A Generic Framework for Higher-Order Generalizations

David M. Cerna
FMV and RISC, Johannes Kepler University Linz, Austria
david.cerna@risc.jku.at

Temur Kutsia
RISC, Johannes Kepler University Linz, Austria
kutsia@risc.jku.at

Abstract

We consider a generic framework for anti-unification of simply typed lambda terms. It helps to compute generalizations which contain maximally common top part of the input expressions, without nesting generalization variables. The rules of the corresponding anti-unification algorithm are formulated, and their soundness and termination are proved. The algorithm depends on a parameter which decides how to choose terms under generalization variables. Changing the particular values of the parameter, we obtained four new unitary variants of higher-order anti-unification and also showed how the already known pattern generalization fits into the schema.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Higher order logic; Theory of computation → Type theory

Keywords and phrases anti-unification, typed lambda calculus, least general generalization

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.10

Funding Supported by the Austrian Science Fund (FWF) under the project P 28789-N32.

Acknowledgements We thank Tomer Libal for useful discussions on the early version of the paper.

1 Introduction

A term r is generalization of a term t, if t can be obtained from r by a variable substitution. The problem of finding common generalizations of two or more terms has been investigated quite intensively. The main idea is to compute least general generalizations (lggs) which maximally keep the similarities between the input terms and uniformly abstract over differences in them by new variables. For instance, if the input terms are $t = f(a, a)$ and $s = f(b, b)$, we are interested in their lgg $f(x, x)$. It gives more precise information about the nature of t and s than their other generalizations such as, e.g., $f(x, y)$ or just x. Namely, it shows that t and s not only have the same head f, but also each of them has its both arguments equal.

The technique of computing generalizations is called anti-unification. It was introduced in 1970s [17, 18] and saw a renewed interest in recent years (see, e.g., [3, 2, 11, 6, 1]), mostly motivated by various applications (see, e.g., [5, 13, 19, 20]).

Concerning anti-unification for higher-order terms, lggs are not unique and special fragments or variants of the problem have to be considered to guarantee uniqueness of lggs. Such special cases include generalizations with higher-order patterns [8, 7, 15, 16], object terms [9], restricted terms [21], etc. For instance, a pattern lgg of $\lambda x.f(g(x))$ and $\lambda x.h(g(x))$ is $\lambda x.Y(x)$, ignoring the fact that those terms have a common subterm $g(x)$. It happens because the pattern restriction requires free variables to apply to sequences of distinct bound variables. That's why we get a generalization in which the free variable Y applies to the bound variable x, and not to the more complex common subterm $g(x)$ of the given terms.

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 10; pp. 10:1–10:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Pattern generalizations have been successfully used, e.g., in term indexing [16] and in software code analysis for quick bug fixing [19]. Another advantage is that they can be computed efficiently, in linear time [8]. However, some problems require more expressive variants than patterns, as, for instance, the application of higher-order anti-unification in automatic detection of recursion schemes in functional programming [5]. Combination of increased expressive power and good computational properties was the motivation behind the introduction of functions-as-constructors terms (fc-terms) in higher-order unification [12]. We address a similar problem for anti-unification in this paper.

One difficulty comes from the fact that not all lggs are good characterizations of generalized terms. For instance, for $\lambda x.f(g(x))$ and $\lambda x.f(h(x))$ both $\lambda x.X(f(g(x)), f(h(x)))$ and $\lambda x.f(Y(g(x), h(x)))$ are lggs (where X and Y are fresh generalization variables), but the latter one is better, since it shows that the input terms have the common head f. This observation leads to the notion of top-maximal generalization, which keeps the maximally large common top part of the input terms. This is a very natural property, which, by default, was guaranteed for pattern lggs, but not necessarily for lggs in richer variants.

Another problem is related with nested generalization variables, which may affect least generality. In practice, such a nesting causes nondeterminism, an undesirable property. Once a difference between terms is detected, it should be abstracted by a generalization variable and no attempt to further generalize should be made under it. This leads to a variant of generalization, which we call shallow with respect to generalization variables, since these variables are not nested. We concentrate on computing top-maximal shallow lggs.

In this setting the interesting question is, what terms are permitted under generalization variables? For the pattern case they are distinct bound variables, but we want more expressive variants. Instead of coming up with different particular cases and designing special anti-unification algorithms for them, we formulate a generic algorithm which always computes top-maximal shallow generalizations, and prove its soundness and termination (Sect. 4). The particular cases can be obtained by instantiating a parameter in one of the rules of the algorithm, which is responsible for choosing terms under generalization variables. Changing the specific values of the parameter, we obtained several new unitary (i.e., with single lgg) variants of higher-order anti-unification: projection-based (Sect. 5.1), generalizations with common subterms (Sect. 5.2.1), relaxed fc (Sect. 5.2.2), and fc (Sect. 5.2.3). We also show how pattern generalization fits into the schema (Sect. 5.2.4). Completeness results for these variants are given. Additional variants of higher-order anti-unification can be developed using our schema by specifying how terms under generalization variables are chosen. For reader's convenience, some illustrative examples are put in the appendix.

2 Preliminaries

We consider simply-typed signature, where *types* are constructed from a set of *basic types* (denoted by δ) by the grammar $\tau ::= \delta \mid \tau \to \tau$, where $\to$ is associative to the right. *Variables* (denoted by $X, Y, Z, x, y, z, \ldots$) and *constants* (denoted by $a, b, c, f, g, h, \ldots$) have an assigned type. The set of variables is denoted by V and the set of constants by C. λ-*terms* (denoted by $t, s, r, \ldots$) are built using the grammar $t ::= x \mid c \mid \lambda x.t \mid t_1\ t_2$ where x is a variable and c is a constant, and are typed as usual. Terms of the form $(\ldots(h\ t_1)\ldots t_m)$, where h is a constant or a variable, will be written as $h(t_1, \ldots, t_m)$, and terms of the form $\lambda x_1.\cdots.\lambda x_n.t$ as $\lambda x_1, \ldots, x_n.t$. We use $\vec{x}$ as a short-hand for $x_1, \ldots, x_n$.

Other standard notions of the simply typed λ-calculus, like bound and free occurrences of variables, subterms, α-conversion, β-reduction, η-long β-normal form, etc. are defined as usual (see, e.g., [4]). $t{\downarrow_\eta}$ denotes the η-normal form of t. We denote the fact that t is a (strict) subterm of s using the infix binary symbol $\sqsubseteq$ ($\sqsubset$). Bound variables will be denoted by lowercase letters and free variables by capital letters. The symbols $\mathsf{fv}(t)$ and $\mathsf{bv}(t)$ are used to denote the sets of free and bound variables, respectively, of a term t. This notation extends to a set of terms as well. A term t is *closed* if $\mathsf{fv}(t) = \emptyset$.

By default, terms are assumed to be written in η-long β-normal form. Therefore, all terms have the form $\lambda x_1, \ldots, x_n.h(t_1, \ldots, t_m)$, where $n, m \geq 0$, h is either a constant or a variable, $t_1, \ldots, t_m$ also have this form, and the term $h(t_1, \ldots, t_m)$ has a basic type. For a term $t = \lambda x_1, \ldots, x_n.h(t_1, \ldots, t_m)$ with $n, m \geq 0$, its *head* is defined as $head(t) = h$.

When we write an equality between two λ-terms, we mean α, β and η equivalence.

Positions in λ-terms are defined with respect to their tree representation in the usual way, as string of integers. For instance, in the term $f(\lambda x.\lambda y.g(\lambda z.h(z, y), x), \lambda u.g(u))$, the symbol f stands in the position ϵ (the empty sequence), the occurrence of $\lambda x.$ stands in the position 1, the bound occurrence of y in 1.1.1.1.1.2, the bound occurrence of u in 2.1.1, etc. Hence, abstractions in this context are treated as symbols. We denote the symbol occurring in position p in a term t by $\mathsf{symb}(t, p)$ and the subterm of t at position p by $t|_p$. We write $p_1 \leq p_2$ if the position p_1 is a prefix of p_2. The strict part of this ordering is denoted by $<$. The set of all positions of a term t is denoted by $Pos(t)$.

Substitutions and their composition ($\circ$) are defined as usual. Namely, $(\sigma \circ \vartheta)X = \vartheta(\sigma(X))$. We extend the application of substitutions to terms in the usual way and denote it by postfix notation. Variable capture is avoided by implicitly renaming variables to fresh names upon binding. A substitution σ is more general than a substitution ϑ, denoted $\sigma \preceq \vartheta$, if there exists a substitution φ such that $\sigma \circ \varphi = \vartheta$. The strict part of this relation is denoted by $\prec$. The relation $\preceq$ is a partial order and generates the equivalence relation which we denote by $\simeq$. We overload $\preceq$ by defining $s \preceq t$ if there exists a substitution σ such that $s\sigma = t$.

3 Special forms of terms, generalization problems

In this section we first introduce certain special forms of terms and then discuss the generalization problem where the generalization terms may be of a special restricted form.

▶ **Definition 1** (Restricted terms). *Let B be a set of variables and s be a term such that $\mathsf{B} \cap \mathsf{bv}(s) = \emptyset$. Assume that distinct bound variables have distinct names in s. We say that a term t is* B-restricted *in s if t is a subterm of s such that* (i) *t is η-equivalent to some $t' \in \mathsf{B} \cup \mathsf{bv}(s)$, or* (ii) *$t = (f\, t_1 \cdots t_n)$, where $n > 0$, $f \in \mathsf{C} \cup \mathsf{B} \cup \mathsf{bv}(s)$ and each t_i, $1 \leq i \leq n$, is a B-restricted term in s.*

▶ **Definition 2** (Relaxed functions-as-constructors triples). *Let F and B be two disjoint sets of variables and s be a term such that $\mathsf{F} \cap \mathsf{bv}(s) = \mathsf{B} \cap \mathsf{bv}(s) = \emptyset$. It is also assumed that distinct bound variables have distinct names in s. We say that the triple $(\mathsf{F}, \mathsf{B}, s)$ is a* relaxed functions-as-constructors triple *or, shortly,* rfc-triple, *if the following conditions are satisfied:*

Argument restriction: *For all occurrences of $(X\, t_1 \cdots t_n)$ in s, where $X \in \mathsf{F}$, t_i is a B-restricted term in s for each $0 < i \leq n$.*

Local restriction: *For all occurrences of $(X\, t_1 \cdots t_n)$ in s, where $X \in \mathsf{F}$, for each $0 < i, j \leq n$, if $i \neq j$, then $t_i{\downarrow_\eta} \not\sqsubseteq t_j{\downarrow_\eta}$.*

Functions-as-constructors triples are rfc-triples obeying a global restriction:

▶ **Definition 3** (Functions-as-constructors triples). *Let F, B, and s be as in Definition 2 and $(\mathsf{F}, \mathsf{B}, s)$ be an rfc-triple. We say that it is a* functions-as-constructors triple *or, shortly,* fc-triple, *if the following extra condition is satisfied:*

Global restriction: *For each two different occurrences of terms $(X\, t_1 \cdots t_n)$ and $(Y\, s_1 \cdots s_m)$ in s with $X, Y \in \mathsf{F}$, for each $0 < i \leq n$, $0 < j \leq m$, we have $t_i{\downarrow_\eta} \not\sqsubset s_j{\downarrow_\eta}$.*

▶ **Definition 4** (Pattern triples). *Let F, B, and s be as in Definition 2 and $(\mathsf{F}, \mathsf{B}, s)$ be an rfc-triple. A triple is* pattern *if the following stronger form of the argument restriction holds:*

Argument restriction for patterns: *For all occurrences of $(X\, t_1 \cdots t_n)$ in s, where $X \in \mathsf{F}$, we have $t_i{\downarrow_\eta} \in \mathsf{B} \cup \mathsf{bv}(s)$ for each $0 < i \leq n$.*

Note that for patterns the local restriction reduces to checking whether $t_i{\downarrow_\eta}$'s are distinct bound variables, and the global restriction is automatically fulfilled. Hence, pattern triples are also a special case of fc-triples.

▶ **Definition 5** (Rfc-terms, fc-terms, patterns, shallow terms). *Let F be a set of variables and t be a term such that $\mathsf{F} \cap \mathsf{bv}(t) = \emptyset$. Then t is an* F-rfc-term *(resp.,* F-fc-term*,* F-pattern*), if $(\mathsf{F}, \emptyset, t)$ is an rfc-triple (resp., fc-triple, pattern-triple). We say that t is an* F-shallow term *if for every subterm $(X\, t_1 \cdots t_n)$ of t with $X \in \mathsf{F}$, we have $\mathsf{F} \cap (\cup_{i=1}^{n} \mathsf{fv}(t_i)) = \emptyset$.*

A term t is shallow*, if it is an $\mathsf{fv}(t)$-shallow term. Rfc-terms, fc-terms and patterns are defined analogously.*

Note that pattern coincides with the well-known higher-order patterns [14] fragment. Every pattern is an fc-term. Every fc-term is an rfc-term. Every rfc-term is a shallow term.

▶ **Example 6.** Consider the following terms:

$$
\begin{aligned}
&t_1 = \lambda x, y.\, f(X(x, \lambda z_1.y(z_1)),\ Y(\lambda z_2.y(z_2), x),\ \lambda u.Z(x, u))\\
&t_2 = \lambda x, y.\, f(X(g(x), p(\lambda z_1.y(z_1), \lambda z_2.y(z_2))), h(Y(g(x), g(h(x))))),\\
&t_3 = \lambda x.\, f(X(g(x), h(x)), h(Y(g(x), g(h(x))))),\\
&t_4 = \lambda x.\, f(X(x, g(a)), x), \qquad t_5 = \lambda x.\, f(X(x, g(x)), x), \qquad t_6 = \lambda x, y.\, f(X(Y(x), y)).
\end{aligned}
$$

t_1 is a pattern; t_2 is an fc-term but not a pattern; t_3 is an rfc-term but not an fc-term; t_4 is a shallow term but not an rfc-term, the argument restriction is violated; t_5 is a shallow term but not an rfc-term, the local restriction is violated; t_6 is not a shallow term.

A term t is called a *generalization* or an *anti-instance* of two terms t_1 and t_2 if $t \preceq t_1$ and $t \preceq t_2$. It is the *least general generalization* (lgg), also known as a *most specific anti-instance,* of t_1 and t_2, if there is no generalization s of t_1 and t_2 which satisfies $t \prec s$.

An *anti-unification triple* (shortly AUT) has the form $X(\vec{x}) : t \triangleq s$ where $\lambda\vec{x}.X(\vec{x})$, $\lambda\vec{x}.t$, and $\lambda\vec{x}.s$ are terms of the same type, t and s are in η-long β-normal form, and X does not occur in t and s. An *anti-unifier* of an AUT $X(\vec{x}) : t \triangleq s$ is a substitution σ such that $dom(\sigma) = \{X\}$ and $\lambda\vec{x}.X(\vec{x})\sigma$ is a term which generalizes both $\lambda\vec{x}.t$ and $\lambda\vec{x}.s$.

An anti-unifier σ of $X(\vec{x}) : t \triangleq s$ is *least general* (or *most specific*) if there is no anti-unifier ϑ of the same problem that satisfies $\sigma \prec \vartheta$. Obviously, if σ is a least general anti-unifier of an AUT $X(\vec{x}) : t \triangleq s$, then $\lambda\vec{x}.X(\vec{x})\sigma$ is a lgg of $\lambda\vec{x}.t$ and $\lambda\vec{x}.s$.

If r is a generalization of t and s, the *set of generalization variables (genvars)* of t and s in r is the set $\mathsf{genvar}(r, t, s) := \mathsf{fv}(r) \setminus (\mathsf{fv}(s) \cup \mathsf{fv}(t))$.

Our main interest is in generalizations, which retain the common parts of the given terms as much as possible, at least until the first differences in each branch during top-down traversal of the given terms. This intuition is formalized in the following definition:

▶ **Definition 7** (Top-maximal generalization). *Let s and t be terms of the same type in η-long β-normal form such that the bound variables are renamed uniformly: the ith bound variable in depth-first pre-order traversal in s and in t have the same name x_i. A common generalization r of s and t is their top-maximal common generalization, if the following conditions hold:*

- *If* $\mathsf{symb}(s,\epsilon) = \mathsf{symb}(t,\epsilon)$*, then* $\mathsf{symb}(r,\epsilon) = \mathsf{symb}(s,\epsilon)$*.*
- *If* $p \in Pos(s) \cap Pos(t)$ *such that* $\mathsf{symb}(s,q) = \mathsf{symb}(t,q)$ *for all positions* $q < p$ *and* $\mathsf{symb}(s,p) = \mathsf{symb}(t,p)$*, then* $p \in Pos(r)$ *and* $\mathsf{symb}(r,p) = \mathsf{symb}(s,p)$*.*
- *If* $p \in Pos(s) \cap Pos(t)$ *such that* $\mathsf{symb}(s,q) = \mathsf{symb}(t,q)$ *for all positions* $q < p$ *and* $\mathsf{symb}(s,p) \neq \mathsf{symb}(t,p)$*, then* $p \in Pos(r)$ *and* $\mathsf{symb}(r,p)$ *is a genvar.*

▶ **Example 8.** Let $s = \lambda x.f(g(x))$ and $t = \lambda x.f(h(x))$. Then $r_1 = \lambda x.X(f(g(x)), f(h(x)))$ is their shallow but not top-maximal generalization, while $r_2 = \lambda x.f(Y(g(x), h(x)))$ is both top-maximal and shallow. Also, $r_3 = \lambda x.f(Z(x))$ is a top-maximal shallow generalization.

3.1 Variants of higher-order anti-unification

In the literature, a *variant* of a unification or anti-unification problem is obtained by imposing restrictions on the form of solutions to the problem (in contrast to *fragments*, where the form of input is restricted). Here we define variants of higher-order anti-unification problem, which we will be solving in the coming sections.

The main variant we consider is what we call the top-maximal genvar-shallow variant:

Given: Two terms t and s of the same type in η-long β-normal form.
Find: A top-maximal generalization r of t and s such that r is a $\mathsf{genvar}(r,t,s)$-shallow term.

The problem statement implies that we are looking for r which is least general among all top-maximal genvar-shallow generalizations of t and s. There can still exist a term which is less general than r, is a top-maximal generalization of both s and t, but is not a genvar-shallow term. Also, there can exist a genvar-shallow generalization of s and t which is less general than r, but it is not a top-maximal generalization of s and t.

By imposing various conditions on r, we get other special problems such as, cs (common subterms), rfc, fc, and pattern variants of higher-order anti-unification.

First, we define the cs-variant. We need an auxiliary definition of extension of a set of terms by variables (bound in the context):

▶ **Definition 9.** *Let* B *be a set of variables and* S *be a set of terms such that* $\mathsf{bv}(S) \cap \mathsf{B} = \emptyset$*. By* B-extension *of* S *we understand the set* $S \cup (\mathsf{B} \setminus \mathsf{fv}(S))$*.*

Now, the definition of generalization with common subterms can be formulated as follows:

▶ **Definition 10** (CS-generalization)**.** *A generalization* r *of two terms* s *and* t *is called their* common-subterms generalization, *shortly* cs-generalization*, if it is a* $\mathsf{genvar}(r,s,t)$*-shallow top-maximal generalization of* t *and* s *satisfying the following condition:*

Common subterms condition: *Let* p *be a position in* r*,* $r|_p = X(r_1,\ldots,r_n)$ *for a genvar* X *and some terms* $r_1,\ldots,r_n$*, and* B *be the set of all variables bound by* λ *at positions above* p*, i.e.,* $\mathsf{B} := \{x \mid \mathsf{symb}(r,q) = \lambda x \text{ for some position } q < p\}$*. Then* $\{r_1,\ldots,r_n\}$ *is the* B'*-extension of some set of common subterms of* $s|_p$ *and* $t|_p$*, where* $\mathsf{B}' \subseteq \mathsf{B}$*.*

A cs-generalization r *of* s *and* t *is their least general cs-generalization (cs-lgg) if no cs-generalization* r' *of* s *and* t *satisfies* $r \prec r'$*.*

In this definition, top-maximality guarantees that all positions $q < p$ in r are also positions in t and s and $\mathsf{symb}(r,q) = \mathsf{symb}(t,q) = \mathsf{symb}(s,q)$ (modulo α-renaming of t and s). Therefore it may well happen that variables from B appear in $t|_p$ or in $s|_p$. Since r is a generalization, $r_1,\ldots,r_n$ must contain all variables from B that appear in $t|_p$ or in $s|_p$, for otherwise one can not get $t|_p$ and $s|_p$ from $r|_p$ by a substitution for X. Hence, actually, we have $\mathsf{B} \cap (\mathsf{fv}(t|_p) \cup \mathsf{fv}(s|_p)) \subseteq \mathsf{B}' \subseteq \mathsf{B}$ in Definition 10.

The cs-variant is the problem of computing cs-generalizations. The rfc, fc, and pattern variants are defined similarly, based on the following definition of the corresponding generalizations:

▶ **Definition 11** (RFC-, FC-, pattern-generalizations). *A generalization r of s and t is their rfc-generalization if r is an rfc- term. It is a least general rfc-generalization (rfc-lgg) of s and t if no rfc-generalization r' of s and t satisfies $r \prec r'$. The rfc-variant of higher-order anti-unification s the problem of computing rfc-generalizations. Fc- and pattern generalizations, lggs, and variants are defined in the same way.*

▶ **Example 12.** We bring examples of various lggs and show how they related to each other. Let $t = \lambda x.f(h(g(g(x))), h(g(x)), a)$ and $s = \lambda x.f(g(g(x)), g(x), h(a))$. Then

- $r_0 = \lambda x.f(X(h(g(g(x))), g(g(x))), X(h(g(x)), g(x)), X(a, h(a)))$ is a shallow top-maximal lgg of t and s.
- $r_1 = \lambda x.f(X(g(g(x))), X(g(x)), Z(a))$ is a cs-lgg of t and s. We have $r_1 \prec r_0$.
- $r_2 = \lambda x.f(X(g(g(x))), X(g(x)), Z)$ is a top-maximal rfc-lgg of t and s. We have $r_2 \prec r_1$.
- $r_3 = \lambda x.f(X(g(x)), Y(g(x)), Z)$ is a top-maximal fc-lgg of t and s and $r_3 \prec r_2$.
- $r_4 = \lambda x.f(X(x), Y(x), Z)$ is a top-maximal pattern-lgg of t and s. Also here $r_4 \prec r_3$.

More precise relationships between cs, rfc, fc, and pattern variants will be investigated in Section 5.2 below.

Top-maximality is an important requirement for an lgg to exist. If we do not require it, we might have $\preceq$-incomparable generalizations. For instance, in Example 8, r_1 and r_2 are not comparable by $\preceq$ and r_1 and r_3 are not either. On the other hand, two top-maximal shallow generalizations r_2 and r_3 are: $r_3 \prec r_2$.

For patterns, top-maximality means also least generality. This is not the case for shallow terms, as Example 8 shows. In fact, from that example we can see that top-maximality does not imply least generality for fc- and rfc-generalizations either, because r_1 and r_2 are both fc- and rfc-generalizations of s and t.

4 Generic anti-unification transformation rules

Transformation rules for anti-unification work on triples $A; S; r$, which we call *states*. Here A is a set of AUTs of the form $\{X_1(\overrightarrow{x_1}) : t_1 \triangleq s_1, \dots, X_n(\overrightarrow{x_n}) : t_n \triangleq s_n\}$ that are pending to anti-unify, S is a set of already solved AUTs (the *store*), and r is a generalization (computed so far). The goal is, given two terms t and s, compute a generalization r which is a $\mathsf{genvar}(r, t, s)$-shallow term. We aim at computing lggs.

The transformation rules given below are generic. At first, they help us to obtain a top-maximal genvar-shallow generalization. From it, we can obtain more special generalizations (e.g., rfc, fc, patterns) by deciding which kind of arguments are allowed under genvars.

▶ Remark 13. We assume that in the set $A \cup S$ each occurrence of λ binds a distinct name variable and that each generalization variable occurs in $A \cup S$ only once.

The set of transformations $\mathfrak{G}$ is defined by the following rules:

Dec: **Decomposition**

$$\{X(\vec{x}) : h(t_1, \dots, t_m) \triangleq h(s_1, \dots, s_m)\} \uplus A;\ S;\ r \Longrightarrow$$
$$\{Y_1(\vec{x}) : t_1 \triangleq s_1, \dots, Y_m(\vec{x}) : t_m \triangleq s_m\} \cup A;\ S;\ r\{X \mapsto \lambda\vec{x}.h(Y_1(\vec{x}), \dots, Y_m(\vec{x}))\},$$

where $Y_1, \dots, Y_n$ are fresh variables of the appropriate types.

Abs: **Abstraction**

$\{X(\vec{x}) : \lambda y.t \triangleq \lambda z.s\} \uplus A;\ S;\ r \Longrightarrow$
$\quad \{X'(\vec{x}, y) : t \triangleq s\{z \mapsto y\}\} \cup A;\ S;\ r\{X \mapsto \lambda\vec{x}, y.X'(\vec{x}, y)\}.$

where X' is a fresh variable of the appropriate type.

Sol: **Solve**

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

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

Mer: **Merge**

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

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

To compute generalizations for t and s, we start with the *initial state* $\{X : t \triangleq s\}; \emptyset; X$, where X is a fresh variable, and apply the transformations as long as possible. These *final states* have the form $\emptyset; S; r$, Then, the *result computed* by $\mathfrak{G}$ is r.

We use the letters C_t and C_s in the Solve rule because these terms resemble multi-contexts. Each of them have a form $\lambda z_1,\ldots,z_n.C'_t$ and $\lambda z_1,\ldots,z_n.C'_s$, where the bound variables $z_1,\ldots,z_n$ play the role of holes. In the store we keep the η-long β-normal form of $(C_t\, y_1 \cdots y_n)$ and $(C_s\, y_1 \cdots y_n)$. When applied to $q_1,\ldots,q_n$, C_t and C_s give, respectively, t and s. However, it should be emphasized that it is not the choice of C_t and C_s that might cause branching applications of Sol, but the choice of the subterms $q_1,\ldots,q_n$. Moreover, choosing different special forms of $q_1,\ldots,q_n$, we obtain different special versions of the anti-unification algorithm.

One can easily show that rules map a state to a state: For each expression $X(\vec{x}) : t \triangleq s \in A \cup S$, the terms $X(\vec{x})$, t and s have the same type, s and t are in η-long β-normal form, and X does not occur in t and s. Moreover, all genvars are distinct.

The property that each occurrence of λ in $A \cup S$ binds a unique variable is also maintained. It guarantees that in the Abs rule, the variable y is fresh for s. After the application of the rule, y will appear nowhere else in $A \cup S$ except $X'(\vec{x}, y)$ and, maybe, t and s.

▶ **Theorem 14.** *Let t and s be terms. Any sequence of transformations in $\mathfrak{G}$ starting from the initial state $\{X : t \triangleq s\}; \emptyset; X$ terminates and each computed result r is a $\mathsf{genvar}(r, t, s)$-shallow top-maximal generalization of t and s.*

Proof. Let the size of an AUT $Z(\vec{z}) : p \triangleq q$ be the number of symbols occurring in p or q, and the size of a set of AUTs be the multiset of sizes of AUTs it contains. Then the first three rules in $\mathfrak{G}$ strictly reduce the size of A. Mer applies when A is empty and strictly reduces the size of S. Hence, the algorithm terminates. The computed result is an $\mathsf{genvar}(r, t, s)$-shallow term, since no rule puts one generalization variable on top of another.

Proving that a computed result is a generalization is more involved. First, we prove that if $A_1; S_1; r \Longrightarrow A_2; S_2; r\vartheta$ is one step, then for any $X(\vec{x}) : t \triangleq s \in A_1 \cup S_1$, we have $X(\vec{x})\vartheta \preceq t$ and $X(\vec{x})\vartheta \preceq s$. Note that if $X(\vec{x}) : t \triangleq s$ was not transformed at this step, then this property trivially holds for it. Therefore, we assume that $X(\vec{x}) : t \triangleq s$ is selected and prove the property for each rule. We only illustrate it for Sol here, for the other rules the proof proceeds as in [8].

Sol: We have $\vartheta = \{X \mapsto \lambda\vec{x}.Y(q_1,\ldots,q_n)\}$, where $q_1,\ldots,q_n$ are distinct subterms in t or s. Let $\psi_1 = \{Y \mapsto \lambda y_1,\ldots,y_n.(C_t\, y_1 \cdots y_n)\}$ and $\psi_2 = \{Y \mapsto \lambda y_1,\ldots,y_n.(C_s\, y_1 \cdots y_n)\}$. Since $(C_t\, q_1 \cdots q_n) = t$, $(C_s\, q_1 \cdots q_n) = s$, and C_t and C_s do not contain any variable $x \in \vec{x}$, we get $X(\vec{x})\vartheta\psi_1 = X(\vec{x})\{X \mapsto \lambda\vec{x}.t,\ldots\} = t$, $X(\vec{x})\vartheta\psi_2 = X(\vec{x})\{X \mapsto \lambda\vec{x}.s,\ldots\} = s$, and, hence, $X(\vec{x})\vartheta \preceq t$ and $X(\vec{x})\vartheta \preceq s$.

We proceed by induction on the length l of the transformation sequence. We will prove a more general statement: If $A_0; S_0; r\vartheta_0 \Longrightarrow^* \emptyset; S_n; r\vartheta_0\vartheta_1\cdots\vartheta_n$ is a transformation sequence in $\mathfrak{G}$, then for any $X(\vec{x}) : t \triangleq s \in A_0 \cup S_0$ we have $X(\vec{x})\vartheta_1\cdots\vartheta_n \preceq t$ and $X(\vec{x})\vartheta_1\cdots\vartheta_n \preceq s$.

When $l = 1$, it is exactly the one-step case we just proved. Assume that the statement is true for any transformation sequence of the length n and prove it for a transformation sequence $A_0; S_0; \vartheta_0 \Longrightarrow A_1; S_1; \vartheta_0\vartheta_1 \Longrightarrow^* \emptyset; S_n; \vartheta_0\vartheta_1\cdots\vartheta_n$ of the length $n+1$.

Below the composition $\vartheta_i\vartheta_{i+1}\cdots\vartheta_k$ is abbreviated as ϑ_i^k with $k \geq i$. Let $X(\vec{x}) : t \triangleq s$ be an AUT selected for transformation at the current step. (Again, the property trivially holds for the AUTs which are not selected). We have to consider each rule, but, like above, only Sol is illustrated. For the other rules the proof is similar to the one in [8].

Sol: We have $X(\vec{x})\vartheta_1^1 = Y(q_1,\ldots,q_n)$ where Y is in the store. By the induction hypothesis, $Y(q_1,\ldots,q_n)\vartheta_2^n \preceq t$ and $Y(q_1,\ldots,q_n)\vartheta_2^n \preceq s$. Therefore, $X(\vec{x})\vartheta_1^n \preceq t$ and $X(\vec{x})\vartheta_1^n \preceq s$.

Finally, note that the obtained generalization is top-maximal, because the algorithm proceeds inserting common top-parts of the input terms in the generalization term as much as possible, and introduces a generalization variable only when a difference is encountered. ◀

▶ **Corollary 15.** *The result computed by $\mathfrak{G}$ for closed terms s and t is a shallow top-maximal generalization of s and t.*

As one can notice, the store keeps track of the differences between the original terms and suggests how to obtain them from the generalization. If the computed generalization for t and s is $\lambda\vec{x}.Y(q_1,\ldots,q_n)$ and the store contains the AUT $Y(y_1,\ldots,y_n) : (C_t\, y_1 \cdots y_n) \triangleq (C_s\, y_1 \cdots y_n)$, the substitution $\{Y \mapsto \lambda y_1,\ldots,y_n.(C_t\, y_1 \cdots y_n)\}$ gives t and $\{Y \mapsto \lambda y_1,\ldots,y_n.(C_s\, y_1 \cdots y_n)\}$ gives s.

▶ **Theorem 16** (Uniqueness modulo $\simeq$)**.** *Assume that the set $\{q_1,\ldots,q_n\}$, C_t, and C_s in the* Solve *rule are uniquely determined (modulo renaming of bound variables). Assume that for a given t' and s', $\mathfrak{G}$ can compute their generalizations r_1 and r_2 with different sequence of rule applications. Then $r_1 \simeq r_2$.*

Proof. In [8] it was proved that different order of the Mer rule application gives equivalent solutions, provided that the other rules are applied in a unique way to the selected AUT. The same for Abs and Dec rules. Sol can be applied also only in one way, since $\{q_1,\ldots,q_n\}$, C_t, and C_s are uniquely determined. Therefore, the theorem follows from Theorem 4 in [8]. ◀

4.1 The Solve rule

The Solve rule is generic and leaves room for special versions of the algorithm depending on how the subterms $q_1,\ldots,q_n$ are chosen. The choice of C_t and C_s is also important since they might affect applicability of the Merge rule. To illustrate the latter, consider the generalization derivation for the terms $\lambda x.f(g(x,a), g(a,x))$ and $\lambda x.f(h(x,a), h(a,x))$:

$$\{\lambda x.f(g(x,a), g(a,x)) \triangleq \lambda x.f(h(x,a), h(a,x))\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs,\ Dec}}$$

$$\{X_1(x) : g(x,a) \triangleq h(x,a),\ X_2(x) : g(a,x) \triangleq h(a,x)\};\ \emptyset;\ \lambda x.f(X_1(x), X_2(x)) \Longrightarrow_{\mathsf{Sol}}$$

$$\{X_2(x) : g(a, x) \triangleq h(a, x)\};\ \{Y(y_1, y_2) : g(y_1, y_2) \triangleq h(y_1, y_2)\};$$
$$\lambda x.f(Y(x, a), X_2(x)) \Longrightarrow_{\mathsf{Sol}}$$
$$\emptyset;\ \{Y(y_1, y_2) : g(y_1, y_2) \triangleq h(y_1, y_2),\ Z(z_1, z_2) : g(a, z_1) \triangleq h(z_2, z_1)\};$$
$$\lambda x.f(Y(x, a), Z(x, a)).$$

Here we chose C_t, C_s terms differently in two applications of Sol: First time, we had $q_1 = x$, $q_2 = a$ and we replaced in $t = g(x, a)$ and $s = h(x, a)$ all occurrences of the q's by fresh variables. Second time, in $t = g(a, x)$ and $s = h(a, x)$, we again took $q_1 = x$, $q_2 = a$, but the occurrence of q_2 in t is not replaced by a new variable. It resulted into the terminal store. However, the obtained generalization is not an lgg. An lgg would be $\lambda x.f(Y(x, a), Y(a, x))$.

If in the second application of Sol we again replaced all occurrences of the q's by fresh variables, we would make the step, leading to the mentioned lgg:

$$\emptyset;\ \{Y(y_1, y_2) : g(y_1, y_2) \triangleq h(y_1, y_2), Z(z_1, z_2) : g(z_2, z_1) \triangleq h(z_2, z_1)\};$$
$$\lambda x.f(Y(x, a), Z(x, a)) \Longrightarrow_{\mathsf{Mer}}$$
$$\emptyset;\ \{Y(y_1, y_2) : g(y_1, y_2) \triangleq h(y_1, y_2)\};\ \lambda x.f(Y(x, a), Y(a, x)).$$

Now we will formulate general rules for choosing the subterms $q_1, \dots, q_n$ and terms C_t and C_s in Sol. The rules will depend on a generic selection function. The function chooses subterms that satisfy a condition allowing them to appear under genvars. Our goal is to show that if $q_1, \dots, q_n$, C_t, and C_s in Sol are chosen according to the rules, then the computed generalization is least general among all similar generalizations.

Note that the condition of Sol implies that $q_1, \dots, q_n$ contain all variables from $\vec{x}$ that appear in t or in s, and contain none from $\vec{x}$ that appear neither in t nor in s.

We call (p_1, p_2) an *extended position pair* if p_1 and p_2 are either positions (positive integer sequences), or p_1 is a position and $p_2 = \bullet$ (a special symbol), or $p_1 = \bullet$ and p_2 is a position. The symbol $\bullet$ is not comparable with any position with respect to prefix ordering $\leq$. The latter is extended to pairs componentwise: $(p_1, p_2) \leq (l_1, l_2)$ iff $p_i \leq l_i$, $i \in \{1, 2\}$. Then for its strict part, $(p_1, p_2) < (l_1, l_2)$ iff either $p_1 < l_1$ and $p_2 \leq l_2$, or $p_1 \leq l_1$ and $p_2 < l_2$.

Given two terms t_1 and t_2, a *triple of subterm occurrence* in t_1 or t_2 is a triple (p_1, p_2, s) where (p_1, p_2) is an extended position pair such that

- if $p_i \in Pos(t_i)$, $i \in \{1, 2\}$, then $s = t_1|_{p_1} = t_2|_{p_2}$,
- if $p_1 \in Pos(t_1)$ and $p_2 = \bullet$, then $s = t_1|_{p_1}$ and s does not occur in t_2,
- if $p_1 = \bullet$ and $p_2 \in Pos(t_2)$, then $s = t_2|_{p_2}$ and s does not occur in t_1.

Now we define a selection function which will be used to define the ways we could select the terms $q_1, \dots, q_n$ in the Sol rule. It will depend on a special condition, a parameter, whose specific values will give specific variants of higher order generalizations in the next sections.

▶ **Definition 17.** *Given a set of variables $\{\vec{x}\}$ and terms t_1 and t_2, the* Select *function with the parametric condition* cond*,* $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t_1, t_2)$*, is the set of all subterm occurrence triples $Q = \{(p_1^1, p_1^2, s_1), \dots, (p_k^1, p_k^2, s_k)\}$ in t_1 or in t_2 such that*

1. $\mathsf{cond}(\{\vec{x}\}, t_1, t_2, Q)$ *holds.*
2. *If a variable from $\{\vec{x}\}$ appears in position p in t_1 (resp. in t_2), then there exist $p' \leq p$ and l such that $(p', l, t_1|_{p'}) \in Q$ (resp. $(l, p', t_2|_{p'}) \in Q$).*
3. *For all $(p_1, p_2, s) \in Q$, there is no $(p_1', p_2', s') \in Q$ such that $(p_1', p_2') < (p_1, p_2)$ holds.*

Now we define general rules for choosing $q_1, \dots, q_n$, C_t, and C_s in Sol. Let $\{X(\vec{x}) : t \triangleq s\} \uplus A$ be the set of AUTs on which Sol operates. Let $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s) = Q$. The rule of choosing $q_1, \dots, q_n$ in Sol is the following:

QR: $\{q_1, \dots, q_n\} = \{q \mid (p_1, p_2, q) \in Q$ for some p_1 and $p_2\}$.

For the rule for C_t and C_s, we will need a special notation. By $t[p \mapsto x]$ we denote a term obtained from t by replacing its subterm at position p by the variable x. If the position p does not exist in t, or if $p = \bullet$, then $t[p \mapsto x] = t$.

CR: Let $Q = \{(p_{i1}, l_{i1}, q_i), \dots, (p_{ik}, l_{ik}, q_i) \mid 1 \leq i \leq n\}$. Then:

$$C_t = \lambda y_1, \dots, y_n.\, t[p_{11} \mapsto y_1] \cdots [p_{1k_1} \mapsto y_1] \cdots [p_{n1} \mapsto y_n] \cdots [p_{nk_n} \mapsto y_n]$$
$$C_s = \lambda y_1, \dots, y_n.\, s[l_{11} \mapsto y_1] \cdots [l_{1k_1} \mapsto y_1] \cdots [l_{n1} \mapsto y_n] \cdots [l_{nk_n} \mapsto y_n].$$

CR says that no eligible occurrence of each q_i is kept in C_t and C_s. In those positions, if they still exist in C_t and C_s, we have the variable y_i.

The next step is to define special cases of the generic algorithm by specifying cond.

5 Special cases

The special cases of the generic algorithm are obtained by deciding what kind of subterms from the input terms we would like to preserve in the generalization under genvars.

We distinguish between two classes of (top-maximal, genvar-shallow) generalizations:

(a) Those which do not care about common subterms under different-head terms to be generalized but, rather, take both different-head terms entirely in the generalization, and

(b) Those which try to find similarities under different-head terms to be generalized and select their certain common subterms to the generalization.

We call the first class *projection-based variant*, since the generalizations there give original terms by projection substitutions. The second class corresponds to the *common-subterm variant*, introduced earlier. There are several subcategories in this class, as we will see.

5.1 Projection-based variant

This is the simplest case. If t and s appear in the Sol rule, we should keep both of them in the generalization. Therefore, we specify Select and, consequently, **QR** and **CR** as follows:

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: cond is always true.
The instance of QR: $q_1 = t$, $q_2 = s$.
The instance of CR: $C_t = \lambda z_1, z_2.\, z_1$, $C_s = \lambda z_1, z_2.\, z_2$.

After applying Sol with $\mathsf{Select}_{\mathsf{cond}}$ specified above, the new AUT in the store will have the form $Y(y_1, y_2) : y_1 \triangleq y_2$. By the exhaustive application of the Mer rule we get that if the computed result contains genvars, then it contains only one such variable (maybe with multiple occurrences). Therefore, we can ignore Mer and use the same variable. The store is not needed at all, since merging is superfluous and the anti-unifiers are fixed to projections. The instance of Sol rule is denoted by Sol-PrB, and the obtained algorithm by $\mathfrak{G}_{\mathrm{prb}}$.

Generalizations that retain both terms whose heads are different are called imitation-free generalizations in [10] (where only second-order generalizations are considered), motivated from [9]. The name originates from the fact that one does not need imitation anti-unifiers. We prefer the name projection-based, since it directly indicates how the anti-unifiers look.

▶ **Theorem 18.** $\mathfrak{G}_{\mathrm{prb}}$ *computes a projection-based* genvar(r, t, s)*-shallow top-maximal generalization r of the input terms t and s in linear time.*

Proof. Top-maximality and shallowness of r follow from Theorem 14, the projection-based property from the instances on **QR** and **CR**, and the linear time complexity from the fact that each symbol in the input is processed only once, when it is put into the generalization. ◀

▶ **Theorem 19** (Completeness of $\mathfrak{G}_{\mathrm{prb}}$). *If r_0 is a projection-based $\mathsf{genvar}(r_0, t, s)$-shallow top-maximal generalization of t and s, then $\mathfrak{G}_{\mathrm{prb}}$ computes r, starting from t and s, such that $r_0 \preceq r$.*

Proof sketch. Top maximal projection-based genvar-shallow generalizations of t and s can differ from each other only by the number of duplicates among genvars. $\mathfrak{G}_{\mathrm{prb}}$ maximizes their sharing. Hence, r is less general than any projection-based genvar-shallow generalization. ◀

▶ **Corollary 20.** *Projection-based variant of higher-order anti-unification is unitary.*

Proof. Follows from Theorem 19 and Theorem 16, since the specification of instances of **QR** and **CR** makes the q's and C's in the Solve rule uniquely determined. ◀

Interestingly, projection-based generalizations are least general among all top-maximal generalizations that do not nest genvars:

▶ **Theorem 21.** *Let r_1 and r_2 be respectively $\mathsf{genvar}(r_1, t, s)$- and $\mathsf{genvar}(r_2, t, s)$-shallow top-maximal generalizations of t and s. Assume that r_1 is projection-based. Then $r_2 \preceq r_1$.*

Proof. By the definition of projection-based generalization, the symbols occurring above the positions of genvars are common for t and s. Top-maximality requires that common symbols are retained in the generalization. Let r_1 and r_2 contain a genvar in position p. Since they are top-maximal, all symbols above p are common in s and t. Since r_1 is $\mathsf{genvar}(r_1, t, s)$-shallow and projection-based, $r_1|_p$ should have a form $Y(t|_p, s|_p)$, where Y is a genvar.

Also, r_2 is $\mathsf{genvar}(r_1, t, s)$-shallow. Therefore, $r_2|_p$ has a form $X(q_1, \dots, q_n)$, where each q_i is a subterm of $t|_p$ or $s|_p$. Since r_2 is a generalization of t and s, there exist substitutions σ_1 and σ_2 such that $X(q_1, \dots, q_n)\sigma_1 = t|_p$ and $X(q_1, \dots, q_n)\sigma_2 = s|_p$. Then $r_1|_p$ can be obtained from $r_2|_p$ by the substitution $\{X \mapsto \lambda y_1, \dots, y_n. Y(X(y_1, \dots, y_n)\sigma_1, X(y_1, \dots, y_n)\sigma_2)\}$.

Because of top-maximality and shallowness, r_1 and r_2 have genvars in the same positions. The projection-based property implies that r_1 contains only one genvar, which we denoted by Y above. Repeating the above reasoning for each genvar position finishes the proof. ◀

A disadvantage of projection-based generalizations is that if two subterms do not have the same head, projection-based generalization does not focus on their common parts. However, often it is interesting to report the commonalities between such subterms. This is what common-subterm generalization is about.

5.2 Generalization with common subterms

5.2.1 CS-variant

In the definition of cs-generalizations (Definition 10) we just required the set $\{r_1, \dots, r_n\}$ to originate from *some set of common subterms* of $s|_p$ and $t|_p$. Such a relaxed definition will allow us in the next sections to relate the cs-variant to more specific categories such as rfc-, fc-, and pattern variants. However, for the Select function we need a stronger way to choose $\{r_1, \dots, r_n\}$, since we aim at computing lggs. Therefore, we introduce the notion of position-maximal common subterm of two terms:

▶ **Definition 22.** *Let t_1, t_2, and s be terms such that for some positions p_1 of t_1 and p_2 of t_2, we have $t_1|_{p_1} = t_2|_{p_2} = s$. We say that s is a* (p_1, p_2)-maximal common subterm *of t_1 and t_2 if*

- $p_1 = \epsilon$ *or* $p_2 = \epsilon$, *or*
- $p_1 = p'_1.i_1$ *and* $p_2 = p'_2.i_2$ *for some* p'_1, i_1, p'_2, *and* i_2, *and* $t_1|_{p'_1} \neq t_2|_{p'_2}$.

A common subterm of two terms is position-maximal *if it is their (p_1, p_2)-maximal common subterm for some positions p_1 of t_1 and p_2 of t_2.*

The set of position-maximal common subterm occurrence triples of t_1 and t_2 is defined as $pmcso(t_1, t_2) := \{(p_1, p_2, s) \mid s$ is a (p_1, p_2)-maximal common subterm of t_1 and $t_2\}$.

Given an $pmcso(t_1, t_2)$ and a set of variables χ such that no bound variable occurring as the term of a triple of $pmcso(t_1, t_2)$ is in χ, an χ-extension *of $pmcso(t_1, t_2)$ is the set*

$$\begin{aligned} pmcso_\chi(t_1, t_2) &:= pmcso(t_1, t_2) \\ &\cup \{(p, \bullet, x) \mid x \in \chi \setminus \mathsf{fv}(t_2),\ p \text{ is the first position with } t_1|_p = x\} \\ &\cup \{(\bullet, p, x) \mid x \in \chi \setminus \mathsf{fv}(t_1),\ p \text{ is the first position with } t_2|_p = x\}. \end{aligned}$$

Remark. Since it is enough to have one occurrence of $(p, \bullet, x)$ and $(\bullet, p, x)$, it does not matter how p is computed. We can, e.g., assume that it is the first leftmost-outermost position.

▶ **Example 23.** The set of all position-maximal common subterms of $f(g(x), g(x), g(g(x)))$ and $h(g(g(x)), a, b)$ is $\{g(x), g(g(x))\}$, where $g(x)$ is the $(1, 1.1)$- and $(2, 1.1)$-maximal common subterm, and $g(g(x))$ is the $(3, 1)$-maximal common subterm.

Now, we obtain the special case of the Sol rule for position-maximal common subterms by choosing cond and, as a consequence, **QR**, as follows:

Specifying Select$_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff $Q = pmcso_{\{\vec{x}\}}(t, s)$.
The instance of QR: $\{q_1, \ldots, q_n\}$ is the $\{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$-extension of the set of all position-maximal common subterms of t and s.

cond and the item 2 of the definition of Select (Definition 17) imply that we have $\{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$-extension in the instance of **QR**. Without item 2, it would be just $\{\vec{x}\}$-extension. Note that for computing cs-generalizations, if would be sufficient to take $\{\vec{x}\}$-extensions, but we aim at computing cs-lggs, that's why we would keep only the necessary variables from $\{\vec{x}\}$ in generalizations. The necessary ones are those that appear in t or in s.

Yet another remark, which concerns the difference between cs-generalizations and Select is that the q's we get from **QR** form the set of *all position-maximal common subterms* of the terms to be generalized, while in cs-generalization the free variables apply to *some set of common subterms* of those terms. This difference is motivated by our wish to have, on the one hand, rfc-, fc-, and pattern-lggs later as special cs-generalizations and, on the other hand, to compute cs-lggs by the specific version of Sol. The specified instance of cond does not imply any special form of C_t and C_s. They are like it was defined in **CR**.

The obtained instance of Sol is denoted by Sol-CS, and the obtained algorithm by $\mathfrak{G}_{\mathrm{cs}}$. We get the theorem, in which (and in the analogous theorems for rfc, fc, and patterns below) n is the size of the input:

▶ **Theorem 24.** $\mathfrak{G}_{\mathrm{cs}}$ *computes a cs-generalization of two terms in time $O(n^3)$.*

Proof. Top-maximality and genvar-shallowness follow from Theorem 14. The cs-generalization property follows from the instance of **QR**. Selecting position-maximal common subterms from two terms can take quadratic time, and $\mathfrak{G}_{\mathrm{cs}}$ can perform this operation linearly many times. Hence the cubic time complexity. Merging at the end can not make it worse. ◀

▶ **Theorem 25** (Completeness of $\mathfrak{G}_{\mathrm{cs}}$). *Let r_0 be a cs-generalization of t and s. Then $\mathfrak{G}_{\mathrm{cs}}$ computes a generalization r of t and s such that $r_0 \preceq r$.*

Proof sketch. Cs-generalizations differ from each other by the amount of position-maximal common subterms they take in the generalization, and by the number of duplicate generalization variables. The Select function makes sure that $\mathfrak{G}_{\mathrm{cs}}$ puts in generalizations as many position-maximal common subterms as possible, and the exhaustive application of Mer makes all possible sharings of genvars. These arguments imply that $r_0 \preceq r$. ◀

▶ **Corollary 26.** *Cs-variant of higher-order anti-unification is unitary.*

Proof. Follows from Theorem 25 and Theorem 16, since the specification of instances of **QR** and **CR** makes the q's and C's in the Solve rule uniquely determined. ◀

▶ **Example 27.** Let $t = \lambda x_1.f(g_1(x_1, a), g_2(\lambda x_2.h(x_2)))$, $s = \lambda y_1.f(h_1(a, a), h_2(\lambda y_2.h(y_2)))$. $\mathfrak{G}_{\mathrm{cs}}$ gives $\lambda x_1.f(Z_1(x_1, a), Z_2(\lambda x_2.h(x_2)))$. (See Example 44 in Appendix.) If we had $\{\vec{x}\}$-extension in the instance of **QR**, we would get $\lambda x_1.f(Z_1(x_1, a), Z_3(x_1, \lambda x_2.h(x_2)))$, which is more general than $\lambda x_1.f(Z_1(x_1, a), Z_2(\lambda x_2.h(x_2)))$.

▶ **Example 28.** let $t = \lambda x.f(g(x), h(x, a))$ and $s = \lambda y.h(g(y), a)$. Then $\mathfrak{G}_{\mathrm{cs}}$ gives the final state $\emptyset$; $\{Y(y_1, y_2, y_3) : f(y_2, h(y_1, y_3)) \triangleq h(y_2, y_3)\}$; $\lambda x.Y(x, g(x), a)$.

In some applications, it is desirable that the arguments of free variables are not subterms of each other. This requirement leads to generalization for (relaxed) fc- and patterns. These special cases also rely on position-maximal common subterm computation, but the obtained set is filtered. For those variants, in the sections below, we assume that the input terms are closed. Otherwise we will need to add some extra tests to make sure that free variables from the input appear in the generalization only if they do not violate the rfc-, fc-, or patterns restrictions. It will just make things more cumbersome without giving any special insights about the problem. Therefore, for simplicity, we prefer to work with closed input.

For closed input terms, genvar-shallow generalizations are just shallow generalizations. Therefore, below we will mention only the latter.

5.2.2 RFC-variant

From Definition 11 it follows that rfc-generalizations are shallow, but not necessarily top-maximal. Moreover, even top-maximal rfc-generalizations do not have to be cs-generalizations. For instance, if $s = \lambda x.f(h_1(g_1(x)), h_1(g_2(x)))$ and $t = \lambda x.f(h_2(g_1(x)), h_2(g_2(x)))$, then $r = \lambda x.f(X(g_1(x), g_2(x)), X(g_1(x), g_2(x)))$ is an rfc-generalization of s and t, but it is not a cs-generalization. However, top-maximal rfc-lggs are cs-generalizations:

▶ **Theorem 29.** *Let r be a top-maximal rfc-lgg of s and t. Then r is their cs-generalization.*

Proof. Let $X(r_1, \dots, r_n) = r|_p$, where X is a genvar. Since r is an rfc-term, $(\{X\}, \mathsf{B}, X(r_1, \dots, r_n))$ is an rfc-triple, where B is the set of variables bound by λ above the position p. The terms $r_1, \dots, r_n$ should contain all variables from $\mathsf{B} \cap (\mathsf{fv}(s|_p) \cup \mathsf{fv}(t|_p))$, otherwise $X(r_1, \dots, r_n)$ can not generalize $s|_p$ and $t|_p$. Moreover, $r_1, \dots, r_n$ should be common subterms of $s|_p$ and $t|_p$. Otherwise it will violate the assumption that r is an lgg: if, say, r_n is not a common subterm of $s|_p$ and $t|_p$ (in the sense mentioned in the previous section), then $Y(r_1, \dots, r_{n-1})$ will be again a generalization of $s|_p$ and $t|_p$, but less general than $X(r_1, \dots, r_n)$. By the assumption, r is top-maximal. As an rfc-generalization, r is shallow. Since p was an arbitrary position with a genvar, all these conditions imply that r is a cs-generalization of s and t. ◀

Since we aim at computing rfc-lggs, we can take an instance of the Sol rule so that it generates only those rfc-generalizations that are cs-generalizations. We call them {cs, rfc}-generalizations. In them, in addition to the common-subterms condition in Definition 10, the subterms of genvars should satisfy argument and local restrictions. It leads to the instance of Select, in which cond starts from the set Q as in cs-generalizations, and removes from it those terms that violate argument and local restrictions:

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$: $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff Q is obtained from the set $pmcso_{\{\vec{x}\}}(t, s)$ by removing from it

(a) all triples (p_1, p_2, q) where $q{\downarrow_\eta}$ is not $\{\vec{x}\}$-restricted in $t \triangleq s$[1] and

(b) all triples (p_1^i, p_2^i, q_i) for which there exists $(p_1^j, p_2^j, q_j) \in pmcso_{\{\vec{x}\}}(t, s)$ such that $q_j{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$.

The instance of QR: $\{q_1, \dots, q_n\}$ is the largest set of position-maximal common subterms of t and s whose η-normal forms are $\{\vec{x}\} \cap (\mathsf{fv}(t) \cap \mathsf{fv}(s))$-restricted in t or in s, and none of those η-normal forms are subterms of each other.

Similar to the previous section, the terms C_t and C_s do not have any special form. Defining the q's in this way, it is easy to see that $Y(q_1, \dots, q_n)$, in the generalization computed by Sol satisfies both the argument restriction and the local restriction. The obtained rule is called Sol-RFC, and the algorithm $\mathfrak{G}_{\mathrm{rfc}}$. We get the theorem:

▶ **Theorem 30.** *$\mathfrak{G}_{\mathrm{rfc}}$ computes a top-maximal {cs, rfc}-generalization in time $O(n^3)$.*

Proof. From Theorem 14 we get top-maximality and shallowness (since the input is assumed to be closed). The {cs, rfc}-property follows from the instance of **QR**. The $O(n^3)$ time of computing cs-generalizations dominates the time needed to filter out subterms that violate the rfc-property (since argument and local restrictions are checked in quadratic time). ◀

▶ **Theorem 31** (Completeness of $\mathfrak{G}_{\mathrm{rfc}}$)**.** *Let r_0 be a top-maximal rfc-generalization of t and s. Then $\mathfrak{G}_{\mathrm{rfc}}$ computes a generalization r of t and s such that $r_0 \preceq r$.*

Proof sketch. Among two top-maximal rfc-generalizations, the one with all position-maximal common subterms and all possible sharings of genvars is less general. ◀

▶ **Corollary 32.** *Rfc-variant of higher-order anti-unification is unitary.*

Proof. Follows from Theorem 31 and Theorem 16, since the specification of instances of **QR** and **CR** makes the q's and C's in the Solve rule uniquely determined. ◀

▶ **Example 33.** Let $t = \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x))))$, $s = \lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))$. Then $\mathfrak{G}_{\mathrm{rfc}}$ stops with the final state $\emptyset$; $\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a), Y_2(y_2) : h_2(y_2) \triangleq h_4(y_2)\}$; $\lambda x.f(Y_1(g(x)), Y_2(g(g(x))))$.

5.2.3 FC-variant

Fc-generalizations are also rfc-generalizations and, hence, the properties of rfc-generalizations are valid for fc-generalizations as well. The counterpart of Theorem 29 holds. Analogously to the rfc case, here we aim at computing {cs, fc}-generalizations.

The peculiarity here is that we have to take into account the global condition of fc-terms. Therefore, we need to impose a strategy on the application of the (yet to be defined) instance of the Sol rule: It should be applied only if no other rule applies. Let at this moment A

[1] We look here at $t \triangleq s$ as a term, also in the selection functions for fc- and pattern generalizations later.

be the set $\{X_1(\overrightarrow{x_1}) : t_1 \triangleq s_1, \dots, X_k(\overrightarrow{x_k}) : t_k \triangleq s_k\}$. Let M_i, $1 \leq i \leq m$, be the set of all position-maximal common subterms of t_i and s_i, and let $M = \cup_{i=1}^{k} M_i$. Then we formulate the instance of Select, in which cond takes into account M, and filters out terms violating argument, local, and global restrictions:

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$**:** $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff Q is obtained from the set $pmcso_{\{\vec{x}\}}(t, s)$ by

- **(a)** removing all $(p_1, p_2, q) \in pmcso_{\{\vec{x}\}}(t, s)$ where $q{\downarrow_\eta}$ is not $\{\vec{x}\}$-restricted in $t \triangleq s$ and
- **(b)** replacing all $(p_1^i, p_2^i, q_i) \in pmcso_{\{\vec{x}\}}(t, s)$ by (p_1^j, p_2^j, q_j), where $q_j{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$ and $q_j \in M$.

Note that the condition **(b)** here includes as a special case the condition **(b)** from the Select instance for rfc-generalizations. This selection function, by Definition 17, leads to the following instance of **QR**:

The instance of QR: Let Q be the largest set of position-maximal common subterms of t and s whose η-normal forms are $\{\vec{x}\} \cap (\mathsf{fv}(t) \cap \mathsf{fv}(s))$-restricted in t or in s. Then $\{q_1, \dots, q_n\}$ is obtained from Q by replacing all $q_i \in Q$ by $q_j \in M$, if $q_j{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$.

Since we take into account the whole of M when deciding which subterms to keep under the genvars, the global restriction of fc-terms is satisfied. Similar to the cs- and rfc-variants, the terms C_t and C_s here do not have any special form. The obtained instance of Sol is denoted by Sol-FC, and the algorithm by $\mathfrak{G}_{\mathrm{fc}}$. The theorems below can be proved similarly to their rfc-counterparts:

▶ **Theorem 34.** *$\mathfrak{G}_{\mathrm{fc}}$ computes a top-maximal {cs, fc}-generalization in time $O(n^3)$.*

▶ **Theorem 35** (Completeness of $\mathfrak{G}_{\mathrm{fc}}$)**.** *Let r_0 be a top-maximal fc-generalization of t and s. Then $\mathfrak{G}_{\mathrm{fc}}$ computes a generalization r of t and s such that $r_0 \preceq r$.*

▶ **Corollary 36.** *Fc-variant of higher-order anti-unification is unitary.*

▶ **Example 37.** For terms in Example 33, $\mathfrak{G}_{\mathrm{fc}}$ stops with the final state $\emptyset$; $\{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),\ Y_2(y_2) : h_2(g(y_2)) \triangleq h_4(g(y_2))\}$; $\lambda x.f(Y_1(g(x)), Y_2(g(x)))$.

5.2.4 Pattern variant

Similarly to rfc- and fc-generalizations, pattern generalizations are shallow but not necessarily top-maximal (and, consequently, not cs-generalizations). For instance, $\lambda x, y.f(X(x, y))$ is a pattern generalization of $s = t = \lambda x, y.f(g(x))$, which is neither top-maximal nor cs-generalization. However, pattern lggs are top-maximal and retain common subterms (note the difference from rfc- and fc-generalization, where lggs are not necessarily top-maximal):

▶ **Theorem 38.** *A least general pattern generalization of two terms is their cs-generalization.*

Proof. Top-maximality of pattern lgg follows from completeness of pattern generalization algorithm described in [7, 8]. The rest of the proof is similar to the proof of Theorem 29. ◀

Specifying $\mathsf{Select}_{\mathsf{cond}}(\{\vec{x}\}, t, s)$**:** $\mathsf{cond}(\{\vec{x}\}, t, s, Q)$ is true iff Q is obtained from the set $pmcso_{\{\vec{x}\}}(t, s)$ by

- **(a)** removing all $(p_1, p_2, q) \in pmcso_{\{\vec{x}\}}(t, s)$ where $q{\downarrow_\eta}$ is not $\{\vec{x}\}$-restricted in $t \triangleq s$ and
- **(b)** replacing all $(p_1^i, p_2^i, q_i) \in pmcso_{\{\vec{x}\}}(t, s)$ by (p_1^j, p_2^j, x), where $x{\downarrow_\eta} \sqsubset q_i{\downarrow_\eta}$ and $x \in \{\vec{x}\}$.

We wrote Select in this form to relate it to the selection functions of the other common subterms based generalizations (cs, rfc, fc). It leads to the instances of **QR** and **CR**:

The instance of QR: $\{q_1, \dots, q_n\} = \{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$.
The instance of CR: $C_t = t$, $C_s = s$, $y_i = q_i$.

The obtained instance of Sol is denoted by Sol-P, and the obtained algorithm by $\mathfrak{G}_{\mathrm{pat}}$. It is, in fact, the algorithm from [8], for the closed input. It is complete. The theorem below is also from [8]:

▶ **Theorem 39.** *$\mathfrak{G}_{\mathrm{pat}}$ computes a least general pattern generalization in time $O(n)$.*

It is known from [8] that pattern variant of higher-order anti-unification is unitary. It can be also seen from Theorem 16 and the definitions of the instances of **QR** and **CR** above, which makes the choice of the q's and C's in Sol unique.

6 Conclusion

We described a general framework for computing top-maximal genvar-shallow generalizations of two terms and proved its properties. Appropriate instantiation of the framework gives concrete instances of variants of higher-order anti-unification. By instantiations, we obtained four new unitary variants of higher-order generalization.

References

1 Hassan Aït-Kaci and Gabriella Pasi. Fuzzy Unification and Generalization of First-Order Terms over Similar Signatures. In Fabio Fioravanti and John P. Gallagher, editors, *LOPSTR 2017*, volume 10855 of *LNCS*, pages 218–234. Springer, 2017. doi:10.1007/978-3-319-94460-9_13.

2 María Alpuente, Demis Ballis, Angel Cuenca-Ortega, Santiago Escobar, and José Meseguer. ACUOS2: A high-performance system for modular ACU generalization with subtyping and inheritance. In *16th European Conference on Logics in Artificial Intelligence, JELIA 2019*, LNCS. Springer, 2019. To appear.

3 María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014. doi:10.1016/j.ic.2014.01.006.

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

5 Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Generation Comp. Syst.*, 79:669–686, 2018. doi:10.1016/j.future.2017.07.024.

6 Alexander Baumgartner and Temur Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017. doi:10.1016/j.ic.2017.01.005.

7 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. A Variant of Higher-Order Anti-Unification. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.RTA.2013.113.

8 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-Order Pattern Anti-Unification in Linear Time. *J. Autom. Reasoning*, 58(2):293–310, 2017. doi:10.1007/s10817-016-9383-3.

9 Cao Feng and Stephen Muggleton. Towards Inductive Generalization in Higher Order Logic. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992*, pages 154–162. Morgan Kaufmann, 1992.

10 Kouichi Hirata, Takeshi Ogawa, and Masateru Harao. Generalization Algorithms for Second-Order Terms. In Rui Camacho, Ross D. King, and Ashwin Srinivasan, editors, *Inductive Logic Programming, 14th International Conference, ILP 2004, Porto, Portugal, September 6-8, 2004, Proceedings*, volume 3194 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2004. doi:10.1007/978-3-540-30109-7_14.

11 Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for Unranked Terms and Hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014. doi:10.1007/s10817-013-9285-6.

12 Tomer Libal and Dale Miller. Functions-as-Constructors Higher-Order Unification. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.26.

13 José Meseguer. Symbolic Reasoning Methods in Rewriting Logic and Maude. In Lawrence S. Moss, Ruy J. G. B. de Queiroz, and Maricarmen Martínez, editors, *WoLLIC 2018*, volume 10944 of *LNCS*, pages 25–60. Springer, 2018. doi:10.1007/978-3-662-57669-4_2.

14 Dale Miller. A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. *J. Log. Comput.*, 1(4):497–536, 1991. doi:10.1093/logcom/1.4.497.

15 Frank Pfenning. Unification and Anti-Unification in the Calculus of Constructions. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 74–85. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151632.

16 Brigitte Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Log.*, 11(1), 2009. doi:10.1145/1614431.1614437.

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

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

19 Reudismam Rolim, Gustavo Soares, Rohit Gheyi, and Loris D'Antoni. Learning Quick Fixes from Code Repositories. *CoRR*, abs/1803.03806, 2018. arXiv:1803.03806.

20 Reudismam Rolim de Sousa. *Learning syntactic program transformations from examples.* PhD thesis, Universidade Federal de Campina Grande, Brazil, 2018.

21 Martin Schmidt, Ulf Krumnack, Helmar Gust, and Kai-Uwe Kühnberger. Heuristic-Driven Theory Projection: An Overview. In Henri Prade and Gilles Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*, pages 163–194. Springer, 2014. doi:10.1007/978-3-642-54516-0_7.

A Examples

▶ **Example 40.** Let $t = \lambda x.f(g(x), g(g(x)))$ and $s = \lambda x.h(g(g(x)), g(x))$. Then

- $r_0 = \lambda x.X(f(g(x), g(g(x))), h(g(g(x)), g(x)))$ is a shallow top-maximal lgg of t and s.
- $r_1 = \lambda x.Y(g(x), g(g(x)))$ is a top-maximal rfc-lgg of t and s. We have $r_1 \prec r_0$. Note that $r_2 = \lambda x.Y(g(g(x)), g(x))$ is also a top-maximal rfc-generalization with $r_1 \simeq r_2$.
- $r_3 = \lambda x.Z(g(x))$ is a top-maximal fc-lgg of t and s. In this case we have $r_3 \simeq r_1$, because $r_3\{Z \mapsto \lambda x.Y(x, g(x))\} = r_1$ and $r_1\{Y \mapsto \lambda x, y.Z(x)\} = r_3$.
- $r_4 = \lambda x.X(x)$ is a top-maximal pattern-lgg of t and s and $r_4 \prec r_3$.

▶ **Example 41.** Let $t = \lambda x.f(x, x)$ and $s = \lambda x.f(g(g(x)), g(x))$. Then the non-shallow term $r = \lambda x.f(Y(Y(x)), Y(x))$ is a top-maximal generalization of t and s, and it is less general than their shallow top-maximal generalization $\lambda x.f(Z(x, g(g(x))), Z(x, g(x)))$.

▶ **Example 42.** Let $t = \lambda x, y.X(f(x), f(y))$ and $s = \lambda x, y.X(g(y), g(x))$. Then the term $r = \lambda x, y.X(Y(f(x), g(y)), Y(f(y), g(x)))$ is a $\mathsf{genvar}(r, t, s)$-shallow top-maximal lgg of t and s, but not a shallow top-maximal lgg.

▶ **Example 43.** Let $t = \lambda x.f(g(x), g(x), g(g(x)))$ and $s = \lambda y.h(g(g(y)), a, b)$. Then the sequence of inferences in $\mathfrak{G}_{\mathrm{cs}}$ is

$$\{X : \lambda x.f(g(x), g(x), g(g(x))) \triangleq \lambda y.h(g(g(y)), a, b)\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs}}$$
$$\{X'(x) : f(g(x), g(x), g(g(x))) \triangleq h(g(g(x)), a, b)\};\ \emptyset;\ \lambda x.X'(x) \Longrightarrow_{\mathsf{Sol\text{-}CS}}$$
$$\emptyset;\ \{Y(y_1, y_2) : f(y_1, y_1, y_2) \triangleq h(y_2, a, b)\};\ \lambda x.Y(g(x), g(g(x))).$$

In the Sol-CS step, we compute all position-maximal common subterms and their positions as in Example 23. Therefore, $f(y_1, y_1, y_2)$ and $h(y_2, a, b)$ are obtained from

$$f(g(x), g(x), g(g(x)))[1 \mapsto y_1][2 \mapsto y_1][3 \mapsto y_2] \text{ and}$$
$$h(g(g(x)), a, b)[1.1 \mapsto y_1][1.1 \mapsto y_1][1 \mapsto y_2],$$

respectively. To obtain t (resp., s) from the computed generalization $\lambda x.Y(g(x), g(g(x)))$, we need to apply the substitution $\{Y \mapsto \lambda y_1, y_2.f(y_1, y_1, y_2)\}$ (resp., $\{Y \mapsto \lambda y_1, y_2.h(y_2, a, b)\}$) to it. These substitutions can be directly read off the store.

▶ **Example 44.** Let $t = \lambda x_1.f(g_1(x_1, a), g_2(\lambda x_2.h(x_2)))$, $s = \lambda y_1.f(h_1(a, a), h_2(\lambda y_2.h(y_2)))$. Then we get the following derivation in $\mathfrak{G}_{\mathrm{cs}}$:

$$\{X : \lambda x_1.f(g_1(x_1, a), g_2(\lambda x_2.h(x_2))) \triangleq \lambda y_1.f(h_1(a, a), h_2(\lambda y_2.h(y_2)))\};\ \emptyset;\ X \Longrightarrow_{\mathsf{Abs}}$$
$$\{Y(x_1) : f(g_1(x_1, a), g_2(\lambda x_3.h(x_2))) \triangleq f(h_1(a, a), h_2(\lambda y_2.h(y_2)))\};\ \emptyset;\ \lambda x_1.Y(x_1) \Longrightarrow_{\mathsf{Dec}}$$
$$\{Y_1(x_1) : g_1(x_1, a) \triangleq h_1(a, a),\ Y_2(x_1) : g_2(\lambda x_2.h(x_2)) \triangleq h_2(\lambda y_2.h(y_2))\};\ \emptyset;$$
$$\lambda x_1.f(Y_1(x_1), Y_2(x_1))$$

Here Sol-CS rule applies. The set of position-maximal common subterms of $g_1(x_1, a)$ and $h_1(a, a)$ is $\{a\}$. We need to extend it by x_1, because x_1 has been bound before (as $Y_1(x_1)$ tells) and it appears in $g_1(x_1, a)$. Hence, after this extension we get the set $\{x_1, a\}$, which will be introduced in the generalization. The store also changes correspondingly:

$$\{Y_2(x_1) : g_2(\lambda x_2.h(x_2)) \triangleq h_2(\lambda y_2.h(y_2))\};\ \{Z_1(z_1, z_2) : g_1(z_1, z_2) \triangleq h_1(z_2, z_2)\};$$
$$\lambda x_1.f(Z_1(x_1, a), Y_2(x_1))$$

Also here, we use Sol-CS. The set of position-maximal common subterms of $g_2(\lambda x_2.h(x_2))$ and $h_2(\lambda y_2.h(y_2))$ is $\{\lambda x_2.h(x_2)\}$ (modulo α-equivalence). This set will not be extended by any bound variable, because the only candidate, x_1, appears neither in $g_2(\lambda x_2.h(x_2))$ nor in $h_2(\lambda y_2.h(y_2))$. Therefore, we get

$$\emptyset;\ \{Z_1(z_1, z_2) : g_1(z_1, z_2) \triangleq h_1(z_2, z_2),\ Z_2(z_3) : g_2(z_3) \triangleq h_2(z_3)\};$$
$$\lambda x_1.f(Z_1(x_1, a), Z_2(\lambda x_2.h(x_2))).$$

Note that if we had $\{\vec{x}\}$-extension instead of $\{\vec{x}\} \cap (\mathsf{fv}(t) \cup \mathsf{fv}(s))$-extension in the instance of **QR** above, then in the last step we would get the generalization $\lambda x_1.f(Z_1(x_1, a), Z_2(x_1, \lambda x_2.h(x_2)))$, which is more general than $\lambda x_1.f(Z_1(x_1, a), Z_2(\lambda x_2.h(x_2)))$, computed by $\mathfrak{G}_{\mathrm{cs}}$.

▶ **Example 45.** Let t and s be the terms, $t = \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x))))$, $s = \lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))$. Then $\mathfrak{G}_{\mathrm{rfc}}$ performs the following steps:

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

$$\begin{aligned}
&\{X'(x) : f(h_1(g(g(x)), a, b), h_2(g(g(x)))) \triangleq \\
&\qquad f(h_3(g(g(x)), g(x), a), h_4(g(g(x))))\}; \emptyset;\ \lambda x.X'(x) \Longrightarrow_{\mathsf{Dec}} \\
&\{Z_1(x) : h_1(g(g(x)), a, b) \triangleq h_3(g(g(x)), g(x), a), \\
&\qquad Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};\ \emptyset;\ \lambda x.f(Z_1(x), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}RFC}} \\
&\{Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\}; \\
&\qquad \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a)\};\ \lambda x.f(Y_1(g(x)), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}RFC}} \\
&\emptyset;\ \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),\ Y_2(y_2) : h_2(y_2) \triangleq h_4(y_2)\}; \\
&\qquad \lambda x.f(Y_1(g(x)), Y_2(g(g(x)))).
\end{aligned}$$

▶ **Example 46.** Let us see how fc-generalization can be computed for terms in Example 45. We can show the part of the computation that starts with Sol-FC:

$$\begin{aligned}
&\{Z_1(x) : h_1(g(g(x)), a, b) \triangleq h_3(g(g(x)), g(x), a), \\
&\qquad Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\};\ \emptyset;\ \lambda x.f(Z_1(x), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}FC}} \\
&\{Z_2(x) : h_2(g(g(x))) \triangleq h_4(g(g(x)))\}; \\
&\qquad \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a)\}; \lambda x.f(Y_1(g(x)), Z_2(x)) \Longrightarrow_{\mathsf{Sol\text{-}FC}} \\
&\emptyset;\ \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a),\ Y_2(y_2) : h_2(g(y_2)) \triangleq h_4(g(y_2))\}; \\
&\qquad \lambda x.f(Y_1(g(x)), Y_2(g(x))).
\end{aligned}$$