

# A Generic Framework for Higher-Order Generalizations\*

David M. Cerna and Temur Kutsia

RISC, Johannes Kepler University Linz

## 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.

## 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. The technique of computing generalizations is called anti-unification. It was introduced in 1970s [31, 32] and saw a renewed interest in recent years (see, e.g., [1–7, 11, 12, 14, 16, 18, 25]), mostly motivated by various applications (see, e.g., [8, 10, 17, 21, 22, 24, 27, 33]).

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 [13, 15, 29, 30], object terms [19], restricted terms [34], 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  $g(x)$ .

Combination of increased expressive power and good computational properties was the motivation behind the introduction of functions-as-constructors terms (fc-terms) in unification [26]. 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

---

\*This research is supported by the FWF project P28789-N32.

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 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), relaxed fc (Sect. 5.2.1), and fc (Sect. 5.2.2). We also show how pattern generalization fits into the schema (Sect. 5.2.3). 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.

## 2 Preliminaries

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

Other standard notions of the simply typed  $\lambda$ -calculus, like bound and free occurrences of variables, subterms,  $\alpha$ -conversion,  $\beta$ -reduction,  $\eta$ -long  $\beta$ -normal form, etc. are defined as usual (see, e.g., [9]).  $t \downarrow_\eta$  denotes the  $\eta$ -normal form of  $t$ . We denote the fact that  $t$  is a (strict) subterm of  $s$  using the infix binary symbol  $\sqsubseteq$  ( $\sqsubset$ ). We will use the convention of denoting bound variables by lowercase letters while free variables will be denoted by capital letters. The symbols  $\text{fv}(t)$  and  $\text{bv}(t)$  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.

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

When we write an equality between two  $\lambda$ -terms, we mean  $\alpha$ ,  $\beta$  and  $\eta$  equivalence.

*Positions* in  $\lambda$ -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  $\epsilon$  (the empty sequence), the occurrence of  $\lambda x.$  stands in the position 1, the bound occurrence of  $y$  in 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  $\text{ymb}(t, p)$  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  $\text{Pos}(t)$ .

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  $\sigma$  is more general than a substitution  $\vartheta$ , denoted  $\sigma \preceq \vartheta$ , if there exists a substitution  $\varphi$  such that  $\sigma \circ \varphi = \vartheta$ . The strict part of this relation is denoted by  $<$ . 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  $\sigma$  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  $\mathbf{B}$  be a set of variables and  $s$  be a term such that  $\mathbf{B} \cap \text{bv}(s) = \emptyset$ . Assume that distinct bound variables have distinct names in  $s$ . We say that a term  $t$  is  $\mathbf{B}$ -restricted in  $s$  if it is a subterm of  $s$  and (i)  $t \in \mathbf{B} \cup \text{bv}(s)$ , or (ii)  $t = (f t_1 \cdots t_n)$ , where  $n > 0$ ,  $f \in \mathbf{C} \cup \mathbf{B} \cup \text{bv}(s)$  and each  $t_i$ ,  $1 \leq i \leq n$ , is a  $\mathbf{B}$ -restricted term in  $s$ .

**Definition 2** (Relaxed functions-as-constructors triples). Let  $\mathbf{F}$  and  $\mathbf{B}$  be two disjoint sets of variables and  $s$  be a term such that  $\mathbf{F} \cap \text{bv}(s) = \mathbf{B} \cap \text{bv}(s) = \emptyset$ . It is also assumed that distinct bound variables have distinct names in  $s$ . We say that the triple  $(\mathbf{F}, \mathbf{B}, s)$  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  $X \in \mathbf{F}$ ,  $t_i \downarrow_\eta$  is a  $\mathbf{B}$ -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  $X \in \mathbf{F}$ , 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  $\mathbf{F}$ ,  $\mathbf{B}$ , and  $s$  be as in Definition 2 and  $(\mathbf{F}, \mathbf{B}, s)$  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  $X, Y \in \mathbf{F}$ , for each  $0 < i \leq n$ ,  $0 < j \leq m$ , we have  $t_i \downarrow_\eta \not\sqsubseteq s_j \downarrow_\eta$ .

**Definition 4** (Pattern triples). Let  $\mathbf{F}$ ,  $\mathbf{B}$ , and  $s$  be as in Definition 2 and  $(\mathbf{F}, \mathbf{B}, s)$  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  $X \in \mathbf{F}$ , we have  $t_i \downarrow_\eta \in \mathbf{B} \cup \text{bv}(s)$  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  $F \cap \text{bv}(t) = \emptyset$ . Then  $t$  is an  $F$ -*rfc-term* (resp.,  $F$ -*fc-term*,  $F$ -*pattern*), if  $(F, \emptyset, t)$  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  $X \in F$ , we have  $F \cap (\cup_{i=1}^n \text{fv}(t_i)) = \emptyset$ .

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

Note that pattern coincides with the well-known higher-order patterns [28] fragment. Every pattern is an fc-term. Every fc-term is an rfc-term. Every rfc-term is a shallow term.

**Example 1.** Consider the following terms:

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

$t_1$  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  $\eta$ -long  $\beta$ -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  $\sigma$  such that  $\text{dom}(\sigma) = \{X\}$  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 of  $X(\vec{x}) : t \triangleq s$  is *least general* (or *most specific*) if there is no anti-unifier  $\vartheta$  of the same problem that satisfies  $\sigma \prec \vartheta$ . Obviously, if  $\sigma$  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  $\text{genvar}(r, t, s) := \text{fv}(r) \setminus (\text{fv}(s) \cup \text{fv}(t))$ .

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 6** (Top-maximal generalization). Let  $s$  and  $t$  be terms of the same type in  $\eta$ -long  $\beta$ -normal form such that the bound variables are renamed uniformly: the  $i$ th 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  $\text{symb}(s, \epsilon) = \text{symb}(t, \epsilon)$ , then  $\text{symb}(r, \epsilon) = \text{symb}(s, \epsilon)$ .

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

**Example 2.** 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.

Here we consider a variant of higher-order anti-unification problem:

**Given:** Two terms  $t$  and  $s$  of the same type in  $\eta$ -long  $\beta$ -normal form.

**Find:** A top-maximal generalization  $r$  of  $t$  and  $s$  such that  $r$  is a  $\text{genvar}(r, t, s)$ -shallow term.

If  $t$  and  $s$  do not contain free variables, the problem reduces to computing shallow generalizations. By imposing additional conditions on  $r$ , we get more specialized problems such as, rfc, fc, pattern or other generalization problems.

The problem statement implies that we are looking for  $r$  which is least general among all shallow terms which are top-maximal 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  $\text{genvar}$ -shallow term. The same is true for other special forms.

**Example 3.** 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 4.** 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  $\text{genvar}(r, t, s)$ -shallow top-maximal lgg of  $t$  and  $s$ , but not a shallow top-maximal lgg.

**Example 5.** 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 shallow top-maximal lgg of  $t$  and  $s$ , where the arguments of genvars are respective common subterms 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  $r_4 \prec r_3$ .

**Example 6.** 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  $r_4 \prec r_3$ .

Top-maximality is an important requirement for existing an lgg. If we do not require it, we might have  $\preceq$ -incomparable generalizations. For instance, in Example 2,  $r_1$  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 2 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  $\{X_1(\vec{x}_1) : t_1 \triangleq s_1, \dots, X_n(\vec{x}_n) : t_n \triangleq s_n\}$  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  $\text{genvar}(r, t, s)$ -shallow term. We aim at computing lggs.

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

**Remark 1.** We assume that in the set  $A \cup S$  each occurrence of  $\lambda$  binds a distinct name variable and that each generalization variable occurs in  $A \cup S$  only once.

**Definition 1.** The set of transformations  $\mathfrak{G}$  is defined by the following rules:

**Dec: Decomposition**

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

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

**Abs: Abstraction**

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

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

**Sol: Solve**

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

where  $t$  and  $s$  are of a basic type,  $\text{head}(t) \neq \text{head}(s)$ ,  $q_1, \dots, q_n$  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  $Y, y_1, \dots, y_n$  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  $\pi : \{\vec{x}\} \rightarrow \{\vec{y}\}$  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  $\lambda z_1, \dots, z_n. C'_t$  and  $\lambda z_1, \dots, z_n. C'_s$ , where the bound variables  $z_1, \dots, z_n$  play the role of holes. In the store we keep the  $\eta$ -long  $\beta$ -normal form of  $(C_t y_1 \cdots y_n)$  and  $(C_s y_1 \cdots y_n)$ . When applied to  $q_1, \dots, q_n$ ,  $C_t$  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  $q_1, \dots, q_n$ . Moreover, choosing different special forms of  $q_1, \dots, q_n$ , 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  $\eta$ -long  $\beta$ -normal form, and  $X$  does not occur in  $t$  and  $s$ . Moreover, all genvars are distinct.

The property that each occurrence of  $\lambda$  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 1.** *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  $\text{genvar}(r, t, s)$ -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  $\text{genvar}(r, t, s)$ -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 [15].

**Sol:** We have  $\vartheta = \{X \mapsto \lambda \vec{x}. Y(q_1, \dots, q_n)\}$ , where  $q_1, \dots, q_n$  are distinct subterms in  $t$  or  $s$ .

Let  $\psi_1 = \{Y \mapsto \lambda y_1, \dots, y_n. (C_t y_1 \cdots y_n)\}$  and  $\psi_2 = \{Y \mapsto \lambda y_1, \dots, y_n. (C_s y_1 \cdots y_n)\}$ . 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  $X(\vec{x})\vartheta\psi_1 = X(\vec{x})\{X \mapsto \lambda \vec{x}. t, \dots\} = t$ ,  $X(\vec{x})\vartheta\psi_2 = X(\vec{x})\{X \mapsto \lambda \vec{x}. s, \dots\} = s$ , 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  $\vartheta_i^k$  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 [15].

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

**Corollary 1.** *If  $t$  and  $s$  do not contain free variables, then the result computed by  $\mathfrak{G}$  is a shallow top-maximal generalization of  $t$  and  $s$ .*

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  $\lambda \vec{x}. Y(q_1, \dots, q_n)$  and the store contains the AUT  $Y(y_1, \dots, y_n) : (C_t y_1 \cdots y_n) \triangleq (C_s y_1 \cdots y_n)$ , then the substitution  $\{Y \mapsto \lambda y_1, \dots, y_n. (C_t y_1 \cdots y_n)\}$  gives  $t$  and another substitution  $\{Y \mapsto \lambda y_1, \dots, y_n. (C_s y_1 \cdots y_n)\}$  gives  $s$ .

**Theorem 2** (Uniqueness modulo  $\simeq$ ). *Assume that the set  $\{q_1, \dots, q_n\}$ ,  $C_t$ , 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 [15] 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  $\{q_1, \dots, q_n\}$ ,  $C_t$ , and  $C_s$  are uniquely determined. Therefore, the theorem follows from Theorem 4 in [15].  $\square$

## 4.1 Solve Rule

The Solve rule is generic and leaves room for special versions of the algorithm depending on how the subterms  $q_1, \dots, q_n$  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 anti-unification problem for the terms  $\lambda x. f(g(x, a), g(a, x))$  and  $\lambda x. f(h(x, a), h(a, x))$ :

$$\begin{aligned} & \{\lambda x. f(g(x, a), g(a, x)) \triangleq \lambda x. f(h(x, a), h(a, x))\}; \emptyset; X \Longrightarrow_{\text{Abs, Dec}} \\ & \{X_1(x) : g(x, a) \triangleq h(x, a), X_2(x) : g(a, x) \triangleq h(a, x)\}; \emptyset; \lambda x. f(X_1(x), X_2(x)) \Longrightarrow_{\text{Sol}} \\ & \{X_2(x) : g(a, x) \triangleq h(a, x)\}; \{Y(y_1, y_2) : g(y_1, y_2) \triangleq h(y_1, y_2)\}; \lambda x. f(Y(x, a), X_2(x)) \Longrightarrow_{\text{Sol}} \\ & \emptyset; \{Y(y_1, y_2) : g(y_1, y_2) \triangleq h(y_1, y_2), Z(z_1, z_2) : g(a, z_1) \triangleq h(z_2, z_1)\}; \lambda x. f(Y(x, a), Z(x, a)). \end{aligned}$$

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 get

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

and, hence, would obtain the lgg.

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  $p_i \in \text{Pos}(t_i)$ ,  $i \in \{1, 2\}$ , then  $s = t_1|_{p_1} = t_2|_{p_2}$ ,
- if  $p_1 \in \text{Pos}(t_1)$  and  $p_2 = \bullet$ , then  $s = t_1|_{p_1}$  and  $s$  does not occur in  $t_2$ ,
- if  $p_1 = \bullet$  and  $p_2 \in \text{Pos}(t_2)$ , 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 7.** Given a set of variables  $\{\vec{x}\}$ , terms  $t_1$  and  $t_2$ , and a set of terms  $S$ , the **Select** function with the parametric condition  $\text{cond}$ ,  $\text{Select}_{\text{cond}}(\{\vec{x}\}, t_1, t_2, S)$ , 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.  $\text{cond}(\{\vec{x}\}, t_1, t_2, S, Q)$  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.

**Remark 2.** The set  $S$  is needed when **Select** takes into account global restrictions, such as the one for fc-terms. Otherwise, it will be the empty set.

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, and  $S$  be a set of subterms of AUTs in  $A$ . Let  $\text{Select}_{\text{cond}}(\{\vec{x}\}, t, s, S) = Q$ . The rule of choosing  $q_1, \dots, q_n$  in **Sol** is the following:

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

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:

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

**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 generalization*, since such generalizations give original terms by projection substitutions. The second class is called *common-subterm generalization*. There are several subcategories in that class, as we will see.

### 5.1 Projection-based generalization

This is the simplest case. If  $t$  and  $s$  are the subject of the **Sol** rule, we should keep both of them in the generalization. It means to specify **Select** and, consequently, **QR** as follows:

**Specifying**  $\text{Select}_{\text{cond}}(\{\vec{x}, t, s, S\})$ :  $S = \emptyset$ , **cond** is always true.

**The instance of QR:**  $q_1 = t, q_2 = s$ .

**QR** implies that  $C_t$  and  $C_s$  in **CR** have a special form:  $C_t = \lambda z_1, z_2. z_1, C_s = \lambda z_1, z_2. z_2$ .

After applying **Sol** with  $\text{Select}_{\text{cond}}$  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  $\mathfrak{G}_{\text{prb}}$ .

Generalizations that retain both terms whose heads are different are called imitation-free generalizations in [23] (where only second-order generalizations are considered), motivated from [19]. 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 3.**  $\mathfrak{G}_{\text{prb}}$  computes a projection-based  $\text{genvar}(r, t, s)$ -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 1, 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.  $\square$

**Theorem 4.** If  $r_0$  is a projection-based  $\text{genvar}(r_0, t, s)$ -shallow top-maximal generalization of  $t$  and  $s$ , then  $\mathfrak{G}_{\text{prb}}$  computes  $r$ , starting from  $t$  and  $s$ , such that  $r_0 \preceq r$ .

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

Interestingly, projection-based generalizations are least general among all top-maximal generalizations that do not nest  $\text{genvars}$ :

**Theorem 5.** Let  $r_1$  and  $r_2$  be respectively  $\text{genvar}(r_1, t, s)$ - and  $\text{genvar}(r_2, t, s)$ -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 position of  $\text{genvars}$  are common for the input terms  $t$  and  $s$ . Top-maximality requires that common symbols are not abstracted by  $\text{genvars}$ : they are retained in the generalization.

Let  $r_1$  and  $r_2$  contain a  $\text{genvar}$  in position  $p$ . Since they are top-maximal, all symbols above  $p$  are common in  $s$  and  $t$ . Since  $r_1$  is  $\text{genvar}(r_1, t, s)$ -shallow and projection-based,  $r_1|_p$  should have a form  $Y(t|_p, s|_p)$ , where  $Y$  is a  $\text{genvar}$ .

Also,  $r_2$  is  $\text{genvar}(r_2, t, s)$ -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  $\sigma_1$  and  $\sigma_2$  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 the shallow property,  $r_1$  and  $r_2$  have  $\text{genvars}$  in the same positions. The projection-based property implies that  $r_1$  contains only one  $\text{genvar}$ , which we denoted by  $Y$  above. Repeating the above reasoning for each  $\text{genvar}$  position finishes the proof.  $\square$

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

First, we define an extension of a set of terms by variables (bound in the context):

**Definition 8.** Let  $\mathbf{B}$  be a set of variables and  $S$  be a set of terms such that  $\text{bv}(S) \cap \mathbf{B} = \emptyset$ . By  $\mathbf{B}$ -extension of  $S$  we understand the set  $S \cup (\mathbf{B} \setminus \text{fv}(S))$ .

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

**Definition 9** (CS-generalization). A generalization  $r$  of two terms  $s$  and  $t$  is called their *common-subterms generalization*, shortly *cs-generalization*, if it is a  $\text{genvar}(r, s, t)$ -shallow top-maximal generalization of  $t$  and  $s$  satisfying the following condition:

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

In this definition, top-maximality guarantees that all positions  $q < p$  in  $r$  are also positions in  $t$  and  $s$  and  $\text{symb}(r, q) = \text{symb}(t, q) = \text{symb}(s, q)$  (modulo  $\alpha$ -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,  $r_1, \dots, r_n$  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  $B \cap (\text{fv}(t|_p) \cup \text{fv}(s|_p)) \subseteq B' \subseteq B$  in Definition 9.

In this definition 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 to refine it later into more specific categories such as rfc-, fc-, and pattern generalizations. 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 10.** Let  $t_1$  and  $t_2$  be terms such that for some positions  $p_1$  of  $t_1$  and  $p_2$  of  $t_2$ , we have  $t_1|_{p_1} = t_2|_{p_2}$ . Let this common subterm be  $s$ . We say that  $s$  is a  $(p_1, p_2)$ -*maximal common subterm* of  $t_1$  and  $t_2$  if (i)  $p_1 = \epsilon$  or  $p_2 = \epsilon$ , or (ii)  $p_1 = p'_1.i_1$  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  $t_2$ , denoted  $\text{pmcso}(s, t)$ , is defined as

$$\text{pmcso}(t_1, t_2) := \{(p_1, p_2, s) \mid s \text{ is a } (p_1, p_2)\text{-maximal common subterm of } t_1 \text{ and } t_2\}$$

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

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

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

**Example 7.** 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**  $\text{Select}_{\text{cond}}(\{\vec{x}, t, s, S\})$ :  $S = \emptyset$ ,  $\text{cond}(\{\vec{x}, t, s, S, Q\})$  is true if and only if  $Q = \text{pmcso}_{\{\vec{x}\}}(t, s)$ .

**The instance of QR:**  $\{q_1, \dots, q_n\}$  is the  $\{\vec{x}\} \cap (\text{fv}(t) \cup \text{fv}(s))$ -extension of the set of all position-maximal common subterms of  $t$  and  $s$ .

$\text{cond}$  and the item 2 of the definition of  $\text{Select}$  (Definition 7) imply that we have  $\{\vec{x}\} \cap (\text{fv}(t) \cup \text{fv}(s))$ -extension in the instance of **QR**. Without item 2, the instance of **QR** would be just  $\{\vec{x}\}$ -extension. Note that for computing cs-generalizations, it would be sufficient to take  $\{\vec{x}\}$ -extensions, but we aim at computing least general cs-generalizations, 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 the selection function 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, least general rfc-, fc-, and pattern-generalizations later as special cases of cs-generalizations (that's why the definition of cs-generalization is relaxed, as we mentioned above) and, on the other hand, to compute least general cs-generalizations by the specific version of **Sol**.

The specified instance of  $\text{cond}$  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 called **Sol-CS**. We denote the obtained algorithm by  $\mathfrak{G}_{\text{cs}}$ .

**Theorem 6.**  $\mathfrak{G}_{\text{cs}}$  computes a cs-generalization of two terms in cubic time.

*Proof.* Top-maximality and  $\text{genvar}$ -shallowness follow from Theorem 1. The cs-generalization property follows from the instance of **QR**. Selecting position-maximal common subterms from two terms can take quadratic time, and  $\mathfrak{G}_{\text{cs}}$  can perform this operation linearly many times. Hence the cubic time complexity. Merging at the end can not make it worse.  $\square$

**Theorem 7.** Let  $r_0$  be a cs-generalization of  $t$  and  $s$ . Then  $\mathfrak{G}_{\text{cs}}$  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  $\text{Select}$  function makes sure that  $\mathfrak{G}_{\text{cs}}$  puts in generalizations as many position-maximal common subterms as possible, and the exhaustive application of **Mer** makes all possible sharings of  $\text{genvars}$ . These arguments imply that  $r_0 \preceq r$ .  $\square$

**Example 8.** 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  $\mathfrak{G}_{\text{cs}}$  is

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

In the **Sol-CS** step, we compute all position-maximal common subterms and their positions as in Example 7. 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 9.** For  $t = \lambda x_1.f(g_1(x_1, a), g_2(\lambda x_2.h(x_2)))$  and  $s = \lambda y_1.f(h_1(a, a), h_2(\lambda y_2.h(y_2)))$  we get the following derivation in  $\mathfrak{G}_{\text{cs}}$ :

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

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:

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

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  $\alpha$ -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

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

Note that if we had  $\{\vec{x}\}$ -extension instead of  $\{\vec{x}\} \cap (\text{fv}(t) \cup \text{fv}(s))$ -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  $\mathfrak{G}_{\text{cs}}$ .

**Example 10.** let  $t = \lambda x.f(g(x), h(x, a))$  and  $s = \lambda y.h(g(y), a)$ . Then  $\mathfrak{G}_{\text{cs}}$  performs the following steps:

$$\begin{aligned} & \{X : \lambda x.f(g(x), h(x, a)) \triangleq \lambda y.h(g(y), a)\}; \emptyset; X \Longrightarrow_{\text{Abs}} \\ & \{X'(x) : f(g(x), h(x, a)) \triangleq h(g(x), a)\}; \emptyset; \lambda x.X'(x) \Longrightarrow_{\text{Sol-CS}} \\ & \emptyset; \{Y(y_1, y_2, y_3) : f(y_2, h(y_1, y_3)) \triangleq h(y_2, y_3)\}; \lambda x.Y(x, g(x), a). \end{aligned}$$

**Example 11.** let  $t = \lambda x.f(a)$  and  $s = \lambda y.g(b)$ . Then  $\mathfrak{G}_{\text{cs}}$  performs the following steps:

$$\begin{aligned} & \{X : \lambda x.f(a) \triangleq \lambda y.g(b)\}; \emptyset; X \Longrightarrow_{\text{Abs}} \{X'(x) : f(a) \triangleq g(b)\}; \emptyset; \lambda x.X'(x) \Longrightarrow_{\text{Sol-CS}} \\ & \emptyset; \{Y : f(a) \triangleq g(b)\}; \lambda x.Y. \end{aligned}$$

In some applications, it is desirable that the arguments of free variables are not subterms of each other. This requirement leads to generalization in (relaxed) fc- and patterns. These special cases also rely on position-maximal common subterm computation, but the obtained set is filtered, unlike the previous case. For those variants, in the sections below, we assume that the input terms are closed (do not contain free variables). When free variables are allowed in the input, we will need to add some extra tests to make sure that those variables 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 refer only to the latter. Note that by definition, rfc-, fc-, and pattern generalizations are shallow generalizations, since they do not allow nested `genvars`. We will refer to the following definition of rfc-, fc-, and pattern-generalizations:

**Definition 11.** A term  $r$ , which is a generalization of  $s$  and  $t$ , is their rfc-generalization if  $r$  is an rfc-term. A least general rfc-generalization (rfc-lgg) of  $s$  and  $t$  is their rfc-generalization  $r$  such that no rfc-generalization  $r'$  of  $s$  and  $t$  satisfies  $r \prec r'$ . fc- and pattern generalizations and lggs are defined in the same way.

### 5.2.1 RFC-generalization

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

*Proof.* Let  $X(r_1, \dots, r_n) = r|_p$  be a subterm of  $r$  at position  $p$ , where  $X$  is a `genvar`. Since  $r$  is an rfc-term,  $(\{X\}, \mathbf{B}, X(r_1, \dots, r_n))$  is an rfc-triple, where  $\mathbf{B}$  is the set of variables bound by  $\lambda$  above the position  $p$ . The terms  $r_1, \dots, r_n$  should contain all variables from  $\mathbf{B}$  which appear as free variables in  $s|_p$  or in  $t|_p$ , 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  $t$ .  $\square$

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 9, the subterms of `genvars` should also satisfy the corresponding argument and local restrictions.

It leads to the following 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 `Selectcond`( $\{\vec{x}, t, s, S\}$ ):**  $S = \emptyset$ , `cond`( $\{\vec{x}, t, s, S, Q\}$ ) is true iff  $Q$  is obtained from  $\text{pmcso}_{\{\vec{x}\}}(t, s)$  by removing from it

- (a) all triples  $(p_1, p_2, q)$  where  $q \downarrow_\eta$  is not  $\{\vec{x}\}$ -restricted in  $t \triangleq s$  (looking at  $t \triangleq s$  as a term) and
- (b) all triples  $(p_1^i, p_2^i, q_i)$  for which there exists  $(p_1^j, p_2^j, q_j) \in pmcso_{\{\vec{x}\}}(t, s)$  s.t.  $q_j \downarrow_\eta \sqsubset q_i \downarrow_\eta$ .

This selection function, by Definition 7, leads to the following instance of **QR**:

**The instance of QR:**  $\{q_1, \dots, q_n\}$  is the largest set of position-maximal common subterms of  $t$  and  $s$  whose  $\eta$ -normal forms are  $\{\vec{x}\} \cap (\text{fv}(t) \cap \text{fv}(s))$ -restricted in  $t$  or in  $s$ , and none of those  $\eta$ -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  $\mathfrak{G}_{\text{rfc}}$ . We get the theorem:

**Theorem 9.**  $\mathfrak{G}_{\text{rfc}}$  computes a top-maximal  $\{\text{cs}, \text{rfc}\}$ -generalization in cubic time.

*Proof.* From Theorem 1 we get top-maximality and shallowness (since the input is assumed to be closed). The  $\{\text{cs}, \text{rfc}\}$ -property of the generalization follows from the instance of **QR**. The cubic time of computing cs-generalizations dominates the time needed to filter out subterms which violate the rfc-property (since the check for argument and local restrictions requires quadratic time).  $\square$

**Theorem 10.** Let  $r_0$  be a top-maximal rfc-generalization of  $t$  and  $s$ . Then  $\mathfrak{G}_{\text{rfc}}$  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.  $\square$

**Example 12.** Let  $t = \lambda x.f(h_1(g(g(x)), a, b), h_2(g(g(x))))$  and  $s = \lambda y.f(h_3(g(g(y)), g(y), a), h_4(g(g(y))))$ . Then  $\mathfrak{G}_{\text{rfc}}$  performs the following steps:

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

**Example 13.** 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  $\mathfrak{G}_{\text{rfc}}$  stops with the final state  $\emptyset; \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a), Y_2(y_2) : h_2(y_2) \triangleq h_4(y_2)\}; \lambda x.f(Y_1(g(x)), Y_2(g(g(x))))$ .

### 5.2.2 FC-generalization

First, note that fc-generalizations are also rfc-generalizations and, hence, the properties of rfc-generalizations are valid for fc-generalizations as well. The counterpart of Theorem 8 holds. Analogously to the rfc case, here we aim at computing  $\{\text{cs}, \text{fc}\}$ -generalizations.

The peculiarity of computing fc-generalizations is that one needs 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  $A$  be the set  $\{X_1(\vec{x}_1) : t_1 \triangleq s_1, \dots, X_k(\vec{x}_k) : t_k \triangleq s_k\}$ . Let  $M_i$ ,  $1 \leq i \leq m$ , be the set of all position-maximal common subterms of  $t_i$  and  $s_i$ , and  $M = \cup_{i=1}^k M_i$ . Then we formulate the instance of **Select**, in which **cond**, takes into account  $M$ , filters out terms violating argument, local, and global restrictions:

**Specifying**  $\text{Select}_{\text{cond}}(\{\vec{x}, t, s, S\})$ :  $S = M$ ,  $\text{cond}(\{\vec{x}, t, s, S, Q\})$  is true iff  $Q$  is obtained from  $\text{pmcso}_{\{\vec{x}\}}(t, s)$  by

- (a) removing all triples  $(p_1, p_2, q)$  from  $\text{pmcso}_{\{\vec{x}\}}(t, s)$  where  $q \downarrow_\eta$  is not  $\{\vec{x}\}$ -restricted in  $t \triangleq s$  (looking at  $t \triangleq s$  as a term) and
- (b) replacing all  $(p_1^i, p_2^i, q_i) \in \text{pmcso}_{\{\vec{x}\}}(t, s)$  by  $(p_1^j, p_2^j, q_j)$ , where  $q_j \downarrow_\eta \sqsubset q_i \downarrow_\eta$  and  $q_j \in S$ .

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 7, 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  $\eta$ -normal forms are  $\{\vec{x}\} \cap (\text{fv}(t) \cap \text{fv}(s))$ -restricted in  $t$  or in  $s$ . Then  $\{q_1, \dots, q_n\}$  is obtained from  $Q$  by replacing all  $q_i \in Q$  by  $q_j \in S$ , if  $q_j \downarrow_\eta \sqsubset q_i \downarrow_\eta$ .

Since we take into account the whole of  $S$  when deciding which subterms to keep under the genvars, the global restriction of fc-terms is satisfied. The obtained instance of **Sol** is denoted by **Sol-FC**, and the algorithm by  $\mathfrak{G}_{\text{fc}}$ . The theorems below can be proved similarly to their rfc-counterparts:

**Theorem 11.**  $\mathfrak{G}_{\text{fc}}$  computes a top-maximal  $\{\text{cs}, \text{fc}\}$ -generalization in cubic time.

**Theorem 12.** Let  $r_0$  be a top-maximal fc-generalization of  $t$  and  $s$ . Then  $\mathfrak{G}_{\text{fc}}$  computes a generalization  $r$  of  $t$  and  $s$  such that  $r_0 \preceq r$ .

**Example 14.** Let us see how fc-generalization can be computed for terms in Example 12. We can show the part of the computation that starts with **Sol-FC**:

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

**Example 15.** For terms in Example 13, the algorithm  $\mathfrak{G}_{\text{fc}}$  stops with the final state  $\emptyset; \{Y_1(y_1) : h_1(g(y_1), a, b) \triangleq h_3(g(y_1), y_1, a), Y_2(y_2) : h_2(g(y_2)) \triangleq h_4(g(y_2))\}; \lambda x.f(Y_1(g(x)), Y_2(g(x)))$ .

### 5.2.3 Pattern generalization

Similarly to rfc- and fc-generalizations, pattern generalizations are shallow but not necessarily top-maximal. They are not necessarily cs-generalizations either. For instance, the term  $\lambda x, y. f(X(x, y))$  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 13.** *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 [13,15]. The rest of the proof is similar to the proof of Theorem 8.  $\square$

**Specifying**  $\text{Select}_{\text{cond}}(\{\vec{x}, t, s, S\})$ :  $S = \emptyset$ ,  $\text{cond}(\{\vec{x}, t, s, S, Q\})$  is true iff  $Q$  is obtained from  $\text{pmcso}_{\{\vec{x}\}}(t, s)$  by

- (a) removing all triples  $(p_1, p_2, q)$  from  $\text{pmcso}_{\{\vec{x}\}}(t, s)$  where  $q \downarrow_\eta$  is not  $\{\vec{x}\}$ -restricted in  $t \triangleq s$  (looking at  $t \triangleq s$  as a term) and
- (b) replacing all  $(p_1^i, p_2^i, q_i) \in \text{pmcso}_{\{\vec{x}\}}(t, s)$  by  $(p_1^j, p_2^j, x)$ , where  $x \downarrow_\eta \sqsubset q_i \downarrow_\eta$  and  $x \in \{\vec{x}\}$ .

We wrote the selection function in this form to show its relation to the selection functions of the other common subterms based generalizations (cs, rfc, and fc). It leads to the following simple instance of **QR**:

**The instance of QR:**  $\{q_1, \dots, q_n\} = \{\vec{x}\} \cap (\text{fv}(t) \cup \text{fv}(s))$ .

**The instance of CR:**  $C_t = t$ ,  $C_s = s$ , and the  $y_i = q_i$ .

The obtained instance of Sol is denoted by Sol-P. What we get, is the pattern generalization algorithm described in [15], for the closed input. We denote it by  $\mathfrak{G}_{\text{pat}}$ .

**Theorem 14.**  $\mathfrak{G}_{\text{pat}}$  computes a least general pattern generalization in linear time.

*Proof.* See [15].  $\square$

## 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] H. Ait-Kaci and G. Pasi. Fuzzy unification and generalization of first-order terms over similar signatures. In F. Fioravanti and J. P. Gallagher, editors, *LOPSTR 2017*, volume 10855 of *LNCS*, pages 218–234. Springer, 2017.

- [2] H. Aït-Kaci and G. Pasi. Lattice operations on terms with fuzzy signatures. *CoRR*, abs/1709.00964, 2017.
- [3] M. Alpuente, D. Ballis, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. ACUOS2: A high-performance system for modular ACU generalization with subtyping and inheritance. In *JELIA 2019*, LNCS. Springer, 2019.
- [4] M. Alpuente, S. Escobar, J. Espert, and J. Meseguer. ACUOS: A system for modular ACU generalization with subtyping and inheritance. In Fermé and Leite [20], pages 573–581.
- [5] M. Alpuente, S. Escobar, J. Espert, and J. Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014.
- [6] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. A modular equational generalization algorithm. In M. Hanus, editor, *Logic-Based Program Synthesis and Transformation, 18th International Symposium, LOPSTR 2008, Valencia, Spain, July 17-18, 2008, Revised Selected Papers*, volume 5438 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2008.
- [7] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. Order-sorted generalization. *Electr. Notes Theor. Comput. Sci.*, 246:27–38, 2009.
- [8] N. Amiridze and T. Kutsia. Anti-unification and natural language processing. EasyChair Preprint no. 203.
- [9] H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.
- [10] A. D. Barwell, C. Brown, and K. Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Generation Comp. Syst.*, 79:669–686, 2018.
- [11] A. Baumgartner and T. Kutsia. A library of anti-unification algorithms. In Fermé and Leite [20], pages 543–557.
- [12] A. Baumgartner and T. Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017.
- [13] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. A variant of higher-order anti-unification. In F. van Raamsdonk, editor, *RTA 2013*, volume 21 of *LIPICs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
- [14] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Nominal anti-unification. In M. Fernández, editor, *26th International Conference on Rewriting Techniques and Applications, RTA 2015, June 29 to July 1, 2015, Warsaw, Poland*, volume 36 of *LIPICs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
- [15] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reasoning*, 58(2):293–310, 2017.

- [16] D. M. Cerna and T. Kutsia. Higher-order equational pattern anti-unification. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPICs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- [17] R. R. de Sousa. *Learning syntactic program transformations from examples*. PhD thesis, Universidade Federal de Campina Grande, Brazil, 2018.
- [18] S. Escobar. Unification and anti-unification modulo equational theories. In K. Korovin and B. Morawska, editors, *27th International Workshop on Unification, UNIF 2013, Eindhoven, Netherlands, June 26, 2013*, volume 19 of *EPiC Series in Computing*, page 1. EasyChair, 2013.
- [19] C. Feng and S. Muggleton. Towards inductive generalization in higher order logic. In D. H. Sleeman and P. Edwards, editors, *ML 1992*, pages 154–162. Morgan Kaufmann, 1992.
- [20] E. Fermé and J. Leite, editors. *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*. Springer, 2014.
- [21] B. A. Galitsky. Generalization of parse trees for iterative taxonomy learning. *Inf. Sci.*, 329:125–143, 2016.
- [22] B. A. Galitsky. Improving relevance in a content pipeline via syntactic generalization. *Eng. Appl. of AI*, 58:1–26, 2017.
- [23] K. Hirata, T. Ogawa, and M. Harao. Generalization algorithms for second-order terms. In R. Camacho, R. D. King, and A. Srinivasan, editors, *ILP 2004*, volume 3194 of *LNCS*, pages 147–163. Springer, 2004.
- [24] B. Konev and T. Kutsia. Anti-unification of concepts in description logic EL. In C. Baral, J. P. Delgrande, and F. Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 227–236. AAAI Press, 2016.
- [25] T. Kutsia, J. Levy, and M. Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014.
- [26] T. Libal and D. Miller. Functions-as-constructors higher-order unification. In D. Kesner and B. Pientka, editors, *FSCD 2016*, volume 52 of *LIPICs*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- [27] J. Meseguer. Symbolic reasoning methods in Rewriting Logic and Maude. In L. S. Moss, R. J. G. B. de Queiroz, and M. Martínez, editors, *WoLLIC 2018*, volume 10944 of *LNCS*, pages 25–60. Springer, 2018.
- [28] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.
- [29] F. Pfenning. Unification and anti-unification in the calculus of constructions. In *LICS*, pages 74–85. IEEE Computer Society, 1991.

- [30] B. Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Log.*, 11(1), 2009.
- [31] G. D. Plotkin. A note on inductive generalization. *Machine Intell.*, 5(1):153–163, 1970.
- [32] J. C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intell.*, 5(1):135–151, 1970.
- [33] R. Rolim, G. Soares, R. Gheyi, and L. D’Antoni. Learning quick fixes from code repositories. *CoRR*, abs/1803.03806, 2018.
- [34] M. Schmidt, U. Krumnack, H. Gust, and K. Kühnberger. Heuristic-driven theory projection: An overview. In H. Prade and G. Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*, pages 163–194. Springer, 2014.