App-A is applicable $O(n)$ many times as well and one application needs linear time and constant space. It strictly reduces the size of a hedge in P_i and one application needs linear time, for the same reasons as the above rules. As Spl-H is applicable at most $O(n)$ many times and doubles the elements of P_i at each application and all the other rules do not increase the length of P_i, Sol-H is applicable $O(n)$ many times too. It follows that the number of introduced variables is $O(n)$ and the size of S_j is also bound by $O(n)$.

We compose the substitution σ_i immediately, but we only keep the mappings for x and X in σ_i, such that $\sigma_i = \{x \mapsto \tilde{r}_i, X \mapsto \tilde{c}_i\}$, for some $\tilde{r}_i, \tilde{c}_i$. As all the introduced variables in Spl-H and App-A are fresh, they only appear once in $\tilde{r}_i$ or $\tilde{c}_i$. This invariant of the first phase leads to $O(n)$ size of σ_i as well as $O(n)$ time for the substitution composition in Spl-H, App-A and Sol-H. All together we get $O(n^2)$ time complexity and $O(n)$ space complexity for the first phase.

The second phase is $\varnothing;\, S_j;\, \sigma_j \Longrightarrow^*_{\text{Res-C}} \varnothing;\, S_k;\, \sigma_k$. The rule Res-C is applicable only once per AUP leading to $O(n)$ many applications. The space overhead is constant at each application, introducing four fresh variables. It needs linear time at each application. We again compose σ_i immediately and for similar reasons as above, the substitution composition in Res-C only needs $O(n)$ time, leading to an overall time complexity of $O(n^2)$ and space complexity $O(n)$.

From the $O(n)$ size of the store, it follows that also the store cleaning rule is applicable $O(n)$ many times and the overall time complexity of this phase is $O(n^2)$, as we compose substitutions immediately like before. The space overhead for Clr-S is zero.

It remains to show that $\varnothing;\, S_l;\, \sigma_l \Longrightarrow^*_{\text{Mer-S}} \varnothing;\, S_n;\, \sigma_n$ only needs $O(n^2)$ time. Therefore we postpone the substitution composition. Comparing $O(n) * O(n)$ AUPs in the store needs $O(n^2)$ time and removing an AUP from the store needs constant time using a linked list. As the size of the store is bound by $O(n)$ and Mer-S removes one AUP at each application, there are $O(n)$ postponed substitution compositions. Each of them of constant size as they all are just variable renamings. This leads to linear space overhead and we have to compose $O(n)$ substitutions where each composition needs $O(n)$ time. This concludes our complexity analysis where we showed that the algorithm runs in $O(n^2)$ time using $O(n)$ space. □