Under consideration for publication in Math. Struct. in Comp. Science

Higher-Order Pattern Generalization Modulo Equational Theories

DAVID M. CERNA[1] and TEMUR KUTSIA[2]

[1,2]*RISC, Johannes Kepler University, Linz, Austria*

[1]*FMV, Johannes Kepler University, Linz, Austria*

[1]*david.cerna@risc.jku.at*

[2]*temur.kutsia@risc.jku.at*

Received

We consider anti-unification for simply typed lambda terms in theories defined by associativity, commutativity, identity (unit element) axioms and their combinations, and develop a sound and complete algorithm which takes two lambda terms and computes their equational generalizations in the form of higher-order patterns. The problem is finitary: the minimal complete set of such generalizations contains finitely many elements. We define the notion of optimal solution and investigate special restrictions of the problem for which the optimal solution can be computed in linear or polynomial time.

1. Introduction

Anti-unification algorithms aim at computing generalizations for given terms. A generalization of t and s is a term r such that t and s are substitution instances of r. Interesting generalizations are those that are least general (lggs). However, it is not always possible to have a unique least general generalization. In these cases the task is either to compute a minimal complete set of generalizations, or to impose restrictions so that uniqueness is guaranteed.

Anti-unification, as considered in this paper, uses both of these ideas. The theory is simply-typed lambda calculus, where some function symbols may be associative, commutative, have an associated unit element, or have any combination of these equational properties. Anti-unification for first-order terms containing function symbols obeying these properties is finitary, and the corresponding modular generalizations algorithms have been proposed in [Alpuente et al., 2014], also in the presence of ordered sorts. Anti-unification for simply typed lambda terms can be restricted to compute generalizations in the form of Miller's patterns [Miller, 1991], which makes it unitary, and the single least general generalization can be computed in linear time by the algorithm proposed in [Baumgartner et al., 2017]. These two approaches combine nicely with each other when one wants to develop a higher-order equational anti-unification algorithm. In this paper we present higher-order pattern anti-unification for terms containing function symbols whose equational axioms may include associativity, commutativity, identity (unit element) and

their combinations. Basically, we extend the syntactic[†] generalization rules from [Baumgartner et al., 2017] by equational decomposition rules inspired by those from [Alpuente et al., 2014]. The existence of the unit element introduces some complications due to the fact that the corresponding equational classes are infinite. To avoid them and still have a complete algorithm, we concentrate on linear generalizations (i.e., each generalization variable appears at most once) when a function symbol has the unit element. At the end, we get a modular algorithm in which different equational axioms for different function symbols can be combined automatically. The algorithm takes a pair of simply typed lambda terms (hence, the input is not restricted to patterns) and returns a set of their generalizations in the form of higher-order patterns. It is terminating, sound, and complete. However, the number of nondeterministic choices when decomposing may result in a large search tree. Although each branch can be developed in linear time, there can be too many of them to search efficiently.

This is the problem that we address in the second part of the paper. The idea is to use a greedy approach: introduce an optimality criterion, use it to select an anti-unification problem among different alternatives obtained by a decomposition rule, and try to solve only that. In this way, we would only compute one generalization. Checking the criterion and selecting the right branch should be done "reasonably fast". To implement this idea, we introduce conditions on the form of anti-unification problems which are guaranteed to compute "optimal" solutions, and study the corresponding complexities. In particular, we identify conditions for which A- (Associative), C- (Commutative), U- (with Unit) generalizations and their combinations can be computed in linear time. We also study how the complexity changes by relaxing these conditions.

Higher-order anti-unification has been investigated by various authors from different application perspectives. Research has been focused mainly on the investigation of special classes for which the uniqueness of lgg is guaranteed. A generic framework of higher-order anti-unification for such classes has been proposed recently in [Cerna and Kutsia, 2019a]. Some application areas include proof generalization [Pfenning, 1991], higher-order term indexing [Pientka, 2009], cognitive modeling and analogical reasoning [Besold et al., 2017, Schmidt et al., 2014], recursion scheme detection in functional programs [Barwell et al., 2018], inductive synthesis of recursive functions [Schmid, 2003], learning fixes from software code repositories [Rolim et al., 2018], just to name a few. Two higher-order anti-unification algorithms [Baumgartner and Kutsia, 2017, Baumgartner et al., 2017] are included in an online open-source anti- unification library [Baumgartner, 2015, Baumgartner and Kutsia, 2014]. First-order order-sorted equational generalization algorithms from [Alpuente et al., 2014] have also been implemented and are available online [Alpuente et al., 2019]. This related work does not consider anti-unification with higher-order terms in the presence of equational axioms. However, such a combination can be useful, for instance, for developing indexing techniques for higher-order theorem provers [Libal and Steen, 2016], in higher order program manipulation tools, proof transformation [Hetzl et al., 2014, Ebner et al., 2019], inductive theorem proving [Eberhard et al., 2017, Eberhard and Hetzl, 2015].

This paper is an extended and improved version of [Cerna and Kutsia, 2018]. It is organized as follows: In Section 2 we introduce the main notions and define the problem. In Section 3 we recall the higher-order anti-unification algorithm from [Baumgartner et al., 2017]. In Section 4 we

[†] We refer to the higher-order anti-unification algorithm from [Baumgartner et al., 2017] as syntactic, although it works modulo $\beta\eta$-conversion.

extend the algorithm with equational decomposition rules for associativity, commutativity, and their combination. Section 5 is devoted to theories with unit elements. In Section 6 we introduce computationally well-behaved restrictions of anti-unification problems. The next sections describe the behavior of equational anti-unification algorithms on these restrictions: In Section 7 we discuss A- and AU- generalization and speak about optimality. Section 8 is about C- and CU-generalization. Section 9 is about AC- and ACU-generalization. Section 10 summarizes the results.

2. Preliminaries

This work builds upon the formulations and results of [Baumgartner et al., 2013, 2017]. Higher-order signatures are composed of *types* constructed from a set of *base types* (typically δ) using the grammar $\tau ::= \delta \mid \tau \to \tau$. We will consider $\to$ to be associative right unless otherwise stated. *Variables* (typically $X, Y, Z, x, y, z, \ldots$) as well as *constants* (typically $f, a, b, c, \ldots$) are assigned types from the set of types constructed using the above grammar. By the symbol h we denote a constant or a variable.

λ-*terms* (typically $t, s, u, \ldots$) are constructed 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 using the type construction mentioned above. Terms of the form $(\ldots(\mathsf{h}\ t_1)\ldots t_m)$ will be written as $\mathsf{h}(t_1, \ldots, t_m)$, and terms of the form $\lambda x_1.\ldots.\lambda x_n.t$ as $\lambda x_1, \ldots, x_n.t$. We use $\vec{x}$ as a short-hand for $x_1, \ldots, x_n$. When necessary, we write a λ-term t together with its type α as $t : \alpha$.

Every constant c will have an associated set of axioms, denoted by $Ax(c)$. If $Ax(c)$ is empty, then c does not have any associated property and is called *free*. Otherwise, $Ax(f) \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$ where A is associativity, i.e. $f(t_1, f(t_2, t_3)) \equiv f(f(t_1, t_2), t_3)$, C is commutativity, i.e. $f(t_1, t_2) \equiv f(t_2, t_1)$, and U is unit element, i.e. $f(t, \epsilon_f) \equiv f(\epsilon_f, t) \equiv t$, where ϵ_f is the unique unit element associated with the function constant f. Note that only function constants of the type $\alpha \to \alpha \to \alpha$ are allowed to have equational properties.

We assume that terms are written in *flattened form*, obtained by replacing all subterms of the form $f(t_1, \ldots, f(s_1, \ldots, s_n), \ldots t_n)$ by $f(t_1, \ldots, s_1, \ldots, s_n, \ldots t_n)$, where $\mathsf{A} \in Ax(f)$. Also, by convention, the term $f(t)$ stands for t, if $\mathsf{A} \in Ax(f)$. Other standard notions of the simply typed λ-calculus, like bound and free occurrences of variables, α-conversion, β-reduction, η-long β-normal form, etc. are defined as usual (see [Barendregt, 1984, Dowek, 2001]). By default, terms are assumed to be written in η-long β-normal form. Therefore, all terms have the form $\lambda x_1, \ldots, x_n.\mathsf{h}(t_1, \ldots, t_m)$, where $n, m \geqslant 0$, $t_1, \ldots, t_m$ have this form, and the term $\mathsf{h}(t_1, \ldots, t_m)$ has a base type.

The set of free variables of a term t is denoted by $\mathrm{Vars}(t)$. When we write an equality between two λ-terms, we mean that they are equivalent modulo α, β and η equivalence.

The *size* of a term t, denoted $|t|$, is defined recursively as $|\mathsf{h}(t_1, \ldots, t_n)| = 1 + \sum_{i=1}^{n} |t_i|$ and $|\lambda x.t| = 1 + |t|$. The *depth* of a term t, denoted $depth(t)$, is defined recursively as $depth(\mathsf{h}(t_1, \ldots, t_n)\ ) = 1 + \max_{i \in \{1,\ldots,n\}} depth(t_i)$ and $depth(\lambda x.t) = 1 + depth(t)$. For a term $t = \lambda x_1, \ldots, x_n.\mathsf{h}(t_1, \ldots, t_m)$ with $n, m \geqslant 0$, its *head* is defined as $head(t) = \mathsf{h}$.

A *higher-order pattern* is a λ-term where, when written in η-long β-normal form, all free variable occurrences are applied to lists of pairwise distinct (η-long forms of) bound variables. For instance, $\lambda x.f(X(x), Y)$, $f(c, \lambda x.x)$ and $\lambda x.\lambda y.X(\lambda z.x(z), y)$ are patterns, while $\lambda x.f(X(X(x)), Y)$, $f(X(c), c)$ and $\lambda x.\lambda y.X(x, x)$ are not.

Substitutions are finite sets of pairs $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where X_i and t_i have the same type and the X's are pairwise distinct variables.They represent functions that map each X_i to t_i and any other variable to itself. They can be extended to type preserving functions from terms to terms as usual, avoiding variable capture. The notions of substitution *domain* and *range* are also standard and are denoted, respectively, by Dom and Ran. Substitutions are denoted by lower case Greek letters, while the identity substitution is denoted by Id.

We use postfix notation for substitution applications, writing $t\sigma$ instead of $\sigma(t)$. As usual, the application $t\sigma$ affects only the free occurrences of variables from $\mathrm{Dom}(\sigma)$ in t. We write $\vec{x}\sigma$ for $x_1\sigma, \ldots, x_n\sigma$, if $\vec{x} = x_1, \ldots, x_n$. Similarly, for a set of terms S, we define $S\sigma = \{t\sigma \mid t \in S\}$. The *composition* of σ and ϑ is written as juxtaposition $\sigma\vartheta$ and is defined as $x(\sigma\vartheta) = (x\sigma)\vartheta$ for all x. Another standard operation, *restriction* of a substitution σ to a set of variables S, is denoted by $\sigma|_S$.

A substitution σ_1 is *more general* than σ_2, written $\sigma_1 \preceq \sigma_2$, if there exists ϑ such that $X\sigma_1\vartheta = X\sigma_2$ for all $X \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)$. The strict part of this relation is denoted by $\prec$. The relation $\preceq$ is a partial order and generates the equivalence relation which we sometimes call equigenerality and denote by $\simeq$. We overload $\preceq$ by defining $s \preceq t$ if there exists a substitution σ such that $s\sigma = t$. The focus of this work is generalization in the presence of equational axioms thus we need a more general concept of ordering of substitutions/terms by their generality. We say that two terms s, t are $s =_{\mathcal{E}} t$ if they are equivalent modulo $\mathcal{E} \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$. For example, $f(a, f(b, c)) \neq f(f(a, b), c)$ but, $f(a, f(b, c)) =_{\{A\}} f(f(a, b), c)$. Under this notion of equality we can say that a substitution σ_1 is *more general than σ_2 modulo an equational theory* $\mathcal{E} \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$, written $\sigma_1 \preceq_{\mathcal{E}} \sigma_2$, if there exists ϑ such that $X\sigma_1\vartheta =_{\mathcal{E}} X\sigma_2$ for all $X \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)$. Note that $\prec$ and $\simeq$ and their term extension are generalized accordingly. From this point on we will use the ordering relation modulo an equational theory when discussing generalization.

A term t is called a *generalization* or an *anti-instance modulo an equational theory* $\mathcal{E}$ of two terms t_1 and t_2 if $t \preceq_{\mathcal{E}} t_1$ and $t \preceq_{\mathcal{E}} t_2$. (Will refer to such objects as $\mathcal{E}$-generalizations.) It is a *higher-order pattern generalization* if additionally t is a higher-order pattern. It is the *least general generalization* (lgg in short), aka 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 $t \prec_{\mathcal{E}} s$.

An *anti-unification problem* (shortly AUP) is a triple $X(\vec{x}) : t \triangleq s$ where

— $\lambda\vec{x}.X(\vec{x})$, $\lambda\vec{x}.t$, and $\lambda\vec{x}.s$ are terms of the same type,
— $\lambda\vec{x}.t$ and $\lambda\vec{x}.s$ are in η-long β-normal form, and
— X does not occur in t and s.

The variable X is called a *generalization variable*. The term $X(\vec{x})$ is called the *generalization term.* The variables that belong to $\vec{x}$, as well as bound variables, are written in the lower case letters $x, y, z, \ldots$. Originally free variables, including the generalization variables, are written with the capital letters $X, Y, Z, \ldots$. This notation intuitively corresponds to the usual convention about syntactically distinguishing bound and free variables.

The size of a set of AUPs is defined as $|\{X_1(\vec{x_1}) : t_1 \triangleq s_1, \ldots, X_n(\vec{x_n}) : t_n \triangleq s_n\}| = \sum_{i=1}^{n} |t_i| + |s_i|$. Notice that the size of $X_i(\vec{x_i})$ is not considered.

An *anti-unifier* of an AUP $X(\vec{x}) : t \triangleq s$ is a substitution σ such that $\mathrm{Dom}(\sigma) = \{X\}$ 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*) modulo an equational theory $\mathcal{E}$ if there is no anti-unifier ϑ of

the same problem that satisfies $\sigma \prec_{\mathcal{E}} \vartheta$. Obviously, if σ is a least general anti-unifier of an AUP $X(\vec{x}) : t \triangleq s$, then $\lambda\vec{x}.X(\vec{x})\sigma$ is a lgg of $\lambda\vec{x}.t$ and $\lambda\vec{x}.s$.

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

Given: Terms t and s of the same type in η-long β-normal form and an equational theory $\mathcal{E} \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$.

Find: A higher-order pattern generalization r of t and s modulo $\mathcal{E} \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$.

Essentially, we are looking for r which is least general among all higher-order patterns which generalize t and s (modulo $\mathcal{E}$). There can still exist a term which is less general than r, generalizes both s and t, but is not a higher-order pattern. In [Baumgartner et al., 2017] there is an instance for syntactic anti-unification: if $t = \lambda x, y.f(h(x, x, y), h(x, y, y))$ and $s = \lambda x, y.f(g(x, x, y), g(x, y, y))$, then $r = \lambda x, y.f(Y_1(x, y), Y_2(x, y))$ is a higher-order pattern, which is an lgg of t and s. However, the term $\lambda x, y.f(Z(x, x, y), Z(x, y, y))$, which is not a higher-order pattern, is less general than r and generalizes t and s.

Another important distinguishing feature of higher-order pattern generalization modulo $\mathcal{E}$ is that there may be more than one least general pattern generalization (lgpg) for a given pair of terms. In the syntactic case there is a unique lgpg. The main contribution of this paper is to find conditions on the AUPs under which there is a unique lgpg for equational cases, and introduce weaker-optimality conditions which allow one to greedily search the space for a less general generalization compared to the syntactic one. We formalize these concepts in the following sections.

3. Higher Order Pattern Generalization in the Empty Theory

Below we assume that for AUPs of the form $X(\vec{x}) : t \triangleq s$, the term $\lambda\vec{x}.X(\vec{x})$ is a higher-order pattern. We now introduce the rules for the higher-order pattern generalization algorithm from [Baumgartner et al., 2017], which works for $\mathcal{E} = \varnothing$. It produces syntactic higher-order pattern generalizations in linear time and will play a key role in our optimality conditions introduced in later sections.

These rules work on triples $A; S; \sigma$, which are called *states*. Here A is a set of AUPs of the form $\{X_1(\vec{x_1}) : t_1 \triangleq s_1, \ldots, X_n(\vec{x_n}) : t_n \triangleq s_n\}$ that are pending to anti-unify, S is a set of already solved AUPs (the *store*), and σ is a substitution (computed so far) mapping variables to patterns. The symbol $\uplus$ denotes disjoint union.

Dec: **Decomposition**

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

where h is a free constant or $\mathsf{h} \in \vec{x}$, and $Y_1, \ldots, Y_m$ are fresh variables of the appropriate types.

Abs: **Abstraction Rule**

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

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

Sol: **Solve Rule**

$$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ \sigma \Longrightarrow A;\ \{Y(\vec{y}) : t \triangleq s\} \cup S;\ \sigma\{X \mapsto \lambda\vec{x}.Y(\vec{y})\},$$

where t and s are of a base type, $head(t) \neq head(s)$ or $head(t) = head(s) = Z \notin \vec{x}$. The sequence $\vec{y}$ is a subsequence of $\vec{x}$ consisting of the variables that appear freely in t or in s, and Y is a fresh variable of the appropriate type.

Although it is not necessary for this version of Solve, we can impose an extra condition on its application requiring that $\mathsf{U} \notin Ax(head(t)) \cup Ax(head(s))$. This condition will become useful later, when we consider theories with the unit element.

Mer: **Merge Rule**

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

where $\pi : \{\vec{x}\} \to \{\vec{y}\}$ is a bijection, extended as a substitution with $t_1\pi = s_1$ and $t_2\pi = s_2$. Note that in the case of the equational theory we will consider later we would use $\equiv_{\mathcal{E}}$ instead of $=$.

We will refer to these generalization rules as $\mathcal{G}_{\text{base}}$. To compute generalizations for two simply typed lambda-terms in η-long β-normal form t and s, the algorithm from [Baumgartner et al., 2017] starts with the *initial state* $\{X : t \triangleq s\}; \varnothing; Id$, where X is a fresh variable, and applies these rules as long as possible. The computed result is the instance of X under the final substitution. It is the syntactic least general higher-order pattern generalization of t and s, and is computed in linear time in the size of the input.

One may notice that an AUP of the form $X(\vec{x}) : Z(s_1, \ldots, s_m) \triangleq Z(t_1, \ldots, t_m)$, where Z is a free variable, is transformed by Sol rather than by the Dec rule. This is because applying decomposition may result into a generalization which is not a higher-order pattern. A simple example is the AUP $X : \lambda x.Z(x, a) \triangleq \lambda x.Z(x, a)$. The algorithm returns the pattern $\lambda x.Y(x)$ as its generalization, while the application of Dec would lead to the generalization $\lambda x.Z(x, a)$, which is not a pattern. However, when an AUP has the form $X(\vec{x}) : c \triangleq c$, where c is a constant or one of the variables in $\vec{x}$, we apply the decomposition rule, i.e. $\{X : c \triangleq c\};\ \varnothing;\ Id \Longrightarrow_{\mathsf{Dec}} \varnothing;\ \varnothing;\ \{X \mapsto c\}$.

To illustrate the use of the above procedure, let us consider the following example from [Baumgartner et al., 2017]:

Example 1. Let $t = \lambda x, y.f(U(g(x), y), U(g(y), x))$ and $s = \lambda x', y'.f(h(y', g(x')), h(x', g(y')))$. Then the algorithm performs the following transformations:

$$\{X : \lambda x, y.f(U(g(x), y), U(g(y), x)) \triangleq \lambda x', y'.f(h(y', g(x')), h(x', g(y')))\};$$
$$\varnothing; Id \Longrightarrow^{2}_{\mathsf{Abs}}$$
$$\{X'(x, y) : f(U(g(x), y), U(g(y), x)) \triangleq f(h(y, g(x)), h(x, g(y)))\};\ \varnothing;$$
$$\{X \mapsto \lambda x, y.X'(x, y)\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{Y_1(x, y) : U(g(x), y) \triangleq h(y, g(x)), Y_2(x, y) : U(g(y), x) \triangleq h(x, g(y))\};\ \varnothing;$$
$$\{X \mapsto \lambda x, y.f(Y_1(x, y), Y_2(x, y))\} \Longrightarrow_{\mathsf{Sol}}$$
$$\{Y_2(x, y) : U(g(y), x) \triangleq h(x, g(y))\}; \{Y_1(x, y) : U(g(x), y) \triangleq h(y, g(x))\};$$

$$\{X \mapsto \lambda x, y.f(Y_1(x,y), Y_2(x,y))\} \Longrightarrow_{\mathsf{Sol}}$$
$$\varnothing;\ \{Y_1(x,y) : U(g(x),y) \triangleq h(y,g(x)), Y_2(x,y) : U(g(y),x) \triangleq h(x,g(y))\};$$
$$\{X \mapsto \lambda x, y.f(Y_1(x,y), Y_2(x,y))\} \Longrightarrow_{\mathsf{Mer}}$$
$$\varnothing;\ \{Y_1(x,y) : U(g(x),y) \triangleq h(y,g(x))\};\ \{X \mapsto \lambda x, y.f(Y_1(x,y), Y_1(y,x))\}.$$

The computed result is $r = \lambda x, y.f(Y_1(x,y), Y_1(y,x))$. It generalizes the input terms t and s: $r\{Y_1 \mapsto \lambda x, y.U(g(x),y)\} = t$ and $r\{Y_1 \mapsto \lambda x, y.h(y,g(x))\} = s$. These substitutions can be read from the final store.

We will use this linear time procedure in the following section to obtain "optimal" least general higher-order pattern generalizations of terms modulo an equation theory. These optimal generalizations are dependent on the generalizations the syntactic algorithm produces. When we need to check more than one decomposition of a given AUP in order to compute the optimal generalizations modulo an equational theory, we compute the optimal generalization for each decomposition path and then compare the results. The details are explained below.

4. Equational Decomposition Rules: A, C, and AC Theories

In this section we discuss an extension of the basic rules concerning higher-order pattern generalization by decomposition rules for A-, C-, and AC- theories. Here, we consider the general, unrestricted case. The theory with the unit element is considered separately in the next section. Efficient special restrictions are discussed in the subsequent section.

We assume that terms, which use polyadic version of associative symbols, are written in *flattened form* obtained by replacing all subterms of the form $f(t_1, \ldots, f(s_1, \ldots, s_m), \ldots, t_n)$ by $f(t_1, \ldots, s_1, \ldots, s_m, \ldots, t_n)$, where $\mathsf{A} \in Ax(f)$. Also, by convention, the term $f(t)$ stands for t, if $\mathsf{A} \in Ax(f)$.

4.1. *Associative Decomposition Rules*

We start from decomposition rules for associative generalization:

Dec-A-L: **Associative Decomposition Left**

$$\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1(\vec{x}) : f(t_1, \ldots, t_k) \triangleq s_1,\ Y_2(\vec{x}) : f(t_{k+1}, \ldots, t_n) \triangleq f(s_2, \ldots, s_m)\} \cup A;$$
$$S;\ \sigma\{X \mapsto \lambda \vec{x}.f(Y_1(\vec{x}), Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{A}\}$, $n, m \geqslant 2$, $1 \leqslant k \leqslant n-1$, and Y_1 and Y_2 are fresh variables of appropriate types.

Dec-A-R: **Associative Decomposition Right**

$$\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow \\ \{Y_1(\vec{x}) : t_1 \triangleq f(s_1, \ldots, s_k),\ Y_2(\vec{x}) : f(t_2, \ldots, t_n) \triangleq f(s_{k+1}, \ldots, s_m)\} \cup A; \\ S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}), Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{A}\}$, $n, m \geqslant 2$, $1 \leqslant k \leqslant m-1$, and Y_1 and Y_2 are fresh variables of appropriate types.

We refer to the extension of $\mathcal{G}_{\text{base}}$ by the above associativity rules as $\mathcal{G}_A$ and extend the termination, soundness and completeness results for $\mathcal{G}_{\text{base}}$ to $\mathcal{G}_A$. To illustrate the use of the above extension of $\mathcal{G}_{\text{base}}$, let us consider the following example where $Ax(f) = \{\mathsf{A}\}$:

Example 2. Let $t = \lambda x.\lambda y.f(x, x, y, y)$ and $s = \lambda z.\lambda v.f(z, v, v)$ be in flattened form. The initial state is $\{X : t \triangleq s\}; \varnothing; Id$. First, we apply the abstraction rule twice:

$$\{X : t \triangleq s\}; \varnothing; Id \Longrightarrow^{\times 2}_{\mathsf{Abs}} \\ \{X'(x, y) : f(x, x, y, y) \triangleq f(x, y, y)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\}.$$

From here, we can continue in multiple ways, applying Dec-A-L or Dec-A-R, each of them in various positions. Assume that we use Dec-A-L at position 2, i.e., with the index k being set to 2 (note that we flatten nested f's also in the substitutions):

$$\{X'(x, y) : f(x, x, y, y) \triangleq f(x, y, y)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\} \Longrightarrow^{k=2}_{\mathsf{Dec\text{-}A\text{-}L}} \\ \{X_1(x, y) : f(x, x) \triangleq x,\ X_2(x, y) : f(y, y) \triangleq f(y, y)\}; \varnothing; \\ \quad \{X \mapsto \lambda x.\lambda y.f(X_1(x, y), X_2(x, y)), \ldots\} \Longrightarrow_{\mathsf{Sol}} \\ \{X_2(x, y) : f(y, y) \triangleq f(y, y)\};\ \{Y(x) : f(x, x) \triangleq x\}; \\ \quad \{X \mapsto \lambda x.\lambda y.f(Y(x), X_2(x, y)), \ldots\} \Longrightarrow^{\times 3}_{\mathsf{Dec}} \\ \varnothing;\ \{Y(x) : f(x, x) \triangleq x\};\ \{X \mapsto \lambda x.\lambda y.f(Y(x), y, y), \ldots\}.$$

The derivation stops here with the computed answer $\lambda x.\lambda y.f(Y(x), y, y)$.

Now assume that the Dec-A-L above was used not at position 2, but at position 1. It will lead to the computation of another lgg, which shows that for the associative case, there exists more than one lgg. (In contrast, higher-order pattern anti-unification in the free theory from [Baumgartner et al., 2017] always results into a unique lgg.)

$$\{X'(x, y) : f(x, x, y, y) \triangleq f(x, y, y)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\} \Longrightarrow^{k=1}_{\mathsf{Dec\text{-}A\text{-}L}} \\ \{X_1(x, y) : x \triangleq x,\ X_2(x, y) : f(x, y, y) \triangleq f(y, y)\}; \varnothing; \\ \quad \{X \mapsto \lambda x.\lambda y.f(X_1(x, y), X_2(x, y)), \ldots\} \Longrightarrow_{\mathsf{Dec}} \\ \{X_2(x, y) : f(x, y, y) \triangleq f(y, y)\};\ \varnothing;\ \{X \mapsto \lambda x.\lambda y.f(x, X_2(x, y)), \ldots\}.$$

Here there are also multiple ways to proceed. We show one of them, by the Dec-A-L rule applied at position 2:

$$\{X_2(x, y) : f(x, y, y) \triangleq f(y, y)\};\ \varnothing;\ \{X \mapsto \lambda x.\lambda y.f(x, X_2(x, y)), \ldots\} \Longrightarrow^{k=2}_{\mathsf{Dec\text{-}A\text{-}L}} \\ \{Y_1(x, y) : f(x, y) \triangleq y,\ Y_2(x, y) : y \triangleq y\}; \varnothing; \\ \quad \{X \mapsto \lambda x.\lambda y.f(x, Y_1(x, y), Y_2(x, y)), \ldots\} \Longrightarrow_{\mathsf{Sol}}$$

$$\{Y_2(x,y) : y \triangleq y\};\ \{Y(x,y) : f(x,y) \triangleq y\};$$
$$\{X \mapsto \lambda x.\lambda y.f(x, Y(x,y), Y_2(x,y)), \ldots\} \Longrightarrow_{\mathsf{Dec}}$$
$$\varnothing;\ \{Y(x,y) : f(x,y) \triangleq y\};\ \{X \mapsto \lambda x.\lambda y.f(x, Y(x,y), y), \ldots\}\,.$$

Hence, we obtained another lgg $\lambda x.\lambda y.f(x, Y(x,y), y)$.

Theorem 1 (Termination). The set of transformations $\mathcal{G}_A$ is terminating.

Proof. Termination follows from the fact that $\mathcal{G}_{\text{base}}$ terminates [Baumgartner et al., 2017] and the rules Dec-A-L and Dec-A-R can be applied finitely many times. □

Theorem 2 (Soundness). If $\{X : t \triangleq s\}; \varnothing; Id \Longrightarrow^* \varnothing; S; \sigma$ is a transformation sequence of $\mathcal{G}_A$, then $X\sigma$ is a higher-order pattern in η-long β-normal form and $X\sigma \preceq t$ and $X\sigma \preceq s$.

Proof. It was shown in [Baumgartner et al., 2017] that $\mathcal{G}_{\text{base}}$ is sound and always results in a higher-order pattern. The associative decomposition rules replace free variables with higher-order patterns in substitutions. Composition of pattern substitutions is again a pattern substitution. Therefore, the associative generalization algorithm also returns higher-order patterns.

The second part of the theorem we prove by induction on the number of arguments of associative function constants appearing in $t \triangleq s$. Let us assume as a base case that all occurrences of associative constants in $t \triangleq s$ have two arguments. Then the rules Dec-A-L and Dec-A-R are equivalent to the Dec rule. As an induction hypothesis (IH), assume soundness holds when all occurrences of associative constants in $t \triangleq s$ have $\leqslant n$ arguments. We show that it holds for $n+1$. Let $t \triangleq s$ be of the form $f(t_1, \ldots, t_m) \triangleq f(s_1, \ldots, s_k)$ for $\max\{m,k\} \leqslant (n+1)$ and let associative constants occurring in $t_1, \ldots t_m, s_1, \ldots s_k$ have at most n arguments. Any application of Dec-A-L or Dec-A-R will produce two AUPs for which the IH holds, and thus, the theorem holds. We can extend this argument to an arbitrary number of associative constants with $n+1$ arguments with another induction. □

Theorem 3 (Completeness). Let $\lambda\vec{x}.t_1$ and $\lambda\vec{x}.t_2$ be higher-order terms and $\lambda\vec{x}.s$ be a higher-order pattern such that $\lambda\vec{x}.s$ is a generalization of both $\lambda\vec{x}.t_1$ and $\lambda\vec{x}.t_2$ modulo associativity. Then there exists a transformation sequence $\{X(\vec{x}) : t_1 \triangleq t_2\}; \varnothing; Id \Longrightarrow^* \varnothing; S; \sigma$ in $\mathcal{G}_A$ such that $\lambda\vec{x}.s \preceq X\sigma$.

Proof. We can reason similarly to the previous proof. It was shown in [Baumgartner et al., 2017] that $\mathcal{G}_{\text{base}}$ is complete. Let us assume as a base case that all occurrences of associative function constants in $t \triangleq s$ have two arguments. Then the rules Dec-A-L and Dec-A-R are equivalent to the Dec rule and completeness holds. When we have $n+1$ arguments there are n ways to group the arguments associatively and the decomposition rules Dec-A-L and Dec-A-R allow one to consider all groupings. □

If we wish to compute the complete set of lggs we would simply exhaust all possible applications of the above rules. However, for most applications an "optimal" generalization is sufficient. We postpone discussion till the next section.

4.2. *Commutative Decomposition Rules*

The decomposition rules for commutative symbols is also pretty intuitive:

Dec-C: **Commutative Decomposition**

$$\{X(\vec{x}) : f(t_1, t_2) \triangleq f(s_1, s_2)\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1(\vec{x}) : t_1 \triangleq s_i,\ Y_2(\vec{x}) : t_2 \triangleq s_{(i \mod 2)+1}\} \cup A;\ S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}), Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{C}\}$, $i \in \{1, 2\}$, and Y_1 and Y_2 are fresh variables of appropriate types.

We refer to the extension of $\mathcal{G}_{\text{base}}$ by the commutativity rule as $\mathcal{G}_C$. To illustrate the use of the above extension of $\mathcal{G}_{\text{base}}$, let us consider the following example where $Ax(f) = \{\mathsf{C}\}$:

Example 3. Let $t = \lambda x.\lambda y.f(g(x, y), g(y, x))$ and $s = \lambda z.\lambda w.f(w, g(z, z))$. The initial state is $\{X : t \triangleq s\}; \varnothing; Id$. After applying the Abs rule twice we reach

$$\{X'(x, y) : f(g(x, y), g(y, x)) \triangleq f(x, g(y, y))\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\},$$

from here there are two way to proceed: applying the Dec-C with $i = 1$ and with $i = 2$.

The derivation with $i = 1$ is as follows:

$$\{X'(x, y) : f(g(x, y), g(y, x)) \triangleq f(x, g(y, y))\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\} \Longrightarrow^{i=1}_{\text{Dec-C}}$$
$$\{X_1(x, y) : g(x, y) \triangleq x,\ X_2(x, y) : g(y, x) \triangleq g(y, y)\}; \varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.f(X_1(x, y), X_2(x, y)), \ldots\} \Longrightarrow_{\text{Sol}}$$
$$\{X_2(x, y) : g(y, x) \triangleq g(y, y)\}; \{Y(x, y) : g(x, y) \triangleq x\};$$
$$\{X \mapsto \lambda x.\lambda y.f(Y(x, y), X_2(x, y)), \ldots\} \Longrightarrow^{\times 2}_{\text{Dec}}$$
$$\{Z_1(x, y) : x \triangleq y\}; \{Y(x, y) : g(x, y) \triangleq x\};$$
$$\{X \mapsto \lambda x.\lambda y.f(Y(x, y), g(y, Z_1(x, y))), \ldots\} \Longrightarrow_{\text{Sol}}$$
$$\varnothing; \{Y(x, y) : g(x, y) \triangleq x,\ Z(x, y) : x \triangleq y\};$$
$$\{X \mapsto \lambda x.\lambda y.f(Y(x, y), g(y, Z(x, y))), \ldots\}.$$

The obtained lgg is $\lambda x.\lambda y.f(Y(x, y), g(y, Z(x, y)))$.

Taking $i = 2$ produces the following derivation:

$$\{X'(x, y) : f(g(x, y), g(y, x)) \triangleq f(x, g(y, y))\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\} \Longrightarrow^{i=2}_{\text{Dec-C}}$$
$$\{X_1(x, y) : g(x, y) \triangleq g(y, y),\ X_2(x, y) : g(y, x) \triangleq x\}; \varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.f(X_1(x, y), X_2(x, y)), \ldots\} \Longrightarrow_{\text{Dec}}$$
$$\{Y_1(x, y) : x \triangleq y,\ Y_2(x, y) : y \triangleq y,\ X_2(x, y) : g(y, x) \triangleq x\}; \varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.f(g(Y_1(x, y), Y_2(x, y)), X_2(x, y)), \ldots\} \Longrightarrow_{\text{Sol}}$$
$$\{Y_2(x, y) : y \triangleq y,\ X_2(x, y) : g(y, x) \triangleq x\}; \{Y(x, y) : x \triangleq y\};$$
$$\{X \mapsto \lambda x.\lambda y.f(g(Y(x, y), Y_2(x, y)), X_2(x, y)), \ldots\} \Longrightarrow_{\text{Dec}}$$
$$\{X_2(x, y) : g(y, x) \triangleq x\}; \{Y(x, y) : x \triangleq y\};$$
$$\{X \mapsto \lambda x.\lambda y.f(g(Y(x, y), y), X_2(x, y)), \ldots\} \Longrightarrow_{\text{Sol}}$$
$$\varnothing; \{Y(x, y) : x \triangleq y,\ Z(x, y) : g(y, x) \triangleq x\};$$
$$\{X \mapsto \lambda x.\lambda y.f(g(Y(x, y), y), Z(x, y)), \ldots\}.$$

Hence, we obtain the second lgg $\lambda x.\lambda y.f(g(Y(x,y),y),Z(x,y))$.

Hence, in commutative generalization, like in the associative case, the lgg is not necessarily unique.

We can easily extend the termination, soundness, and completeness results to $\mathcal{G}_{\mathsf{C}}$.

4.3. *Associative-Commutative Decomposition Rules*

Unlike commutativity, which considers a fixed number of terms, and associativity, which enforces an ordering on terms, AC constants allow an arbitrary number of arguments with no fixed ordering on the terms. The corresponding decomposition rules take it into account:

Dec-AC-L: **Associative-Commutative Decomposition Left**

$$\begin{array}{l}\{X(\vec{x}) : f(t_1,\ldots,t_n) \triangleq f(s_1,\ldots,s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow \\ \quad \{Y_1(\vec{x}) : f(t_{i_1},\ldots,t_{i_l}) \triangleq s_k, \\ \quad\ Y_2(\vec{x}) : f(t_{i_{(l+1)}},\ldots,t_{i_n}) \triangleq f(s_1,\ldots,s_{k-1},s_{k+1},\ldots,s_m)\} \cup A; \\ \quad S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}),Y_2(\vec{x}))\},\end{array}$$

where $Ax(f) = \{\mathsf{A},\mathsf{C}\}, \{i_1,\ldots,i_n\} \equiv \{1,\ldots,n\}, l \in \{1,\ldots,n-1\}, k \in \{1,\ldots,m\}, n,m \geqslant 2$, and Y_1 and Y_2 are fresh variables of appropriate types.

Dec-AC-R: **Associative-Commutative Decomposition Right**

$$\begin{array}{l}\{X(\vec{x}) : f(t_1,\ldots,t_n) \triangleq f(s_1,\ldots,s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow \\ \quad \{Y_1(\vec{x}) : t_k \triangleq f(s_{i_1},\ldots,s_{i_l}), \\ \quad\ Y_2(\vec{x}) : f(t_1,\ldots,t_{k-1},t_{k+1},\ldots,t_n) \triangleq f(s_{i_{(l+1)}},\ldots,s_{i_m})\} \cup A; \\ \quad S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}),Y_2(\vec{x}))\},\end{array}$$

where $Ax(f) = \{\mathsf{A},\mathsf{C}\}, \{i_1,\ldots,i_m\} \equiv \{1,\ldots,m\}, l \in \{1,\ldots,m-1\}, k \in \{1,\ldots,n\}, n,m \geqslant 2$, and Y_1 and Y_2 are fresh variables of appropriate types.

We refer to the extension of $\mathcal{G}_{\mathrm{base}}$ by the AC decomposition rules as $\mathcal{G}_{\mathsf{AC}}$. To illustrate the use of the above extension of $\mathcal{G}_{\mathrm{base}}$, let us consider the following example where $Ax(f) = \{\mathsf{A},\mathsf{C}\}$:

Example 4. Let $t = \lambda x.\lambda y.f(x,x,y)$ and $s = \lambda z.\lambda v.f(v,v,z)$. Starting from the initial state $\{X : t \triangleq s\};\emptyset;Id$, after two applications of Dec we reach the state $\{X'(x,y) : f(x,x,y) \triangleq f(y,y,x)\};\emptyset;\{X \mapsto \lambda x.\lambda y.X'(x,y)\}$, which can be further transformed in multiple ways. We present two derivations. The first one starts with Dec-AC-L, where $k = 3$ (the position), $l = 1$ (the subsequence length) and looks as follows:

$$\begin{array}{l}\{X'(x,y) : f(x,x,y) \triangleq f(y,y,x)\};\emptyset;\{X \mapsto \lambda x.\lambda y.X'(x,y)\} \Longrightarrow^{k=3,\,l=1}_{\textsf{Dec-AC-L}} \\ \{Y_1(x,y) : x \triangleq x,\ Y_2(x,y) : f(x,y) \triangleq f(y,y)\};\emptyset; \\ \qquad \{X \mapsto \lambda x.\lambda y.f(Y_1(x,y),Y_2(x,y)),\ldots\} \Longrightarrow_{\textsf{Dec}} \\ \{Y_2(x,y) : f(x,y) \triangleq f(y,y)\};\emptyset;\{X \mapsto \lambda x.\lambda y.f(x,Y_2(x,y)),\ldots\} \Longrightarrow^{k=2,\,l=1}_{\textsf{Dec-AC-R}} \\ \{Z_1(x,y) : y \triangleq y,\ Z_2(x,y) : x \triangleq y\};\emptyset; \\ \qquad \{X \mapsto \lambda x.\lambda y.f(x,Z_1(x,y),Z_2(x,y)),\ldots\} \Longrightarrow_{\textsf{Dec}} \\ \{Z_2(x,y) : x \triangleq y\};\emptyset;\{X \mapsto \lambda x.\lambda y.f(x,y,Z_2(x,y)),\ldots\} \Longrightarrow_{\textsf{Sol}}\end{array}$$

$\varnothing; \{Z(x,y) : x \triangleq y\}; \{X \mapsto \lambda x.\lambda y.f(x,y,Z(x,y)), \ldots\}$.

Hence, the obtained lgg is $\lambda x.\lambda y.f(x,y,Z(x,y))$.
The second derivation is the following:

$\{X'(x,y) : f(x,x,y) \triangleq f(y,y,x)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x,y)\} \Longrightarrow^{k=1,\,l=1}_{\text{Dec-AC-L}}$
$\{Y_1(x,y) : x \triangleq y,\ Y_2(x,y) : f(x,y) \triangleq f(y,y)\}; \varnothing;$
$\quad \{X \mapsto \lambda x.\lambda y.f(Y_1(x,y), Y_2(x,y)), \ldots\} \Longrightarrow_{\text{Sol}}$
$\{Y_2(x,y) : f(x,y) \triangleq f(y,y)\}; \{Y(x,y) : x \triangleq y\};$
$\quad \{X \mapsto \lambda x.\lambda y.f(Y(x,y), Y_2(x,y)), \ldots\} \Longrightarrow^{k=1,\,l=1}_{\text{Dec-AC-L}}$
$\{Z_1(x,y) : x \triangleq y,\ Z_2(x,y) : y \triangleq x\}; \{Y(x,y) : x \triangleq y\};$
$\quad \{X \mapsto \lambda x.\lambda y.f(Y(x,y), Z_1(x,y), Z_2(x,y)), \ldots\} \Longrightarrow_{\text{Sol}}$
$\{Z_2(x,y) : y \triangleq x\}; \{Y(x,y) : x \triangleq y,\ Z'(x,y) : x \triangleq y\};$
$\quad \{X \mapsto \lambda x.\lambda y.f(Y(x,y), Z'(x,y), Z_2(x,y))\} \Longrightarrow_{\text{Sol}}$
$\varnothing; \{Y(x,y) : x \triangleq y,\ Z'(x,y) : x \triangleq y,\ Z''(x,y) : y \triangleq x\};$
$\quad \{X \mapsto \lambda x.\lambda y.f(Y(x,y), Z'(x,y), Z''(x,y))\} \Longrightarrow_{\text{Mer}}$
$\varnothing; \{Y(x,y) : x \triangleq y,\ Z''(x,y) : y \triangleq x\};$
$\quad \{X \mapsto \lambda x.\lambda y.f(Y(x,y), Y(x,y), Z''(x,y))\} \Longrightarrow_{\text{Mer}}$
$\varnothing; \{Y(x,y) : x \triangleq y\}; \{X \mapsto \lambda x.\lambda y.f(Y(x,y), Y(x,y), Y(y,x))\}$.

Hence, the second lgg is $\lambda x.\lambda y.f(Y(x,y), Y(x,y), Y(y,x))$.

Again, termination, soundness and completeness are easily extended to this case.

5. Theories with the Unit Element

5.1. *Generalization Modulo* U

A peculiarity of theories with unit elements is that terms with different heads may have nontrivial least general generalizations. For instance, the lgg of $\lambda x.f(a,x)$ and $\lambda x.a$ is $\lambda x.f(a,X(x))$, if f has the unit element. (Otherwise, $\lambda x.X(x)$ would have been the lgg.) In order not to miss such generalizations, we should not use the Solve rule for the AUP $X(\vec{x}) : t \triangleq s$, if the head of t or of s is a constant f such that $\mathsf{U} \in Ax(f)$. Instead, the following expansion rules should be applied:

Exp-U-L: **Expansion for Unit, Left**

$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ \sigma \Longrightarrow \{X(\vec{x}) : t' \triangleq s\} \uplus A;\ S;\ \sigma$,

where $f = head(s) \neq head(t)$, $\mathsf{U} \in Ax(f)$, ϵ_f is the unit element of f, $t' \in \{f(t,\epsilon_f), f(\epsilon_f,t)\}$.

Exp-U-R: **Expansion for Unit, Right**

$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ \sigma \Longrightarrow \{X(\vec{x}) : t \triangleq s'\} \uplus A;\ S;\ \sigma$,

where $f = head(t) \neq head(s)$, $\mathsf{U} \in Ax(f)$, ϵ_f is the unit element of f, $s' \in \{f(s,\epsilon_f), f(\epsilon_f,s)\}$.

Extending the base algorithm with these rules (and modifying Solve as described above) gives

an algorithm whose soundness is straightforward. Termination is also easy to see because the expansion rules are to be followed by the decomposition and the problem becomes strictly smaller than it was before the expansion. However, it turns out that the algorithm is not complete, as the following example shows:

Example 5. Let $t = g(a, a)$ and $s = f(g(b, \epsilon_f), b)$, where $Ax(f) = \{\mathsf{U}\}$ and ϵ_f is the unit element of f. Note that these terms come from the first-order fragment of the term language. Then we have a derivation:

$$\{X : g(a,a) \triangleq f(g(b,\epsilon_f),b)\};\ \varnothing;\ Id \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}L}}$$
$$\{X : f(g(a,a),\epsilon_f) \triangleq f(g(b,\epsilon_f),b)\};\ \varnothing;\ Id \Longrightarrow_{\mathsf{Dec}}$$
$$\{X_1 : g(a,a) \triangleq g(b,\epsilon_f),\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(X_1,X_2)\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{Y_1' : a \triangleq b,\ Y_2' : a \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(g(Y_1',Y_2'),X_2)\} \Longrightarrow^3_{\mathsf{Sol}}$$
$$\varnothing;\ \{Y_1 : a \triangleq b,\ Y_2 : a \triangleq \epsilon_f,\ Y_3 : \epsilon_f \triangleq b\};\ \{X \mapsto f(g(Y_1,Y_2),Y_3)\}.$$

With another term choice $f(\epsilon_f, g(a, a))$ in Exp-U-L we would get a derivation of a more general generalization $f(X_1, X_2)$. However, even $f(g(Y_1, Y_2), Y_3)$ is not least general: It is strictly more general than $f(g(f(Z_1, Z_2), Z_1), Z_3)$. To get convinced that the latter is indeed a generalization of t and s, take

$$f(g(f(Z_1,Z_2),Z_1),Z_3)\{Z_1 \mapsto a,\ Z_2 \mapsto \epsilon_f, Z_3 \mapsto \epsilon_f\} \equiv_{\mathsf{U}} f(g(a,a),\epsilon_f) \equiv_{\mathsf{U}} g(a,a) = t,$$
$$f(g(f(Z_1,Z_2),Z_1),Z_3)\{Z_1 \mapsto \epsilon_f,\ Z_2 \mapsto b,\ Z_3 \mapsto b\} \equiv_{\mathsf{U}} f(g(b,\epsilon_f),b) = s.$$

The problem highlighted in Example 5 is related to the fact that from two U-equigeneral terms Y and $f(Z_1, Z_2)$ sometimes we have to choose one in the generalization and sometimes another, depending which variable can occur more than once in the generalization. However, if we compute linear generalizations (i.e., no variable occurring more than once and, hence, Merge rule is not applied), then there is no need to consider $f(Z_1, Z_2)$ as an alternative of Y (as a generalization). Notice that in Example 5, Z_1 has multiple occurrences allowing one to add additional occurrences of f. While we do not go into detail in this paper concerning how to handle unit in the non-linear case, we conjecture that a terminating complete algorithm is possible using a similar framework as was introduced in [Cerna and Kutsia, 2019b] where a terminating and complete algorithm was provided for idempotent generalization. In that case, the expansion rules cover all alternatives to "repair" head disagreement between two terms, where the head of one of those terms has the unit element. Therefore, we call the algorithm $\mathcal{G}_{\mathsf{U\text{-}lin}}$.

For the general case, one might hope to take an advantage of the unit element and generalize even arbitrary head-different terms.[‡] The following rule would deal with all such possibilities:

DH-U: **Terms with Different Heads in the Unit Element Theory**

$$\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1(\vec{x}) : t \triangleq \epsilon_f,\ Y_2(\vec{x}) : \epsilon_f \triangleq s\} \uplus A;\ S;\ \sigma\{X \mapsto f(Y_1(\vec{x}), Y_2(\vec{x}))\},$$

where $head(t) \neq head(s)$, $\mathsf{U} \in Ax(f)$, ϵ_f is the unit element of f, $t \neq \epsilon_f$, and $s \neq \epsilon_f$.

[‡] In U theory rules, when we write an AUP of the form $Y(\vec{x}) : t \triangleq \epsilon_f$ or $Y(\vec{x}) : \epsilon_f \triangleq t$, it is assumed that t and ϵ_f have the same type.

Extending $\mathcal{G}_{\mathsf{U\text{-}lin}}$ with DH-U, we would be able to compute the lgg for terms in Example 5, continuing the derivation that stopped there:

$$\{Y_1 : a \triangleq b,\ Y_2 : a \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(g(Y_1, Y_2), X_2)\} \Longrightarrow_{\mathsf{DH\text{-}U}}$$
$$\{Z_1 : a \triangleq \epsilon_f,\ Z_2 : \epsilon_f \triangleq b,\ Y_2 : a \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;$$
$$\{X \mapsto f(g(f(Z_1, Z_2), Y_2), X_2)\} \Longrightarrow^{4}_{\mathsf{Sol}}$$
$$\varnothing;\ \{Z_1 : a \triangleq \epsilon_f,\ Z_2 : \epsilon_f \triangleq b,\ Y_2 : a \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};$$
$$\{X \mapsto f(g(f(Z_1, Z_2), Y_2), X_2)\} \Longrightarrow_{\mathsf{Mer}}$$
$$\varnothing;\ \{Z_1 : a \triangleq \epsilon_f,\ Z_2 : \epsilon_f \triangleq b,\ X_2 : \epsilon_f \triangleq b\};\ \{X \mapsto f(g(f(Z_1, Z_2), Z_1), X_2)\} \Longrightarrow_{\mathsf{Mer}}$$
$$\varnothing;\ \{Z_1 : a \triangleq \epsilon_f,\ Z_2 : \epsilon_f \triangleq b\};\ \{X \mapsto f(g(f(Z_1, Z_2), Z_1), Z_2)\}.$$

While the use of DH-U can help to find lggs in the general case as in this example, it has a serious drawback: If we have more than one function constant with the unit element, it will generate an infinite branch in the derivation tree:[§]

$$\{X : a \triangleq b\};\ \varnothing;\ Id \Longrightarrow_{\mathsf{DH\text{-}U}}$$
$$\{X_1 : a \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(X_1, X_2)\} \Longrightarrow_{\mathsf{DH\text{-}U}}$$
$$\{Y_1 : a \triangleq \epsilon_g,\ Y_2 : \epsilon_g \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(g(Y_1, Y_2), X_2)\} \Longrightarrow_{\mathsf{DH\text{-}U}}$$
$$\{Z_1 : a \triangleq \epsilon_f,\ Z_2 : \epsilon_f \triangleq \epsilon_g,\ Y_2 : \epsilon_g \triangleq \epsilon_f,\ X_2 : \epsilon_f \triangleq b\};\ \varnothing;$$
$$\{X \mapsto f(g(f(Z_1, Z_2), Y_2), X_2)\} \Longrightarrow_{\mathsf{DH\text{-}U}}$$
$$\dots$$

Along the branch we will have generalizations

$$f(X_1, X_2),$$
$$f(g(Y_1, Y_2), X_2),$$
$$f(g(f(Z_1, Z_2), Y_2), X_2),$$
$$f(g(f(g(U_1, Y_2), Z_2), Y_2), X_2),$$
$$f(g(f(g(f(V_1, Z_2), Y_2), Z_2), Y_2), X_2),$$
$$\dots$$

but all of them are U-equigeneral. In fact, one can notice that by the repeated application of DH-U with more than one unit element, the same AUPs (with fresh generalization variables) are generated over and over again. Therefore, with a simple loop checking, or by setting the bound to the derivation depth based on the size/depth of the original problem, we can obtain a terminating algorithm for the general case as well.

Nevertheless, to avoid such unpleasant consequences of using the DH-U rule, in the rest of the paper we will restrict ourselves with the algorithm $\mathcal{G}_{\mathsf{U\text{-}lin}}$ when the equational axioms involve U. Hence, we will be interested in computing linear generalizations for those theories.

Theorem 4 (Completeness of $\mathcal{G}_{\mathsf{U\text{-}lin}}$). Let $\lambda\vec{x}.t_1$ and $\lambda\vec{x}.t_2$ be higher-order terms and $\lambda\vec{x}.s$ be

[§] A similar behavior can be observed in a related theory of idempotence, studied in [Cerna and Kutsia, 2019b].

a linear higher-order pattern such that $\lambda\vec{x}.s$ is a U-generalization of $\lambda\vec{x}.t_1$ and $\lambda\vec{x}.t_2$. Then there exists a transformation sequence $\{X : \lambda\vec{x}.t_1 \triangleq \lambda\vec{x}.t_2\}; \varnothing; Id \Longrightarrow^* \varnothing; S; \sigma$ in $\mathcal{G}_{\mathsf{U\text{-}lin}}$ such that $\lambda\vec{x}.s \preceq X\sigma$.

Proof. We assume that $\lambda\vec{x}.t_1$, $\lambda\vec{x}.t_2$, and $\lambda\vec{x}.s$ have been normalized, i.e. subterms of the form $f(t', \epsilon_f)$ or $f(\epsilon_f, t')$, where $\mathsf{U} \in Ax(f)$, do not occur. We prove the theorem by induction on $depth(\lambda\vec{x}.t_1) + depth(\lambda\vec{x}.t_2)$ which we denote by n.

Case 1: $n = 2$, i.e., t_1 and t_2 are constants.

a) First, assume that $depth(s) = 1$. If $t_1 = t_2$, then $s = t_1 = t_2$ and s is computed by the derivation $\{X : t_1 \triangleq t_2\}; \varnothing; Id \Longrightarrow_{\mathsf{Dec}} Id; \varnothing; \{X \mapsto t_1\}$. If $t_1 \neq t_2$, then s must be a variable, computed by the derivation $\{X : t_1 \triangleq t_2\}; \varnothing; Id \Longrightarrow_{\mathsf{Sol}} \varnothing; \{X : t_1 \triangleq t_2\}; Id$.

b) Now assume as the induction hypothesis that for every generalization s of t_1 and t_2 of depth at most k, either $s \preceq t_1$ and $t_1 = t_2$, or $s \preceq X$ and $t_1 \neq t_2$. We show that this holds for a generalization s' of depth $k + 1$. Let $head(s') = f$. Our assumptions imply that $\mathsf{U} \in \mathsf{A}x(f)$ because both t_1 and t_2 are of depth 1. Thus, $s' = f(s_1, s_2)$.

By the definition of a generalization, there must exists two substitutions σ_1 and σ_2 such that $s'\sigma_1 = t_1$ and $s'\sigma_2 = t_2$. If $s_1\sigma_1 = s_1\sigma_2 = \epsilon_f$ (resp. if $s_2\sigma_1 = s_2\sigma_2 = \epsilon_f$), then s_2 (resp., s_1) is, by the induction hypothesis, more general than t_1 when $t_1 = t_2$, or more general than X when $t_1 \neq t_2$. This implies, by the linearity assumption that there exists a substitution ϑ such that $s_2\vartheta = s_2$ and $s_1\vartheta = \epsilon_f$. Thus, $s'\vartheta = s_2$, i.e. $s' \prec s_2$.

However, if $s_2\sigma_1 = \epsilon_f$ and $s_1\sigma_2 = \epsilon_f$, or vice versa, then additional observations are required. We assume without loss of generality the former case.

If $t_1 = t_2$ then both s_1 and s_2 are generalizations of $t_1 \triangleq t_2$ and by the induction hypothesis $s_1 \preceq t_1$ and $s_2 \preceq t_1$. If $t_1 \neq t_2$ then we need to make a distinction:

b1. If neither t_1 nor t_2 are units of function constants f_{t_1} and f_{t_2}, respectively, which may appear in s, then there exists a variable Y occurring in s_1 such that $Y\sigma_1 = t_1$ and a variable Y' occurring in s_2 such that $Y'\sigma_2 = t_2$. However, by the linearity of S, this implies that there exist two substitutions σ_1' and σ_2' which coincide everywhere with σ_1 and σ_2 except on Y and Y' respectively. That is, $Y\sigma_1' = t_2$ and $Y'\sigma_2' = t_1$. This implies that both s_1 and s_2 are generalizations of $t_1 \triangleq t_2$ which have depth $\leqslant k + 1$. Thus, $s_1 \preceq X$ and $s_2 \preceq X$.

b2. If either t_1 or t_2 is a unit of the function constants f_{t_1} and f_{t_2}, respectively, which may appear in s, then additional observations are necessary. If neither t_1 or t_2 occurs in s then we have the same situation as b1. Otherwise, if f_{t_1} occurs in s_1 (respectively f_{t_2} in s_2 then it must occur as the head symbol of a term with t_1 as a subterm because $s_1\sigma_2 = \epsilon_f$. This implies that there must be a variable Y in s_1 which σ_1 maps to t_1. Similar can be said concerning s_2, t_2, and σ_2. We can construct a new substitution which coincides with σ_1 (respectively, with σ_2) everywhere but on the variable Y (resp. Y') which it maps to t_2 (resp. to t_1). This means that s_1 and s_2 are generalizations of $t_1 \triangleq t_2$ and by the induction hypothesis $s_1 \preceq X$ $s_2 \preceq X$. This completes the case 1.

Case 2: $n > 2$.

a) Let us assume that $t_1 = \lambda y.t'_1$ and $t_2 = \lambda z.t'_2$, then it must be the case that $s = \lambda y.s'$ where s' is a generalization of the AUP $X(y) : t'_1 \triangleq t'_2$. Note that $depth(t'_1) + depth(t'_2) = n - 1$ and thus by the induction hypothesis there exists a generalization s^* computable using the rules of $\mathcal{G}_{\mathsf{U\text{-}lin}}$ such that $s' \preceq s^*$. Thus, a generalization for the AUP $t_1 \triangleq t_2$ may be computed using the $\mathcal{G}_{\mathsf{U\text{-}lin}}$ by first applying the abs rule to $t_1 \triangleq t_2$ and then computing s^*. Thus, $\lambda y.s^*$ is a generalization of $t_1 \triangleq t_2$.

b) Let us assume that $t_1 = f(w_1, \ldots, w_m)$ and $t_2 = f(r_1, \ldots, r_m)$, such that $\mathsf{U} \notin \mathsf{Ax}(f)$ then by applying the Dec rule to the AUP $X : t_1 \triangleq t_2$ we get m AUPs $X_1 : w_1 \triangleq r_1, \ldots, X_m : w_1 \triangleq r_1$ each of which has a depth sum of $n-1$. Thus, by the induction hypothesis, for each generalization s' generalizing $X_i : w_i \triangleq r_i$ there exists a generalization s^*_i, computed using $\mathcal{G}_{\mathsf{U\text{-}lin}}$, such that, $s' \preceq s^*_i$. Now let S^*_i be the set of all such generalizations computed using $\mathcal{G}_{\mathsf{U\text{-}lin}}$. We may now define the set of generalizations S^* as follows:

$$S^* = \{f(s^*_1, \ldots, s^*_m) \mid s^*_i \in S^*_i \text{ for all } 1 \leqslant i \leqslant m\}.$$

Note that each term in S^* is a generalization of $X : t_1 \triangleq t_2$ computed using $\mathcal{G}_{\mathsf{U\text{-}lin}}$. Thus, any generalization s' of $X : t_1 \triangleq t_2$ such that $head(s') = f$ is more general than some generalization of S^*. Thus we need only to consider generalization s' such that $head(s') \neq f$. This implies that $\mathsf{U} \in \mathsf{Ax}(head(s'))$.

If s' does not contain f, then $s' \preceq X$. Thus let us assume that $s' = g(s'_1, s'_2)$ where $\mathsf{U} \in \mathsf{Ax}(g)$ and without loss of generality $head(s'_1) = f$. This implies that $s'_2 \preceq \epsilon_g$ (note that s' is linear) and thus $s'_1 \preceq s'$. This reduction can be performed inductively thus showing that for any generalization s' with $head(s') \neq f$ there exists $s'' \in S^*$ such that $s' \preceq s''$.

c) Let us assume that $t_1 = f(w_1, w_2)$ and $t_2 = f(r_1, r_2)$, such that $\mathsf{U} \in \mathsf{Ax}(f)$. Then we can proceed in a similar fashion as in case b) by constructing S^*. Thus, any generalization s' of $X : t_1 \triangleq t_2$ such that $head(s') = f$ and $s' = f(d_1, d_2)$, where d_1 is a generalization of $w_1 \triangleq r_1$, d_2 a generalization of $w_2 \triangleq r_2$, is more general than some generalization of S^*. When $\mathsf{U} \in \mathsf{Ax}(head(s'))$ and some generalization s'' is a subterm of s' such that there exists $s^* \in S^*$ with $s'' \preceq s^*$, a similar approach can be taken as in the second half of case b).

d) Let us assume that $t_1 = f(w_1, \ldots, w_m)$ and $t_2 = g(r_1, \ldots, r_m)$, where either $\mathsf{U} \in \mathsf{Ax}(f)$ or $\mathsf{U} \in \mathsf{Ax}(g)$, or both. by a single application of Exp-U-L or Exp-U-R this case can be reduced to c).

□

5.2. *Linear Generalization Modulo* AU

When an associative function constant has a unit element, we can not simply combine associative decomposition and unit expansion rules. Such a combination would generalize, for instance $f(a, b)$ and $f(b, a)$ by $f(X, Y)$, but the lggs are $f(X, a, Y)$ and $f(X, b, Y)$. The problem is related to the fact that by A-decomposition (by the rules Dec-A-L and Dec-A-R), we can not obtain AUPs which retain the first argument of a term on one side and an arbitrary term from the arguments on the another side.

The problem can be solved by special rules for AU-decomposition, which are used for those

f's for which $Ax(f) = \{\mathsf{A}, \mathsf{U}\}$. However, for termination, we should make sure that they do not generate trivial AUPs of the form $Y(\vec{x}) : \epsilon_f \triangleq \epsilon_f$. This is what the condition about non-triviality of new AUPs requires in the conditions below:

Dec-AU-L: **Associative-Unit Decomposition Left**

$$\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow \\ \{Y_1(\vec{x}) : f(t_1, \ldots, t_k) \triangleq s_1,\ Y_2(\vec{x}) : f(t_{k+1}, \ldots, t_n) \triangleq f(s_2, \ldots, s_m)\} \cup A; \\ S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}), Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{A}, \mathsf{U}\}$, $n, m \geqslant 2$, $0 \leqslant k \leqslant n$, Y_1 and Y_2 are fresh variables of appropriate types, $f(t_0) = f(t_{n+1}) = \epsilon_f$, and the new AUPs are not trivial.

Dec-AU-R: **Associative-Unit Decomposition Right**

$$\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow \\ \{Y_1(\vec{x}) : t_1 \triangleq f(s_1, \ldots, s_k),\ Y_2(\vec{x}) : f(t_2, \ldots, t_n) \triangleq f(s_{k+1}, \ldots, s_m)\} \cup A; \\ S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}), Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{A}, \mathsf{U}\}$, $n, m \geqslant 2$, $0 \leqslant k \leqslant m$, Y_1 and Y_2 are fresh variables of appropriate types, $f(s_0) = f(s_{m+1}) = \epsilon_f$, and the new AUPs are not trivial.

Note the difference from Dec-A-L and Dec-A-R: k is allowed to reach the boundaries. It can become 0, n, or m.

Soundness of AU-decomposition rules are easy to see. As for termination, we may require that an application of a unit expansion rule is immediately followed by the application of an AU-decomposition rule. Since the latter does not generate a trivial AUP, the sizes of the new AUPs will be smaller than the one to which the unit expansion rule was applied, which implies termination.

Note that if we did not put the trivial AUP restriction condition in the AU-decomposition rules, we could get an infinite derivation of the form $\{X : f(a, b) \triangleq a\}; \varnothing; Id \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}R}} \{X : f(a, b) \triangleq f(a, \epsilon_f)\}; \varnothing; Id \Longrightarrow_{\mathsf{Dec\text{-}AU\text{-}L}} \{Y : f(a, b) \triangleq a,\ Z : \epsilon_f \triangleq \epsilon_f\}; \varnothing; \{X \mapsto f(Y, Z)\} \Longrightarrow \cdots$.

Hence, to compute linear generalizations modulo AU we extend $\mathcal{G}_{\mathsf{U\text{-}lin}}$ by AU-decomposition rules. We call this algorithm $\mathcal{G}_{\mathsf{AU\text{-}lin}}$. With the AU-decomposition rules we obtain all possible decompositions. The unit expansion rules allows one to transform AUPs with mismatching head symbols into AUPs with matching head symbols when at least one of the head symbols is an AU-symbol. Therefore, by $\mathcal{G}_{\mathsf{AU\text{-}lin}}$ we will never miss an existing linear lgg of two terms. To illustrate the use of the above extension of $\mathcal{G}_{\mathrm{base}}$, let us consider the following example where $Ax(f) = \{\mathsf{A}, \mathsf{U}\}$:

Example 6. Let $t = \lambda x.\lambda y.f(x, x, y, y)$ and $s = \lambda z.\lambda v.f(z, v, v)$. The initial state is $\{X : t \triangleq s\}; \varnothing; Id$ and the derivation of the lgg $\lambda x.\lambda y.f(x, Y(x, y), f(y, y))$ is as follows:

$$\{X : t \triangleq s\}; \varnothing; Id \Longrightarrow^{\times 2}_{\mathsf{Abs}} \\ \{X'(x, y) : f(x, x, y, y) \triangleq f(x, y, y)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x, y)\} \Longrightarrow^{k=2}_{\mathsf{Dec\text{-}AU\text{-}L}} \\ \{X_1(x, y) : f(x, x) \triangleq x,\ X_2(x, y) : f(y, y) \triangleq f(y, y)\}; \varnothing; \\ \quad \{X \mapsto \lambda x.\lambda y.f(X_1(x, y), X_2(x, y))\} \Longrightarrow^{\times 3}_{\mathsf{Dec}}$$

$$\{X_1(x,y) : f(x,x) \triangleq x\}; \varnothing; \{X \mapsto \lambda x.\lambda y.f(X_1(x,y), f(y,y))\} \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}R}}$$
$$\{X_1(x,y) : f(x,x) \triangleq f(x,\epsilon_f)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.f(X_1(x,y), f(y,y))\} \Longrightarrow^{k=1}_{\mathsf{Dec\text{-}AU\text{-}L}}$$
$$\{X_3(x,y) : x \triangleq x,\ X_4(x,y) : x \triangleq \epsilon_f\}; \varnothing;$$
$$\quad \{X \mapsto \lambda x.\lambda y.f(f(X_3(x,y), X_4(x,y)), f(y,y))\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{X_4(x,y) : x \triangleq \epsilon_f\}; \varnothing; \{X \mapsto \lambda x.\lambda y.f(f(x, X_4(x,y)), f(y,y))\} \Longrightarrow_{\mathsf{Sol}}$$
$$\varnothing; \{Y(x,y) : x \triangleq \epsilon_f\}; \{X \mapsto \lambda x.\lambda y.f(f(x, Y(x,y)), f(y,y))\}.$$

Notice that the AU-lin generalization $\lambda x.\lambda y.f(x, Y(x,y), y, y)$ computed here is less general than the A-generalizations $\lambda x.\lambda y.f(Y(x), y, y)$ and $\lambda x.\lambda y.f(x, Y(x,y), y)$ computed in Example 2.

5.3. *Linear Generalization Modulo* CU

Generalization in a commutative theory with the unit element is simpler than AU-generalization described above. The reason is in the Dec-C rule, which generates new AUPs from the arguments of the given AUP, removing the head symbol. The effect of its combination with the unit expansion rules is to anti-unify one argument from one side with the term in another side, while the other argument is anti-unified with the unit element. These are exactly all the alternatives CU-generalization should consider. For (linear) CU-generalization algorithm we can add to $\mathcal{G}_{\mathsf{U\text{-}lin}}$ the counterpart of Dec-C rule which is applied when $Ax(f) = \{\mathsf{C}, \mathsf{U}\}$, obtaining the algorithm $\mathcal{G}_{\mathsf{CU\text{-}lin}}$. Its soundness, termination, and completeness properties are straightforward. To illustrate the use of the above extension of $\mathcal{G}_{\mathrm{base}}$, let us consider the following example where $Ax(f) = \{\mathsf{C}, \mathsf{U}\}$:

Example 7. Let $t = \lambda x.\lambda y.f(g(x,y), g(y,x))$ and $s = \lambda z.\lambda v.g(z,z)$. The initial state is $\{X : t \triangleq s\}; \varnothing; Id$ and the derivation of the lgg $\lambda x.\lambda y.f(g(x, Y(x,y)), Z(x,y))$ is as follows:

$$\{X : t \triangleq s\}; \varnothing; Id \Longrightarrow^{\times 2}_{\mathsf{Abs}}$$
$$\{X'(x,y) : f(g(x,y), g(y,x)) \triangleq g(x,x)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x,y)\} \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}R}}$$
$$\{X'(x,y) : t \triangleq f(\epsilon_f, s)\}; \varnothing; \{X \mapsto \lambda x.\lambda y.X'(x,y), \ldots\} \Longrightarrow^{i=2}_{\mathsf{Dec\text{-}C}}$$
$$\{X_1(x,y) : g(x,y) \triangleq g(x,x),\ X_2(x,y) : g(y,x) \triangleq \epsilon_f\}; \varnothing;$$
$$\quad \{X \mapsto \lambda x.\lambda y.f(X_1(x,y), X_2(x,y)), \ldots\} \Longrightarrow^{\times 2}_{\mathsf{Dec}}$$
$$\{X_4(x,y) : y \triangleq x,\ X_2(x,y) : g(y,x) \triangleq \epsilon_f\}; \varnothing;$$
$$\quad \{X \mapsto \lambda x.\lambda y.f(g(x, X_4(x,y)), X_2(x,y)), \ldots\} \Longrightarrow_{\mathsf{Sol}}$$
$$\varnothing; \{Y(x,y) : y \triangleq x, Z(x,y) : g(y,x) \triangleq \epsilon_f\}; \{X \mapsto \lambda x.\lambda y.f(g(x, Y(x,y)), Z(x,y)), \ldots\}.$$

5.4. *Linear Generalization Modulo* ACU

ACU-lin-generalization is characterized by the properties of both AU-lin- and CU-lin- generalizations. From AU-lin, it should inherit the condition that new AUPs are not trivial. It is similar to CU-lin in that the original decomposition does not need to be changed: since the order of arguments is not fixed, there is no problem in reaching the boundaries in the combination with the unit expansion rules (which was problematic in the A case). Therefore, ACU-decomposition rules have the following form:

Dec-ACU-L: **Associative-Commutative-Unit Decomposition Left**

$$\{X(\vec{x}) : f(t_1,\ldots,t_n) \triangleq f(s_1,\ldots,s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1(\vec{x}) : f(t_{i_1},\ldots,t_{i_l}) \triangleq s_k,$$
$$Y_2(\vec{x}) : f(t_{i_{(l+1)}},\ldots,t_{i_n}) \triangleq f(s_1,\ldots,s_{k-1},s_{k+1},\ldots,s_m)\} \cup A;$$
$$S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}),Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{A},\mathsf{C},\mathsf{U}\}$, $\{i_1,\ldots,i_n\} \equiv \{1,\ldots,n\}$, $l \in \{1,\ldots,n-1\}$, $k \in \{1,\ldots,m\}$, $n,m \geqslant 2$, Y_1 and Y_2 are fresh variables of appropriate types, and the new AUPs are not trivial.

Dec-ACU-R: **Associative-Commutative-Unit Decomposition Right**

$$\{X(\vec{x}) : f(t_1,\ldots,t_n) \triangleq f(s_1,\ldots,s_m)\} \uplus A;\ S;\ \sigma \Longrightarrow$$
$$\{Y_1(\vec{x}) : t_k \triangleq f(s_{i_1},\ldots,s_{i_l}),$$
$$Y_2(\vec{x}) : f(t_1,\ldots,t_{k-1},t_{k+1},\ldots,t_n) \triangleq f(s_{i_{(l+1)}},\ldots,s_{i_m})\} \cup A;$$
$$S;\ \sigma\{X \mapsto \lambda\vec{x}.f(Y_1(\vec{x}),Y_2(\vec{x}))\},$$

where $Ax(f) = \{\mathsf{A},\mathsf{C},\mathsf{U}\}$, $\{i_1,\ldots,i_m\} \equiv \{1,\ldots,m\}$, $l \in \{1,\ldots,m-1\}$, $k \in \{1,\ldots,n\}$, $n,m \geqslant 2$, Y_1 and Y_2 are fresh variables of appropriate types, and the new AUPs are not trivial.

Extending $\mathcal{G}_{\mathsf{U\text{-}lin}}$ with Dec-ACU-L and Dec-ACU-R, we obtain an algorithm for linear ACU-generalization which we call $\mathcal{G}_{\mathsf{ACU\text{-}lin}}$. Its soundness is straightforward. Arguments for termination are similar to those for AU. For completeness, note that we essentially consider all decompositions with all permutations of arguments under ACU symbols, and the unit expansion rules introduce a unit element allowing comparison between the unit element and all terms occurring as arguments to an ACU symbols. Therefore, no linear lgg will be missed. To illustrate the use of the above extension of $\mathcal{G}_{\text{base}}$, let us consider the following example where $Ax(f) = \{\mathsf{A},\mathsf{C},\mathsf{U}\}$:

Example 8. Let $t = \lambda x.\lambda y.\lambda z.f(x,z,y)$ and $s = \lambda x'.\lambda y'.\lambda z'.f(y',z',x',g(x'))$. The initial state is $\{X : t \triangleq s\};\varnothing;Id$ and the derivation of the lgg $\lambda x.\lambda y.\lambda z.f(x,y,z,Y(x,y,z))$ is as follows:

$$\{X : t \triangleq s\};\varnothing;Id \Longrightarrow^{\times 3}_{\mathsf{Abs}}$$
$$\{X'(x,y,z) : f(x,z,y) \triangleq f(y,z,x,g(x))\};\varnothing;\{X \mapsto \lambda x.\lambda y.\lambda z.X'(x,y,z)\} \Longrightarrow^{l=1,k=3}_{\mathsf{Dec\text{-}AC\text{-}L}}$$
$$\{X_1(x,y,z) : x \triangleq x,\ X_2(x,y,z) : f(z,y) \triangleq f(y,z,g(x))\};\varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.\lambda z.f(X_1(x,y,z),X_2(x,y,z)),\ldots\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{X_2(x,y,z) : f(z,y) \triangleq f(y,z,g(x))\};\varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.\lambda z.f(x,X_2(x,y,z)),\ldots\} \Longrightarrow^{l=1,k=2}_{\mathsf{Dec\text{-}AC\text{-}R}}$$
$$\{X_3(x,y,z) : y \triangleq y,\ X_4(x,y,z) : z \triangleq f(z,g(x))\};\varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.\lambda z.f(x,X_3(x,y,z),X_4(x,y,z)),\ldots\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{X_4(x,y,z) : z \triangleq f(z,g(x))\};\varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.\lambda z.f(x,y,X_4(x,y,z)),\ldots\} \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}L}}$$
$$\{X_4(x,y,z) : f(z,\epsilon_f) \triangleq f(z,g(x))\};\varnothing;$$
$$\{X \mapsto \lambda x.\lambda y.\lambda z.f(x,y,X_4(x,y,z)),\ldots\} \Longrightarrow^{\times 2}_{\mathsf{Dec}}$$
$$\{X_5(x,y,z) : \epsilon_f \triangleq g(x)\};\varnothing;\{X \mapsto \lambda x.\lambda y.\lambda z.f(x,y,z,X_5(x,y,z)),\ldots\} \Longrightarrow_{\mathsf{Sol}}$$
$$\varnothing;\{Y(x,y,z) : \epsilon_f \triangleq g(x)\};\{X \mapsto \lambda x.\lambda y.\lambda z.f(x,y,z,Y(x,y,z)),\ldots\}.$$

5.5. *Combining Different Theories*

Finally, we consider the general case when different function constants satisfy associativity and/or commutativity and/or identity axioms. Like in [Alpuente et al., 2014], we can use the rules above all together. (In the presence of the unit element, we can restrict ourselves with computing linear generalizations) All rules, except DH-U, are local in the sense of [Alpuente et al., 2014]: they are local to the given top function constant in the given AUP they are acting upon, irrespective of what other constants and what other axioms may be present in the given alphabet and the equational theory. Such a locality means that the rules are modular and they do not change when new A and/or C and/or U constants are introduced.

It should be also mentioned that in the algorithms considered above minimality was not our goal. The obtained complete set of generalizations are not necessarily minimal. They can be minimized later. Also, the rules have not been optimized and in general, nondeterminism can be high. In the rest of the paper, we will try to reduce it, aiming at computing some kind of optimal solutions.

6. Towards Special Restrictions

This section is devoted to computing special kind of "optimal" generalizations, which can be done more efficiently than the general unrestricted cases considered in the previous section.

The idea is the following: The equational decomposition rules introduce branching in the search space. Each branch can be developed in linear time, but there can be too many of them. However, if the branching factor is bounded, we could choose one of the alternative states (produced by decomposition) based on some "optimality" criterion, and develop only that branch. Such a greedy approach will give one "optimal" generalization.

While restricting the input terms enough will guarantee the production of generalizations in linear time, these generalizations are not necessarily useful, meaningful, or relevant. For example, we can allow function symbols with an equational theory but only consider terms which do not need the equation decomposition rules. In order to guarantee relevance we want to find the least restrictive restrictions of the input terms which have some guaranteed complexity bounds and produce generalizations which are less general than the syntactic counterpart. In this sense, the restrictions given in the following sections are useful, meaningful, or relevant.

In order to have a "reasonable" computational complexity, we should be able to choose such an optimal state from "reasonably" many alternatives in "reasonable" time. Towards this goal, we start by introducing the concept of $\mathcal{E}$-refined generalizations. They will be our main target to compute.

Definition 1 ($\mathcal{E}$-refined generalization). Given two terms t and s and their $\mathcal{E}$-generalizations r and r', we say that r is *at least as good as* r' with respect to $\mathcal{E}$ if either $r' \preceq_{\mathcal{E}} r$ or they are not comparable with respect to $\preceq_{\mathcal{E}}$.

An $\mathcal{E}$-generalization r of t and s is called their *$\mathcal{E}$-refined generalization* iff r is at least as good (with respect to $\mathcal{E}$) as a syntactic lgg of t and s.

In our equational theories, to obtain a syntactic generalization of two terms we assume that all occurrences of associative symbols in the terms are associated to the right.

Example 9. Let $t = f(a, b, c, c, a)$ and $s = f(d, d, b, b, c, a)$ be two terms, where f is associative. Their syntactic lgg (with f being right associated) is $r' = f(x, f(y, f(z, f(z, u)))$, where x is a generalization of a and d, y of b and d, z of c and b, and u of a and $f(c, a)$.

An A-refined generalization of t and s is $r = f(x, b, y, c, a)$, where x generalizes a and $f(d, d)$ and y generalizes c and b. Note that r is as good as r': they are incomparable with respect to $\preceq_{\mathsf{A}}$.

We design the following general procedure to compute $\mathcal{E}$-refined generalizations:

Refine A procedure to compute $\mathcal{E}$-refined generalizations

1: Let $A; S; \sigma$ be a state containing an AUP P upon which equational decomposition rules can be applied in n different ways.
2: From $A; S; \sigma$ generate n new states $A_1; S_1; \sigma_1$ through $A_n; S_n; \sigma_n$ which result from the various ways equational decomposition rules may be applied to P.
3: Now find an lgg, denoted by l_i, for each $A_i; S_i; \sigma_i$ using $\mathcal{G}_{\text{base}}$.
4: Select the least general l_i (or one by some heuristics when multiple generalizations are least general) and choose $A_i; S_i; \sigma_i$ as the successor state of $A; S; \sigma$.
5: If $A_i; S_i; \sigma_i$ contains an AUP which may be equationally decomposed, then go back to 1. Otherwise, apply a rule from $\mathcal{G}_{\text{base}}$ if possible, and repeat this line. If no rule applies, then we exit.

Note that every syntactic generalization is also an $\mathcal{E}$-refined generalization. Therefore, to guarantee that **Refine** indeed computes $\mathcal{E}$-refined generalizations, we need to make sure that in step 2, equational decomposition rules produce results among which at least one is as good as the result of syntactic decomposition. Another consequence of the definition of $\mathcal{E}$-refined generalization is that every element of the minimal complete set of $\mathcal{E}$-generalizations (in our equational theories) of two terms is an $\mathcal{E}$-refined generalization of t and s. However, there might exist $\mathcal{E}$-refined generalizations which do not belong to the minimal complete set of generalizations.

Example 10. Let us consider a simple first-order example of this property, namely, the AUP $f(a, f(b, c)) \triangleq f(c, f(a, b))$ where $Ax(f) = \{\mathsf{A}, \mathsf{C}\}$. Note that the minimal complete set of AC-generalizations for this AUP contains one element, $f(a, f(b, c))$. Its syntactic generalization is $f(x, f(y, z))$ where x, y, and z are fresh variables. An example of AC-refined generalization, which is not in the minimal complete set of AC-generalizations, is $f(a, f(y, z))$.

Looking back at the description of the **Refine** procedure, we can say that at each branching point we will be aiming at choosing the alternative that would lead to "the best" $\mathcal{E}$-refined generalization. To limit the number of choices, we will need to identify restrictions of equational anti-unification problems which would have a constant decomposition branching factor.

The concept of $\mathcal{E}$-refined generalizations allows us to compute better generalizations than the base procedure would do, without concerning ourselves with certain difficulty to handle decompositions. We will outline what we mean by "difficult" in later sections. Some of these difficult decompositions can be handled by finding *alignments* between two sequences of symbols. These sequences are usually extracted as sequences of *root* symbols from the given AUPs, where the root of a term is defined as $root(\lambda x_1, \ldots, x_k.t) = \lambda x_1, \ldots, x_k$ if $k > 0$, and

$root(\lambda x_1, \ldots, x_k.t) = head(t)$ if $k = 0$. Note that as a root, we treat $\lambda x_1, \ldots, x_k$ as *one symbol* and it is identified with any lambda prefix with k variables, e.g., $\lambda x_1, \ldots, x_k = \lambda y_1, \ldots, y_k$.

Definition 2 (Pairs of argument root sequences). Let $t = \lambda x_1, \ldots, x_k.\mathsf{h}(t_1, \ldots, t_n)$ and $s = \lambda x_1, \ldots, x_k.\mathsf{h}(s_1, \ldots, s_m)$, where $k \geqslant 0$, be terms with the same type and the same head h (which can be a constant, free variable, or a bound variable). Then the *pair of argument root sequences* of t and s, denoted as $pars(t, s)$, is defined as follows:

$$pars(t, s) = \langle (root(t_1), \ldots, root(t_n)),\ (root(s_1), \ldots, root(s_m)) \rangle .$$

This notion extends to AUPs: A pair of argument root sequences of an AUP $X(\vec{x}) : t \triangleq s$ is the pair of argument root sequences of the terms t and s.

Definition 3 (Alignment, rigidity function). Let w_1 and w_2 be sequences of symbols. Then the sequence $a_1[i_1, j_1] \cdots a_n[i_n, j_n]$, for $n \geqslant 0$ is an *alignment* of w_1 and w_2 if

— i's and j's are integers such that $0 < i_1 < \cdots < i_n \leqslant |w_1|$ and $0 < j_1 < \cdots < j_n \leqslant |w_2|$, and

— $a_k = w_1|_{i_k} = w_2|_{j_k}$, for all $1 \leqslant k \leqslant n$.

The set of all alignments will be denoted by **A**. A *rigidity function* $\mathcal{R}$ is a function that returns, for every pair of sequences of symbols w_1 and w_2, a set of alignments of w_1 and w_2.

The main intuition behind the use of rigidity functions for generalization is to capture the structure (modulo a given rigidity property) of as many nonvariable terms as possible [Kutsia et al., 2014].

Example 11. Let us consider the two sequences of symbols (b, a, b, a) and (a, b, c, b, a). The following alignments are singleton alignments: $a[2, 1]$, $b[1, 2]$, and $b[3, 4]$. A larger alignment would be $a[2, 1]b[3, 2]a[4, 5]$. Notice that this alignment also happens to be one of the longest sequences of common symbols (longest common subsequence, LCS). The other LCS is $b[1, 2]b[3, 4]a[4, 5]$. We can define a rigidity function that returns, for instance, the set of all LCS's, or the set of all singleton alignments, or something more specific: a rigidity function $\mathcal{R}_{LCS}$ for the given two symbol sequences selects the LCS alignment, which is lexicographically smallest (with respect to the positions) among all such alignments. For example, $b[1, 2]b[3, 4]a[4, 5]$ is lexicographically smaller than $a[2, 1]b[3, 2]a[4, 5]$, because $(1, 2)$ is lexicographically smaller than $(2, 1)$.

$$\mathcal{R}_{LCS}((c, b, a, b, a), (a, b, c, b, a)) = \{c[1, 3]b[2, 4]a[3, 5]\}.$$
$$\mathcal{R}_{LCS}((a, b, c, b, a), (c, b, a, b, a)) = \{a[1, 3]b[2, 4]a[5, 5]\}.$$
$$\mathcal{R}_{LCS}((b, a, b, a), (a, b, c, b, a)) = \{b[1, 2]b[3, 4]a[4, 5]\}.$$

There is a subset of AUPs, referred to as 1-*determined AUPs*, which have interesting $\mathcal{E}$-refined generalizations computable in linear time. The number 1 means that the equational decomposition can be done only in one possible way. Hence, there is no branching in equational anti-unification rule applications. ($n = 1$ in step 2 of **Refine**.) The more general k-determined AUPs allow a bounded number of possible choices, that is k choices, whenever equational decomposition may be applied. ($n \leqslant k$ in step 2 of **Refine**.) Even for 2-determined AUPs computing the set of lggs is of exponential complexity. Therefore, we introduce the notion of $(\mathcal{R}, C, \mathcal{G})$-*optimal generalization* where $\mathcal{R}$ is rigidity function, C is a choice function picking one of the available decompositions, and $\mathcal{G}$ is the particular algorithm for which we are defining optimality. Under such optimality

conditions, we will see later that we are able to compute an $\mathcal{E}$-refined generalization in quadratic time for (uniformly) k-determined AUPs and in cubic time for arbitrary AUPs with associative constants.

The equational decomposition rules above are too non-deterministic and the computed set of generalizations has to be minimized to obtain minimal complete sets of generalizations. However, even if we performed more guided decompositions, obtaining e.g., terms with the same root in new AUPs (as in [Kutsia et al., 2014]), there would still be alternatives. For instance, consider the following AUP where f is associative: $X(\vec{x}) : f(t_1, \dots t_i, \dots, t_j, \dots, t_n) \triangleq f(s_1, \dots s_i, \dots, s_j, \dots, s_m)$. Now let $root(t_i) = root(s_j)$, $root(s_i) = root(t_j)$, and for every other term comparison whose index is $\leqslant j$ the root symbols are not equivalent. An example of such a situation as two sequences of root symbols would be $(c, c, c, a, c, b, c, c, c)$ and $(d, d, d, b, d, a, d, d, d)$. This situation results in two singleton alignments, $a[3, 6]$ and $b[6, 3]$. Note that any application of associative decomposition will have to choose between these alignments, i.e. choosing one gives two AUPs in which the other alignment does not appear. It is not clear from the available information which choice will lead to a less general generalization. While this situation illustrates what happens when there are two alignments to choose from, it can easily be generalized to k different possible alignments, for example $(c, c, c, a, c, b, c, c, c)$ and $(d, d, d, b, a, a, d, d, d)$ which contains the singleton alignments $a[3, 6]$, $a[3, 5]$, and $b[6, 3]$.

Definition 4 (Maximal alignment). An alignment $\mathfrak{a} = a_1[i_1, j_1] \cdots a_n[i_n, j_n]$ is called an *extension* of an alignment $\mathfrak{b}$, if $\mathfrak{b}$ is obtained from $\mathfrak{a}$ by removing some letters $a_{k_1}[i_{k_1}, j_{k_1}], \dots, a_{k_r}[i_{k_r}, j_{k_r}]$, $\{k_1, \dots, k_r\} \subseteq \{1, \dots, n\}$. It is a *proper extension* if $r > 0$.

An alignment $\mathfrak{a}$ is a *maximal alignment* of two symbol sequences w_1 and w_2, if no proper extension of $\mathfrak{a}$ is an alignment of w_1 and w_2. The *set of maximal nonempty alignments* of w_1 and w_2 is denoted by $\mathsf{max\text{-}ne\text{-}align}(w_1, w_2)$.

Example 12. The sequences (a, b, a) and (c, a, b, c) have two maximal alignments: $a[1, 2]b[2, 3]$ and $a[3, 2]$. The first one is even the longest common subsequence of the given sequences, while the second one is not.

Definition 5 (k-determined sequence pair). A pair of sequences of symbols $\langle w_1, w_2 \rangle$ is called *k-determined*, if $\mathsf{max\text{-}ne\text{-}align}(w_1, w_2)$ contains at most k elements.

Obviously, $\langle w_1, w_2 \rangle$ is k-determined iff $\langle w_2, w_1 \rangle$ is k-determined. The definition implies that any k-determined pair of sequences is also n-determined for any $n \geqslant k$.

Example 13. We illustrate the definition of k-determined pairs with the examples below:

1. $\langle (a, b, c), (a, d, b, c) \rangle$ is 1-determined, because

   $$\mathsf{max\text{-}ne\text{-}align}((a, b, c), (a, d, b, c)) = \{a[1, 1]b[2, 3]c[3, 4]\}.$$

2. $\langle (a, b, a), (c, a, b, c) \rangle$ is 2-determined, because

   $$\mathsf{max\text{-}ne\text{-}align}((a, b, a), (c, a, b, c)) = \{a[1, 2]b[2, 3],\ a[3, 2]\}.$$

3. $\langle (a, c, c, b, a, c), (a, d, b, a, c) \rangle$ is 3-determined, because

   $$\mathsf{max\text{-}ne\text{-}align}((a, c, c, b, a, c), (a, d, b, a, c)) =$$

$$\{a[1,1]c[2,5],\ a[1,1]c[3,5],\ a[1,1]b[4,3]a[5,4]c[6,5]\}.$$

4. $\langle (a,b),(b,a)\rangle$ is 2-determined, because

$$\mathsf{max\text{-}ne\text{-}align}((a,b),(b,a)) = \{a[1,2],\ b[2,1]\}.$$

5. $\langle (a,a),(a,a)\rangle$ is 1-determined, because

$$\mathsf{max\text{-}ne\text{-}align}((a,a),(a,a)) = \{a[1,1]a[2,2]\}.$$

6. $\langle (a,b),(c,d,e)\rangle$ is 0-determined, because

$$\mathsf{max\text{-}ne\text{-}align}((a,b),(c,d,e)) = \varnothing.$$

Definition 6 (Uniform and max-uniform alignments). Let $\mathfrak{a} = a_1[i_1,j_1]\cdots a_n[i_n,j_n]$ be an alignment of two symbol sequences $w_1 = (l_1,\ldots,l_m)$ and $w_2 = (r_1,\ldots,r_k)$. We say that $\mathfrak{a}$ is a *uniform alignment* of w_1 and w_2, if the following conditions are satisfied:

(1) $i_1 = 1$ iff $j_1 = 1$,
(2) $i_n = m$ iff $j_n = k$,
(3) for all $1 < q \leqslant n$, we have $i_q - i_{q-1} = 1$ iff $j_q - j_{q-1} = 1$.

A uniform alignment $\mathfrak{a}$ of w_1 and w_2 is their *maximal uniform alignment* (shortly, *max-uniform alignment*), if no proper extension of $\mathfrak{a}$ is a uniform alignment of w_1 and w_2. The *set of all nonempty max-uniform alignments* of w_1 and w_2 is denoted by $\mathsf{max\text{-}unif\text{-}ne\text{-}align}(w_1,w_2)$.

The first condition of uniformity forbids the first element in one sequence to be aligned with a non-first element in another sequence. The second condition is dual to the first one, putting the similar requirement on the last elements in the given sequences. The third condition guarantees that consecutive elements in w_1 are aligned with consecutive elements of w_2 and vice versa.

The empty alignment is always uniform. It is the trivial uniform alignment.

Example 14. The sequences (a,b,a) and (c,a,b,c) from Example 12 have two (nontrivial) uniform alignments: $b[2,3]$ and $a[3,2]$. They are also max-uniform alignments. Note that a maximal alignment $a[1,2]b[2,3]$ is not uniform, because the first condition of uniformity is violated.

The sequences (a,b,c) and (a,d,b,c) have five (nontrivial) uniform alignments $a[1,1]$, $b[2,3]$, $c[3,4]$, $a[1,1]c[3,4]$, and $b[2,3]c[3,4]$. Note that $a[1,1]b[2,3]c[3,4]$ is a maximal but non-uniform alignment, because the third condition of uniformity is violated. The max-uniform alignments are $a[1,1]c[3,4]$ and $b[2,3]c[3,4]$.

Definition 7 (Uniformly k-determined sequence pair). The pair of sequences of symbols $\langle w_1,w_2\rangle$ is called *uniformly k-determined*, if $\mathsf{max\text{-}unif\text{-}ne\text{-}align}(w_1,w_2)$ contains at most k elements.

Example 15. We illustrate uniformly k-determined pairs with the examples below and compare them with k-determined pairs from Example 13:

1. $\langle (a,b,c),(a,d,b,c)\rangle$ is uniformly 2-determined (1-determined in Example 13), because

$$\mathsf{max\text{-}unif\text{-}ne\text{-}align}((a,b,c),(a,d,b,c)) = \{a[1,1]c[3,4],\ b[2,3]c[3,4]\}.$$

2. $\langle (a,b,a),(c,a,b,c)\rangle$ is uniformly 1-determined (2-determined in Example 13), because

   $\mathsf{max\text{-}unif\text{-}ne\text{-}align}((a,b,a),(c,a,b,c)) = \{b[3,2]\}$.

3. $\langle (a,c,c,b,a,c),(a,d,b,a,c)\rangle$ is uniformly 1-determined (3-determined in Example 13), because

   $\mathsf{max\text{-}unif\text{-}ne\text{-}align}((a,c,c,b,a,c),(a,d,b,a,c)) = \{a[1,1]b[4,3]a[5,4]c[6,5]\}$.

4. $\langle (a,b),(b,a)\rangle$ is uniformly 0-determined (2-determined in Example 13), because

   $\mathsf{max\text{-}unif\text{-}ne\text{-}align}((a,b),(b,a)) = \varnothing$.

5. $\langle (a,a),(a,a)\rangle$ is uniformly 1-determined (1-determined in Example 13), because

   $\mathsf{max\text{-}unif\text{-}ne\text{-}align}((a,a),(a,a)) = \{a[1,1]a[2,2]\}$.

6. $\langle (a,b),(c,d,e)\rangle$ is uniformly 0-determined (1-determined in Example 13), because

   $\mathsf{max\text{-}unif\text{-}ne\text{-}align}((a,b),(c,d,e)) = \varnothing$.

We will need also *orderless* counterparts of definitions 4-7. They deal with alignments in which the order of symbols does not matter and, thus, can be considered as multisets.

Definition 8 (Orderless alignment, orderless rigidity function). Let w_1 and w_2 be sequences of symbols. Then the multiset $\{\!\{a_1[i_1,j_1],\ldots,a_n[i_n,j_n]\}\!\}$, for $n \geqslant 0$, is an *orderless alignment* of w_1 and w_2 if

— i's and j's are integers such that $0 < i_k \leqslant |w_1|$ and $0 < j_k \leqslant |w_2|$ for all $1 \leqslant k \leqslant n$,
— $i_k \neq i_l$ and $j_k \neq j_l$ for all $k \neq l$, and
— $a_k = w_1|_{i_k} = w_2|_{j_k}$, for all $1 \leqslant k \leqslant n$.

An *orderless rigidity function* $\mathcal{R}_{\mathsf{O}}$ returns for every pair of sequences of symbols a set of their orderless alignments.

Example 16. Let $w_1 = (a,b,a)$ and $w_2 = (c,a,b,c)$. The set of orderless alignments of w_1 and w_2 is $\{\{\!\{\}\!\}, \{\!\{a[1,2]\}\!\}, \{\!\{a[3,2]\}\!\}, \{\!\{b[2,3]\}\!\}, \{\!\{a[1,2],b[2,3]\}\!\}, \{\!\{a[3,2],b[2,3]\}\!\}\}$. If $\mathcal{R}_{\mathsf{O}}$ computes longest orderless alignments, then it will give the set $\{\{\!\{a[1,2],b[2,3]\}\!\}, \{\!\{a[3,2],b[2,3]\}\!\}\}$.

The notions of alignment extension and maximal alignment easily extend to orderless alignments:

Definition 9 (Maximal orderless alignment). Let $\mathfrak{a}$ and $\mathfrak{b}$ be two orderless alignments. We say that $\mathfrak{a}$ is an *extension* of $\mathfrak{b}$ if $\mathfrak{b} \subseteq \mathfrak{a}$, where $\subseteq$ is multiset inclusion. It is a *proper extension* if $\mathfrak{b} \subset \mathfrak{a}$.

An alignment $\mathfrak{a}$ is a *maximal orderless alignment* of two symbol sequences w_1 and w_2, if no proper extension of $\mathfrak{a}$ is an orderless alignment of w_1 and w_2. The *set of all maximal orderless alignments* of w_1 and w_2 is denoted by $\mathsf{oless\text{-}max\text{-}ne\text{-}align}(w_1,w_2)$.

Obviously, maximal orderless alignments of w_1 and w_2 consist of symbols from the intersection of multisets of symbols from w_1 and w_2.

Definition 10 (O-k-determined sequence pair). A pair of sequences of symbols $\langle w_1,w_2\rangle$ is called O-k-*determined*, if $\mathsf{oless\text{-}max\text{-}ne\text{-}align}(w_1,w_2)$ contains at most k elements.

Example 17. This example illustrates the notion of O-k-determinedness. Note that the first six pairs below were also used in Example 13.

1. $\langle (a, b, c), (a, d, b, c)\rangle$ is O-1-determined, because

   $$\mathsf{oless\text{-}max\text{-}ne\text{-}align}((a, b, c), (a, d, b, c)) = \{\{\!\{a[1, 1], b[2, 3], c[3, 4]\}\!\}\}.$$

2. $\langle (a, b, a), (c, a, b, c)\rangle$ is O-2-determined, because

   $$\mathsf{oless\text{-}max\text{-}ne\text{-}align}((a, b, a), (c, a, b, c)) = \{\{\!\{a[1, 2], b[2, 3]\}\!\}, \{\!\{a[3, 2], b[2, 3]\}\!\}\}.$$

3. $\langle (a, c, c, b, a, c), (a, d, b, a, c)\rangle$ is O-6-determined, because

   $$\begin{aligned}\mathsf{oless\text{-}max\text{-}ne\text{-}align}&((a, c, c, b, a, c), (a, d, b, a, c)) = \\ &\{\{\!\{a[1, 1], a[5, 4], b[4, 3], c[2, 5]\}\!\},\ \{\!\{a[1, 4], a[5, 1], b[4, 3], c[2, 5]\}\!\},\\ &\{\{\!\{a[1, 1], a[5, 4], b[4, 3], c[6, 5]\}\!\},\ \{\!\{a[1, 4], a[5, 1], b[4, 3], c[3, 5]\}\!\},\\ &\{\{\!\{a[1, 1], a[5, 4], b[4, 3], c[3, 5]\}\!\},\ \{\!\{a[1, 4], a[5, 1], b[4, 3], c[6, 5]\}\!\}\}.\end{aligned}$$

4. $\langle (a, b), (b, a)\rangle$ is O-1-determined, because

   $$\mathsf{oless\text{-}max\text{-}ne\text{-}align}((a, b), (b, a)) = \{\{\!\{a[1, 2], b[2, 1]\}\!\}\}.$$

5. $\langle (a, a), (a, a)\rangle$ is O-2-determined, because

   $$\mathsf{oless\text{-}max\text{-}ne\text{-}align}((a, a), (a, a)) = \{\{\!\{a[1, 1], a[2, 2]\}\!\},\ \{\!\{a[1, 2], a[2, 1]\}\!\}\}.$$

6. $\langle (a, b), (c, d, e)\rangle$ is 0-determined, because

   $$\mathsf{max\text{-}ne\text{-}align}((a, b), (c, d, e)) = \varnothing.$$

7. $\langle (a, b), (c, a)\rangle$ is O-1-determined, because

   $$\mathsf{oless\text{-}max\text{-}ne\text{-}align}((a, b), (c, a)) = \{\{\!\{a[1, 2]\}\!\}\}.$$

8. $\langle (a, b), (a, a)\rangle$ is O-2-determined, because

   $$\mathsf{oless\text{-}max\text{-}ne\text{-}align}((a, b), (a, a)) = \{\{\!\{a[1, 1]\}\!\},\ \{\!\{a[1, 2]\}\!\}\}.$$

One can see that the elements in $\mathsf{oless\text{-}max\text{-}ne\text{-}align}(w_1, w_2)$ differ from each other by different positions of the same symbols in the input sequences. As symbol multisets, they are the same.

All 1-determined sequence pairs have the feature: if the sequences contain a common element, then it appears only once in each sequence. In O-1-determined case it means that the intersection of the sequences considered as multisets is, in fact, a set.

Definition 11 (Uniform and max-uniform orderless alignments). Let $\mathfrak{a} = \{\!\{a_1[i_1, j_1], \ldots, a_n[i_n, j_n]\}\!\}$ be an orderless alignment of two symbol sequences $w_1 = (l_1, \ldots, l_m)$ and $w_2 = (r_1, \ldots, r_k)$. We say that $\mathfrak{a}$ is a *uniform orderless alignment* of w_1 and w_2, if

$$\{\!\{l_1, \ldots, l_m\}\!\} = \{\!\{a_1, \ldots, a_n\}\!\} \text{ iff } \{\!\{r_1, \ldots, r_k\}\!\} = \{\!\{a_1, \ldots, a_n\}\!\}.$$

A uniform orderless alignment $\mathfrak{a}$ of w_1 and w_2 is their *maximal uniform orderless alignment* (shortly, *max-uniform orderless alignment*), if no proper extension of $\mathfrak{a}$ is a uniform orderless alignment of w_1 and w_2. The *set of all max-uniform orderless alignments* of w_1 and w_2 is denoted by $\mathsf{oless\text{-}max\text{-}unif\text{-}ne\text{-}align}(w_1, w_2)$.

Similarly to uniform alignments, uniform orderless alignments prevent "misalignments" of the type empty-vs-nonempty sequences in w_1 and w_2.

Definition 12 (Uniformly O-k-determined sequence pair). A pair of sequences of symbols $\langle w_1, w_2 \rangle$ is called *uniformly* O-k-*determined*, if oless-max-unif-ne-align(w_1, w_2) contains at most k elements.

Example 18. All sequence pairs that were O-k-determined in Example 17 (except the first pair) are also *uniformly* O-k-determined for the same k. As for the first pair, we have the following:

$\langle (a,b,c), (a,d,b,c) \rangle$ is uniformly O-3-determined (O-1-determined in Example 17), because

$$\text{oless-max-unif-ne-align}((a,b,c),(a,d,b,c)) = \\ \{\{\!\{a[1,1], b[2,3]\}\!\},\ \{\!\{a[1,1], c[3,4]\}\!\},\ \{\!\{b[2,3], c[3,4]\}\!\}\}.$$

Next, from symbol sequence pairs we move to term pairs and define the notions of k-determinedness and uniformly k-determinedness for them. Note that these definitions accommodate the corresponding orderless cases as well.

Definition 13 (k-determined and uniformly k-determined term pair). A pair of terms $\langle t, s \rangle$ of the same type is (*uniformly*) k-*determined* iff either

— $head(t) \neq head(s)$, or
— $head(t) = head(s)$ and $Ax(head(t)) = \varnothing$, or
— $head(t) = head(s) = f$, $\varnothing \neq Ax(f) \subseteq \{\mathsf{A}, \mathsf{U}\}$, and $pars(t,s)$ is (uniformly) k-determined, or
— $head(t) = head(s) = f$, $\mathsf{C} \in Ax(f)$, and $pars(t,s)$ is (uniformly) O-k-determined.

Finally, we formulate the main definition of this section, defining two notions: total k-determined and total uniformly k-determined term pairs. They will play an important role in characterizing special cases of equational higher-order generalization.

Definition 14 (Total k-determined and total uniformly k-determined term pair). A pair of terms $\langle t, s \rangle$ is *total* k-*determined* (resp. *total uniformly* k-*determined*) if $\langle t, s \rangle$ is k-*determined* (resp. *uniformly* k-*determined*), and

— if $t = \lambda\vec{x}.\mathsf{h}(t_1, \ldots, t_n)$, $s = \lambda\vec{x}.\mathsf{h}(s_1, \ldots, s_m)$ where $Ax(\mathsf{h}) \subseteq \{\mathsf{A}, \mathsf{U}\}$, then for each $\mathfrak{a} \in$ max-ne-align$(pars(t,s))$ (resp. for each $\mathfrak{a} \in$ max-unif-ne-align$(pars(t,s))$) and each $a[i,j] \in \mathfrak{a}$, the pair $\langle t_i, s_j \rangle$ is total k-determined (resp. total uniformly k-determined),
— if $t = \lambda\vec{x}.\mathsf{h}(t_1, \ldots, t_n)$, $s = \lambda\vec{x}.\mathsf{h}(s_1, \ldots, s_m)$ where $\mathsf{C} \in Ax(\mathsf{h})$, then for each $\mathfrak{a} \in$ oless-max-ne-align$(pars(t,s))$ (resp. for each $\mathfrak{a} \in$ oless-max-unif-ne-align$(pars(t,s))$) and each $a[i,j] \in \mathfrak{a}$, the pair $\langle t_i, s_j \rangle$ is total k-determined (resp. total uniformly k-determined).

We say that an AUP $X(\vec{x}) : t \triangleq s$ is total k-determined (resp. total uniformly k-determined) if the term pair $\langle t, s \rangle$ is total k-determined (resp. total uniformly k-determined).

As one can see from Definition 14, the first item concerns four theories without commutativity ($\varnothing$, A, U, AU), and the second one to other four theories with commutativity (C, AC, CU, ACU).

Proposition 1. For a given constant k, the complexity of checking if the term pair $\langle t, s \rangle$ is

(uniformly) k-determined is $O(n^2)$ and total (uniformly) k-determined is $O(k^n n^2)$, where n is maximum of the lengths of t and s.

Proof. Checking whether a pair of terms $\langle t, s\rangle$ is k-determined requires computing the set $\mathsf{max\text{-}ne\text{-}align}(pahs(t,s))$ which in worst case requires $O(n^2)$ time. When checking total (uniformly) k-determinedness we need to repeat this computation recursively over the term pairs resulting from the alignments of $\mathsf{max\text{-}ne\text{-}align}(pahs(t,s))$. If we assume that the maximum depth between t and s is n as well the resulting complexity is $O(k^n n^2)$. □

7. Associative and Associative-Unit Generalization: Special Restrictions and Optimality

Below we introduce a special restriction of associative and associative-unit generalization based on the concepts introduced in the previous section. Furthermore, we introduce so called preferred choice functions which allow us to circumvent parts of the given AUP for which computation of a generalization is expensive and can be avoided in the search for a generalization which is at least as good as the syntactic generalization.

7.1. *1-Determined Associative and Associative-Unit Generalization*

We start with defining a strategy of applying associative decomposition rules guided by a given maximal alignment. Assume that we are given the state $State = A; S; \sigma$, where $A = \{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \uplus A'$ for an associative (resp., associative-unit) f and a max-uniform alignment (resp. a maximal alignment) of $pars(f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m))$ has the form $\mathfrak{a} = g_1[i_1, j_1] \cdots g_n[i_k, j_k]$.

Let us denote $X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)$ by P. Recall that it is a flattened form of $X(\vec{x}) : f(t_1, f(t_2, \ldots, f(t_{n-1}, t_n) \cdots)) \triangleq f(s_1, f(s_2, \ldots, f(s_{m-1}, s_m) \cdots))$.

For a number $l < \min(i_1, j_1)$, define $\mathfrak{a} - l$ as the alignment $g_1[i_1 - l, j_1 - l] \cdots g_n[i_k - l, j_k - l]$.

The strategy of eliminating the first alignment element $g_1[i_1, j_1]$ from $\mathfrak{a}$ is defined below. The Y's and Z's are fresh variables of appropriate types. For simplicity, we show only P and its successors.

Case 1: $i_1 = j_1 \geqslant 1$:

$$\begin{aligned} &\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \\ \Longrightarrow^{i_1}_{\mathsf{Dec}} &\{Z(\vec{x}) : f(t_{i_1+1}, \ldots, t_n) \triangleq f(s_{i_1+1}, \ldots, s_m), \\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{i_1}(\vec{x}) : t_{i_1} \triangleq s_{i_1}\}. \end{aligned}$$

Case 2: $i_1 > 1$, $j_1 > 1$, $i_1 < j_1$:

$$\begin{aligned} &\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \\ \Longrightarrow^{i_1-2}_{\mathsf{Dec}} &\{X_1(\vec{x}) : f(t_{i_1-1}, t_{i_1}, \ldots, t_n) \triangleq f(s_{i_1-1}, \ldots, s_{j_1}, \ldots, s_m), \\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{i_1-2}(\vec{x}) : t_{i_1-2} \triangleq s_{i_1-2}\} \\ \Longrightarrow_{\mathsf{Dec\text{-}A\text{-}R}} &\{X_2(\vec{x}) : f(t_{i_1}, \ldots, t_n) \triangleq f(s_{j_1}, \ldots, s_m), \\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{i_1-2}(\vec{x}) : t_{i_1-2} \triangleq s_{i_1-2}, \\ &Y_{i_1-1}(\vec{x}) : t_{i_1-1} \triangleq f(s_{i_1-1}, \ldots, s_{j_1-1})\} \end{aligned}$$

$$\begin{aligned}\Longrightarrow_{\mathsf{Dec}} \{&Z_1(\vec{x}) : f(t_{i_1+1}, \ldots, t_n) \triangleq f(s_{j_1+1}, \ldots, s_m),\\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{i_1-2}(\vec{x}) : t_{i_1-2} \triangleq s_{i_1-2},\\ &Y_{i_1-1}(\vec{x}) : t_{i_1-1} \triangleq f(s_{i_1-1}, \ldots, s_{j_1-1}),\\ &Y_{i_1}(\vec{x}) : t_{i_1} \triangleq s_{j_1})\}.\end{aligned}$$

Case 3: $i_1 > 1, j_1 > 1, i_1 > j_1$:

$$\begin{aligned}&\{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\}\\ \Longrightarrow_{\mathsf{Dec}}^{j_1-2} &\{X_1(\vec{x}) : f(t_{j_1-1}, \ldots, t_{i_1}, \ldots, t_n) \triangleq f(s_{j_1-1}, s_{j_1}, \ldots, s_m),\\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{j_1-2}(\vec{x}) : t_{j_1-2} \triangleq s_{j_1-2}\}\\ \Longrightarrow_{\mathsf{Dec\text{-}A\text{-}L}} &\{X_2(\vec{x}) : f(t_{i_1}, \ldots, t_n) \triangleq f(s_{j_1}, \ldots, s_m),\\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{i_1-2}(\vec{x}) : t_{i_1-2} \triangleq s_{i_1-2},\\ &Y_{j_1-1}(\vec{x}) : f(t_{j_1-1}, \ldots, t_{i_1-1}) \triangleq s_{j_1-1}\}\\ \Longrightarrow_{\mathsf{Dec}} &\{Z_1(\vec{x}) : f(t_{i_1+1}, \ldots, t_n) \triangleq f(s_{j_1+1}, \ldots, s_m),\\ &Y_1(\vec{x}) : t_1 \triangleq s_1, \ldots, Y_{j_1-2}(\vec{x}) : t_{i_1-2} \triangleq s_{i_1-2},\\ &Y_{j_1-1}(\vec{x}) : f(t_{j_1-1}, \ldots, t_{i_1-1}) \triangleq s_{j_1-1},\\ &Y_{j_1}(\vec{x}) : t_{i_1} \triangleq s_{j_1})\}.\end{aligned}$$

When f is AU and $\mathfrak{a}$ is a uniform alignment, the process is similar, but we may have one extra case when $\min(i_1, j_1) = 1$ and $\max(i_1, j_1) > 1$. In this case the applied rule is Dec-AU-L or Dec-AU-R, introducing an AUP with ϵ_f on one of its sides.

Hence, after these transformations we get a new state $State_1$ containing a new problem $P_1 = Z_1(\vec{x}) : f(t_{i_1+1}, \ldots, t_n) \triangleq f(s_{j_1+1}, \ldots, s_m)$. We also obtain a new alignment $\mathfrak{a}_1 = \mathfrak{a} - \min(i_1, j_1)$. Repeating the same process $k-1$ times, we end up with the state $State_k$ containing a new problem $P_k = Z_k(\vec{x}) : f(t_{i_k+1}, \ldots, t_n) \triangleq f(s_{j_k+1}, \ldots, s_m)$. The alignment is now empty. Therefore, we can decompose P_k as follows:

— If $n - i_1 = m - j_1$, then apply Dec $n - i_1$ times.
— If $n - i_1 < m - j_1$, the apply Dec $n - i_1 - 1$ times, followed by an application of Dec-A-R.
— If $n - i_1 > m - j_1$, the apply Dec $m - j_1 - 1$ times, followed by an application of Dec-A-L.

(When f is an AU-symbol, P_k may have ϵ_f in one of its sides. In this case, no decomposition applies.)

We end up with the state $State_{k+1}$, which is the result of total decomposition of P. In $State_{k+1}$ we got rid of at least k occurrences of f compared to $State$. Moreover, since $\mathfrak{a}$ was a max-uniform or maximal alignment, those AUPs in $State_{k+1}$, which do not correspond to any of the element of $\mathfrak{a}$, do not have the same root. In the associative case, the only rule that applies to them is Solve. In the associative-unit case, except Solve also the unit expansion or DH-U rules may apply, but since we aim at computing $\mathcal{E}$-refined generalizations, it is enough to transform them by Solve. It means that we can apply a sequence of Solve rules to $State_{k+1}$, which would keep in A only those AUPs whose root was one of the g's from $\mathfrak{a}$. The other AUPs will move to the store. Let the obtained state be $Cong_{k+2} = A_{k+2}; S_{k+2}; \sigma_{k+2}$.

Take $Y(\vec{x}) : t' \triangleq s' \in A_{k+2}$. If $t' \neq \epsilon_f$, then t' was a subterm of $f(t_1, ..., t_n)$. The same is true for s' and $f(s_1, ..., s_m)$. Let $P - A_{k+2}$ be the AUP obtained from P by replacing all

such subterms (i.e., the non-unit sides of AUPs from A_{k+2}) by some constants. Let us call this operation the *subtraction* from P the AUPs from A_{k+2}.

Theorem 5. Given a state $A; S; \sigma$, where all AUPs in A are total uniformly 1-determined (resp. total 1-determined) associative (resp. associative-unit) generalization problems and the size of A is n, we can reach $\varnothing; S'; \sigma'$

— in time $O(n)$, if all the max-uniform (resp. maximal) nonempty alignments are given;
— in time $O(n^3)$, if the max-uniform (resp. maximal) nonempty alignments are to be computed.

Proof. First, consider the associative case and assume all max-uniform alignments are given. Since the problem is total uniformly 1-determined, all those max-unif-ne-align sets are singletons. For each AUP P in A, we have the following:

—If the root of P is not associative, then by using Dec, Abs, and Solve rules, we either eventually eliminate all the successor problems of P from A in $O(|P|)$ steps, or reach a new problem P' with an associative head in $O(|P - \{P'\}|)$ steps, where $|P - \{P'\}|$ is obtained by subtracting the AUP P' from P.
—If the root of P is associative, then by the decomposition procedure outlined above, in linearly many steps in the size of P, we either eliminate all successor problem of P from A in $O(|P|)$ steps, or reach a new problem P' with an associative head in $O(|P - \{P'\}|)$ steps, where $|P - \{P'\}|$ is obtained by subtracting the AUP P' from P.

It implies that eventually eliminating all AUPs that originate from P takes time $O(|P|)$. Therefore, eliminating all AUPs from A needs time $O(n)$, where n is the size of A.

Now assume the max-uniform alignments should be computed. We do it each time when we encounter an AUP with an associative head. For each such AUP, there is at most one max-uniform nonempty alignment. It can be computed in time quadratic in the size of the AUP by dynamic programming. Since the number of steps when we need to apply these computations is bounded linearly in n, we obtain $O(n^3)$ running time in this case.

The associative-unit case can be proved analogously. □

Based on Theorem 5, we obtain the following theorems:

Theorem 6. A higher-order $\{\mathsf{A}\}$-refined (resp. $\{\mathsf{A}, \mathsf{U}\}$-refined) pattern generalization for a total uniformly 1-determined (resp. total 1-determined) AUP, where all max-uniform (resp. maximal) alignments are given, can be computed in time $O(n)$ where n is the size of the AUP.

Proof. We use the construction outlined in the proof of Theorem 5, by which we can reach the state $\varnothing; S; \sigma$ from the initial one in $O(n)$ time. From $\varnothing; S; \sigma$ till the final answer we proceed as in Baumgartner et al. [2017], which proves overall $O(n)$ runtime complexity.

The obtained generalization is a refined generalization, because the involved alignments are maximal, which ensures that all nonvariable subterms (i.e., those that are not η-equivalent to generalization variables) of the syntactic lgg occur also in the generalization we compute. The only case to be discussed is when our generalizations contain no other nonvariable subterms. In that case, those AUPs that are generalized by a variable in the lgg end up in the store of our derivation, and then the Merge rule will make sure that the computed generalization is as good as the syntactic lgg. Note that for $\{\mathsf{A}, \mathsf{U}\}$-refined generalizations we do not require linearity.

Although we do not use Exp-U-L, Exp-U-R, and DH-U rules for them (the syntactic lgg would anyway use variables to generalize the AUPs to which those rules apply), the merging rule is not forbidden. □

Theorem 7. A higher-order $\{\mathsf{A}\}$-refined (resp. $\{\mathsf{A},\mathsf{U}\}$-refined) pattern generalization for a total uniformly 1-determined (resp. total 1-determined) AUP can be computed in time $O(n^3)$, where n is the size of the AUP.

Proof. By Theorem 5, to reach $\varnothing; S; \sigma$ from the initial state requires $O(n^3)$. From $\varnothing; S; \sigma$, to get to the final answer we need linear time by the algorithm from Baumgartner et al. [2017], which gives the total $O(n^3)$ running time. Proving the refined part is similar to Theorem 6. □

In the next section we consider AUPs which are uniformly k-determined for $k > 1$ but not uniformly $(k-1)$-determined. This will require a new concept of optimality based on a choice function greedily applied during decomposition.

7.2. *Choice Functions and Optimality*

In this section we introduce procedures and optimality conditions for total uniformly k-determined AUPs where $k > 1$, that is AUPs where there are at most k ways to apply equational decomposition.

If we were to compute the set of $\mathcal{E}$-refined generalizations for a total uniformly k-determined AUP by testing every decomposition, even for $k = 2$ the size of the search space is too large to deal with efficiently. However, we can find a $(\mathcal{R}, C, \mathcal{G})$-*optimal* $\mathcal{E}$-*refined generalization* (precisely defined below) in quadratic time, where $\mathcal{R}$ is a rigidity function, C an $\mathcal{R}$-choice function, $\mathcal{G}$ is a set of state transformation rules. Essentially, $(\mathcal{R}, C, \mathcal{G})$-optimality means the $\mathcal{R}$-choice function chooses the "right" computation path via $\mathcal{G}$ based on the rigidity function $\mathcal{R}$. The effect is that we reduce the problem of total uniformly k-determined AUPs to the case of total uniformly 1-determined AUPs with the additional complexity of computing the choice function at each step. We will provide a choice function with linear time complexity based on the procedure for $\mathcal{G}_{\text{base}}$.

Definition 15 ($(P, \mathfrak{a})$-decomposition). Let P be an AUP $X(\vec{x}) : \mathsf{h}(t_1, \ldots, t_n) \triangleq \mathsf{h}(s_1, \ldots, s_m)$, $\mathfrak{a}$ is an alignment of $\langle (root(t_1), \ldots, root(t_n)), (root(s_1), \ldots, root(s_m)) \rangle$ (see Definition 2). An $(P, \mathfrak{a})$-*decomposition of* P is $dec(P, \mathfrak{a}) = \{ Y_{ij}(\vec{x}) : t_i \triangleq s_j \mid h[i,j] \in \mathfrak{a} \}$, where Y_{ij} are new variables of appropriate type.

Definition 16 ($\mathcal{G}$-feasible decomposition). Let $A; S; \sigma$ be a state such that $P \in A$ where P is an AUP $X(\vec{x}) : \mathsf{h}(t_1, \ldots, t_n) \triangleq \mathsf{h}(s_1, \ldots, s_m)$, $\mathfrak{a}$ be an alignment of $\langle (root(t_1), \ldots, root(t_n)), (root(s_1), \ldots, root(s_m)) \rangle$ and $\mathcal{G} \supseteq \mathcal{G}_{\text{base}}$ be a set of state transformation rules. We say that $dec(P, \mathfrak{a})$ is $\mathcal{G}$-*feasible* if there exists $A; S; \sigma \Longrightarrow^* A'; S'; \sigma'$ using $\mathcal{G}$ such that $A' = (A \backslash P) \cup dec(P, \mathfrak{a})$.

Definition 17 ($(\mathcal{R}, P, \mathcal{G})$-branching). Let P be an AUP $X(\vec{x}) : \mathsf{h}(t_1, \ldots, t_n) \triangleq \mathsf{h}(s_1, \ldots, s_m)$, $w_1 = (root(t_1), \ldots, root(t_n))$, $w_2 = (root(s_1), \ldots, root(s_m))$, $\mathcal{R}$ be a rigidity function, and $\mathcal{G} \supseteq \mathcal{G}_{\text{base}}$ be a set of state transformation rules. An $(\mathcal{R}, P, \mathcal{G})$-*branching* is a set $B(\mathcal{R}, P, \mathcal{G}) = \{ dec(P, \mathfrak{a}) \mid \mathfrak{a} \in \mathcal{R}(w_1, w_2)$ and $dec(P, \mathfrak{a})$ is $\mathcal{G}$-feasible$\}$.

Definition 18 ($(\mathcal{R},\mathcal{G})$-choice function). Let $\mathcal{R}$ be a rigidity function and $\mathcal{G} \supseteq \mathcal{G}_{\text{base}}$ be a set of state transformation rules. An $(\mathcal{R},\mathcal{G})$-*choice function* $C_{(\mathcal{R},\mathcal{G})}$ is a partial function from AUPs to alignments such that if for some AUP P we have $C_{(\mathcal{R},\mathcal{G})}(P) = \mathfrak{a}$, then $dec(P,\mathfrak{a}) \in B(\mathcal{R},P,\mathcal{G})$.

Below in the paper we will be using a specific choice function, defined as follows:

Definition 19 (Preferred $(\mathcal{R},\mathcal{G})$-choice function). Let P be an AUP $X(\vec{x}) : \mathsf{h}(t_1,\dots,t_n) \triangleq \mathsf{h}(s_1,\dots,s_m)$. Let $\mathcal{R}$ be a rigidity function and $\mathcal{G} \supseteq \mathcal{G}_{\text{base}}$ be a set of state transformation rules.. A *preferred* $(\mathcal{R},\mathcal{G})$-*choice function* $PC_{(\mathcal{R},\mathcal{G})}$ is an $(\mathcal{R},\mathcal{G})$-choice function defined as

$$PC_{(\mathcal{R},\mathcal{G})}(P) = \begin{cases} \mathfrak{a}_{\min}, & \text{if } B(\mathcal{R},P,\mathcal{G}) \neq \varnothing \\ \textbf{undef}, & \text{otherwise} \end{cases}$$

where $\mathfrak{a}_{\min}$ is an alignment satisfying the following property:

— Let $\sigma_{\mathfrak{a}}$ be the substitution computed by the derivation $\{P\};\varnothing;Id \Longrightarrow^*_{\mathcal{G}} dec(P,\mathfrak{a});S';\sigma' \Longrightarrow^*_{\mathcal{G}_{\text{base}}} \varnothing;S;\sigma_{\mathfrak{a}}$, where $dec(P,\mathfrak{a}) \in B(\mathcal{R},P,\mathcal{G})$. Then $X\sigma_{\mathfrak{a}_{\min}}$ is not more general than any other $X\sigma_{\mathfrak{a}}$.

If there are several such $\mathfrak{a}_{\min}$'s, $PC_{(\mathcal{R},\mathcal{G})}(P)$ is defined as one of them (chosen by some heuristics).

Definition 20 ($(\mathcal{R},C,\mathcal{G})$-optimal generalization). Let A be $\{X : t \triangleq s\}$, $\mathcal{R}$ be a rigidity function, C be an $\mathcal{R}$-choice function, and $\mathcal{G} \supseteq \mathcal{G}_{\text{base}}$ be a set of state transformation rules. We say that a generalization r of the terms t and s is an $(\mathcal{R},C,\mathcal{G})$-*optimal generalization* if $r = X\sigma$, where σ is resulting from the derivation $A;\varnothing;Id \Longrightarrow^* \varnothing;S;\sigma$ using the rules of $\mathcal{G}$, in which every decomposition is either syntactic or is performed with respect to the alignment computed by the choice function C.

All the definitions in this section can be used with uniform and orderless alignments as well. It is straightforward to obtain the corresponding variants and, therefore, we have not spelled them out explicitly.

In the following subsection we show how the above definitions can lead to a more general result (compared to the one in the previous section) concerning associative generalization.

7.3. *k-Determined Associative and Associative-Unit Generalization*

First, we generalize Theorem 5 from 1-determined to k-determined AUPs.

Theorem 8. Given a state $A;S;\sigma$, where all AUPs in A are total uniformly k-determined (resp. total k-determined) associative (resp. associative-unit) generalization problems with $k > 1$ and the size of A is n, we can reach $\varnothing;S';\sigma'$

— in time $O(n^2)$, if all the max-uniform (resp. maximal) nonempty alignments are given;
— in time $O(n^3)$, if the max-uniform (resp. maximal) nonempty alignments are to be computed.

Proof. Case 1, alignments are given: The difference with Theorem 5 is that in the k-determined case we have to make a choice to select the preferred alignment between the given at most k alternatives, when we are decomposing an AUP with associative head. The choice requires running the linear G_{base} algorithm k times. Hence, each choice is made in linear time (in contrast to

1-determined case, when no choice was needed). Consequently, we get quadratic running time. This reasoning applies to both uniform and general cases.

Case 2, alignments are to be computed: When an AUP with an associative head is getting decomposed, we need first to compute alignments and then choose the preferred one among them. The running time of these two consecutive operations is dominated by the alignment computation, which is quadratic (computing at most k alignments, each of them in quadratic time) in contrast to the linear choice in the second step. Hence, it gives cubic running time. This reasoning applies to both uniform and general cases. □

Our rigidity function $\mathcal{R}_{\mathsf{A}}$ takes a pair of symbol sequences and returns the set of their max-uniform nontrivial alignments: $\mathcal{R}_{\mathsf{A}}(w_1, w_2) := \mathsf{max\text{-}unif\text{-}ne\text{-}align}(w_1, w_2)$.

The preferred $(\mathcal{R}_{\mathsf{A}}, \mathcal{G}_{\mathsf{A}})$-choice function uses the linear time procedure $\mathcal{G}_{\mathrm{base}}$ to make a choice of $\mathfrak{a}_{\min}$ between the various possible alignments. Notice that we use associative decomposition for $\{P\}; \varnothing; Id \Longrightarrow^*_{\mathcal{G}_{\mathsf{A}}} dec(P, \mathfrak{a}); S'; \sigma'$ and syntactic decomposition in the derivation $dec(P, \mathfrak{a}); S'; \sigma' \Longrightarrow^*_{\mathcal{G}_{\mathrm{base}}} \varnothing; S; \sigma_{\mathfrak{a}}$.

When we move to the AU-case, we remove the uniformity restriction and consider a different rigidity function: $\mathcal{R}_{\mathsf{AU}}(w_1, w_2) := \mathsf{max\text{-}ne\text{-}align}(w_1, w_2)$. We will denote by $\mathcal{G}_{\mathsf{AU}}$ the algorithm obtained by extending $\mathcal{G}_{\mathrm{base}}$ with Dec-AU-L and Dec-AU-R rules.

Theorem 9. An $(\mathcal{R}_{\mathsf{A}}, PC_{(\mathcal{R}_{\mathsf{A}}, \mathcal{G}_{\mathsf{A}})}, \mathcal{G}_{\mathsf{A}})$-optimal higher-order $\{\mathsf{A}\}$-refined pattern generalization for a total uniformly k-determined AUP, $k > 1$, when all the alignments are given, can be computed in time $O(n^2)$, where n is the size of the AUP.

An $(\mathcal{R}_{\mathsf{AU}}, PC_{(\mathcal{R}_{\mathsf{AU}}, \mathcal{G}_{\mathsf{AU}})}, \mathcal{G}_{\mathsf{AU}})$-optimal higher-order $\{\mathsf{A}, \mathsf{U}\}$-refined pattern generalization for a total k-determined AUP, $k > 1$, when all the alignments are given, can be computed in time $O(n^2)$, where n is the size of the AUP.

Proof. Similar to Theorem 6, using Theorem 8. □

Theorem 10. An $(\mathcal{R}_{\mathsf{A}}, PC_{(\mathcal{R}_{\mathsf{A}}, \mathcal{G}_{\mathsf{A}})}, \mathcal{G}_{\mathsf{A}})$-optimal higher-order $\{\mathsf{A}\}$-refined pattern generalization for a *total uniformly k-determined* AUP, $k > 1$, can be computed in time $O(n^3)$, where n is the size of the AUP.

An $(\mathcal{R}_{\mathsf{AU}}, PC_{(\mathcal{R}_{\mathsf{AU}}, \mathcal{G}_{\mathsf{AU}})}, \mathcal{G}_{\mathsf{AU}})$-optimal higher-order $\{\mathsf{A}, \mathsf{U}\}$-refined pattern generalization for a *total k-determined* AUP, $k > 1$, can be computed in time $O(n^3)$, where n is the size of the AUP.

Proof. Similar to Theorem 7, using Theorem 8. □

Note that we can achieve the same results even if the AUP is not total (uniformly) k-determined, but our rigidity functions enforce computation of maximum k alignments. For instance, we can define $\mathcal{R}_{\mathsf{A}}(w_1, w_2)$ as a set consisting of at most k max-uniform nonempty alignments of w_1 and w_2, which we denote by $\mathsf{max\text{-}unif\text{-}ne\text{-}align}_k$: $\mathcal{R}_{\mathsf{A}}(w_1, w_2) := \mathsf{max\text{-}unif\text{-}ne\text{-}align}_k(w_1, w_2)$. Similarly, we can define $\mathcal{R}_{\mathsf{AU}}(w_1, w_2) := \mathsf{max\text{-}ne\text{-}align}_k(w_1, w_2)$. Then both Theorem 9 and Theorem 10 hold for such rigidity functions without requiring that the AUPs are k-determined. Moreover, if we take $k = 1$, then we can obtain counterparts of Theorem 6 and Theorem 7 without requiring that the AUPs there are total (uniformly) 1-determined.

8. Commutative and Commutative-Unit Case

The C-theory is the simplest one among our (non-free) equational theories. Commutative functions have two arguments whose order does not matter, which implies that the notions of determinedness and uniform determinedness coincide for AUPs with commutative symbols.

For total 1-determined AUPs we have the result about linear complexity of computing $\{\mathsf{C}\}$- and $\{\mathsf{C}, \mathsf{U}\}$-refined generalizations. It does not make a difference whether the alignments are given or not: it just takes constant time to get them at each commutative decomposition step.

Theorem 11. A higher-order $\{\mathsf{C}\}$-refined (and $\{\mathsf{C}, \mathsf{U}\}$-refined) pattern generalization for a total uniformly 1-determined (hence, for a total 1-determined AUP) can be computed in linear time.

Proof. Linear running time follows from linearity of the higher-order pattern generalization algorithm from Baumgartner et al. [2017].

The obtained generalization is a refined generalization, because the alignments guarantee that all subterms of the syntactic lgg, which are not η-equivalent to generalization variables, occur also in the generalization we compute. When our generalizations contain no other nonvariable subterms than the syntactic lgg, those AUPs that are generalized by a variable in the lgg end up in the store of our derivation, and then the Merge rule will make sure that the computed generalization is as good as the syntactic lgg. For $\{\mathsf{C}, \mathsf{U}\}$-refined generalizations we do not require linearity. Although we do not use Exp-U-L, Exp-U-R, and DH-U rules for them (the syntactic lgg would anyway use variables to generalize the AUPs to which those rules apply), the merging rule is not forbidden. ¶ □

When the determinedness restriction is lifted, the preferred choice function for commutative AUPs will have to make a choice among at most two orderless alignments. Hence, one can say that unrestricted AUPs with commutative functions are the same as total 2-determined AUPs with commutative functions. Taking this relation into account, we get the following theorem:

Theorem 12. A $(\mathcal{R}_{\mathsf{C}}, C_{(\mathcal{R}_{\mathsf{C}}, \mathcal{G}_{\mathsf{C}})}, \mathcal{G}_{\mathsf{C}})$-optimal higher-order $\{\mathsf{C}\}$-refined (resp. $\{\mathsf{C}, \mathsf{U}\}$-refined) pattern generalization for any AUP can be computed in $O(n^2)$ time, where n is the size of the AUP.

Proof. Note that we use the same rigidity and choice functions for both $\{\mathsf{C}\}$- and $\{\mathsf{C}, \mathsf{U}\}$-refined generalizations. To decide which of the two alignments at the C-decomposition step would lead to a better generalization we need linear time. Combining it with the statement of Theorem 11, we get the overall quadratic running time. □

9. Associative-Commutative and Associative-Commutative-Unit Case

9.1. *1-Determined* AC *and* ACU *Generalization*

Similarly to the associative and associative-unit case, we define a strategy of applying associative-commutative and associative-commutative-unit decomposition rules guided by a given max-uniform or maximal orderless alignment. Assume that we are given the state $State = A; S; \sigma$,

¶ In principle, we could easily incorporate Exp-U-L and Exp-U-R rules in the derivation, if, e.g., in the AUP $X(\vec{x}) : t \triangleq f(s_1, s_2)$ the root of t is the same as the root of s_1 or s_2, but it is not really necessary for refined generalizations.

where $A = \{X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)\} \uplus A'$ for an associative-commutative (resp., associative-commutative-unit) f. Let $\mathfrak{o} = \{\!\{g_1[i_1, j_1], \ldots, g_n[i_k, j_k]\}\!\}$ be a max-uniform orderless alignment (resp. a maximal orderless alignment) of $pars(f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m))$, where $i_1 = \min\{i_1, \ldots, i_n, j_1, \ldots, j_n\}$ (without loss of generality). Then by $\mathfrak{o} - g_1[i_1, j_1]$ we mean an alignment $\{\!\{g_2[i_2 - 1, j'_2], \ldots, g_n[i_k - 1, j'_k]\}\!\}$, where $j'_l = j_l$ if $j_l < j_1$, and $j'_l = j_l - 1$ if $j_l > j_1$.

Let us denote $X(\vec{x}) : f(t_1, \ldots, t_n) \triangleq f(s_1, \ldots, s_m)$ by P. Recall that it is a flattened form of $X(\vec{x}) : f(t_1, f(t_2, \ldots, f(t_{n-1}, t_n) \cdots)) \triangleq f(s_1, f(s_2, \ldots, f(s_{m-1}, s_m) \cdots))$.

The strategy of eliminating the first alignment element $g_1[i_1, j_1]$ from $\mathfrak{o}$ is defined below. It is much simpler than what we did for the associative and associative-unit cases. The Y's and Z's are fresh variables of appropriate types. For simplicity, we show only P and its successors.

$$\begin{aligned} & \{X(\vec{x}) : f(t_1, \ldots, t_{i_1}, \ldots, t_n) \triangleq f(s_1, \ldots, s_{i_1}, \ldots, s_m)\} \\ \Longrightarrow_{\textsf{Dec-AC-L}} & \{Z_1(\vec{x}) : f(t_1, \ldots, t_{i_1 - 1}, t_{i_1 + 1}, \ldots, t_n) \triangleq f(s_1, \ldots, s_{i_1 - 1}, s_{i_1 + 1}, \ldots, s_m), \\ & Y(\vec{x}) : t_{i_1} \triangleq s_{i_1}\}. \end{aligned}$$

Hence, after this transformation we get a new state $State_1$ containing a new problem $P_1 = Z_1(\vec{x}) : f(t_1, \ldots, t_{i_1 - 1}, t_{i_1 + 1}, \ldots, t_n) \triangleq f(s_1, \ldots, s_{i_1 - 1}, s_{i_1 + 1}, \ldots, s_m)$. We also obtain a new alignment $\mathfrak{o}_1 = \mathfrak{o} - g_1[i_1, j_1]$. Repeating the same process $k - 1$ times, from $f(t_1, \ldots, t_n)$ (resp. from $f(s_1, \ldots, s_m)$) we remove $t_{i_1}, \ldots, t_{i_k}$ (resp. $s_{i_1}, \ldots, s_{i_k}$). When $\mathfrak{o}$ is a max-uniform alignment, there are two possibilities: either the terms $f(t_1, \ldots, t_n)$ and $f(s_1, \ldots, s_m)$ got completely eliminated, or there are some "leftovers" from both of them. the state $State_k$, obtained after these $k - 1$ steps, in the latter case would contain a new problem $P_k = Z_k(\vec{x}) : t' \triangleq s'$, where t' and. s' are those "leftover terms". The alignment is now empty. Assume $t' = f(t'_1, \ldots, t'_{n'})$ and $s' = f(s'_1, \ldots, s'_{m'})$. Assume also without loss of generality that $n' \leqslant m'$. Then we can decompose P_k by applying Dec-AC-R rule $n' - 2$ times, each time removing the first arguments from each side, and then finally applying it again to get two AUPs of the form $t'_{n'-1} \triangleq s'_{n'-1}$ and $t'_{n'} \triangleq f(s'_{n'}, \ldots, s'_{m'})$.

(When f is an ACU-symbol, P_k may have ϵ_f in one of its sides. In this case, no decomposition applies.)

We end up with the state $State_{k+1}$, which is the result of total decomposition of P. In $State_{k+1}$ we got rid of at least k occurrences of f compared to $State$. Moreover, since $\mathfrak{o}$ was a max-uniform or maximal alignment, those AUPs in $State_{k+1}$, which do not correspond to any of the element of $\mathfrak{o}$, do not have the same root. In the AC case, the only rule that applies to them is Solve. In the ACU case, except Solve also the unit expansion or DH-U rules may apply, but since we aim at computing $\mathcal{E}$-refined generalizations, it is enough to transform them by Solve. It means that we can apply a sequence of Solve rules to $State_{k+1}$, which would keep in A only those AUPs whose root was one of the g's from $\mathfrak{o}$. The other AUPs will move to the store. Let the obtained state be $Cong_{k+2} = A_{k+2}; S_{k+2}; \sigma_{k+2}$.

Take $Y(\vec{x}) : t^* \triangleq s^* \in A_{k+2}$. If $t^* \neq \epsilon_f$, then t^* was a subterm of $f(t_1, \ldots, t_n)$. The same is true for s^* and $f(s_1, \ldots, s_n)$. Let $P - A_{k+2}$ be the AUP obtained from P by replacing all such subterms (i.e., the non-unit sides of AUPs from A_{k+2}) by some constants. Let us call this operation the *subtraction* from P the AUPs from A_{k+2}.

Theorem 13. Given a state $A; S; \sigma$, where all AUPs in A are total uniformly 1-determined (resp.

total 1-determined) associative-commutative (resp. associative-commutative-unit) generalization problems and the size of A is n, we can reach $\varnothing; S'; \sigma'$

— in time $O(n)$, if all the max-uniform (resp. maximal) nonempty alignments are given;
— in time $O(n^3)$, if the max-uniform (resp. maximal) nonempty alignments are to be computed.

Proof. First, consider the AC case and assume all max-uniform alignments are given. Since the problem is total uniformly 1-determined, all those oless-max-unif-ne-align sets are singletons. For each AUP P in A, we have the following:

—If the root of P is not AC, then by using Dec, Abs, and Solve rules, we either eventually eliminate all the successor problems of P from A in $O(|P|)$ steps, or reach a new problem P' with an associative head in $O(|P - \{P'\}|)$ steps, where $|P - \{P'\}|$ is obtained by subtracting the AUP P' from P.
—If the root of P is AC, then by the decomposition procedure outlined above, in linearly many steps in the size of P, we either eliminate all successor problem of P from A in $O(|P|)$ steps, or reach a new problem P' with an AC head in $O(|P - \{P'\}|)$ steps, where $|P - \{P'\}|$ is obtained by subtracting the AUP P' from P.

It implies that eventually eliminating all AUPs that originate from P takes time $O(|P|)$. Given an alignment element $g_l[i_l, j_l]$, extracting i_l'th and j_l's subterms from the given AUP can be done in constant time. Therefore, eliminating all AUPs from A needs time $O(n)$, where n is the size of A.

Now assume the max-uniform alignments should be computed. We do it each time when we encounter an AUP with an AC head. For each such AUP, there is at most one max-uniform nonempty alignment. It can be computed in time quadratic in the size of the AUP. Since the number of steps when we need to apply these computations is bounded linearly in n, we obtain $O(n^3)$ running time in this case.

The ACU case can be proved analogously. □

Based on Theorem 13, we obtain the following theorems:

Theorem 14. A higher-order {A, C}-refined (resp. {A, C, U}-refined) pattern generalization for a total uniformly 1-determined (resp. total 1-determined) AUP, where all max-uniform (resp. maximal) orderless alignments are given, can be computed in time $O(n)$ where n is the size of the AUP.

Proof. Similar to the proof of Theorem 6, using Theorem 13. □

Theorem 15. A higher-order {A, C}-refined (resp. {A, C, U}-refined) pattern generalization for a total uniformly 1-determined (resp. total 1-determined) AUP can be computed in time $O(n^3)$, where n is the size of the AUP.

Proof. Similar to the proof of Theorem 7, using Theorem 13. □

9.2. *k-Determined Associative-Commutative and Associative-Commutative-Unit Generalization*

We generalize Theorem 13 from 1-determined to k-determined AUPs. It can be proved similarly to Theorem 5.

Theorem 16. Given a state $A; S; \sigma$, where all AUPs in A are total uniformly k-determined (resp. total k-determined) associative-commutative (resp. associative-commutative-unit) generalization problems with $k > 1$ and the size of A is n, we can reach $\varnothing; S'; \sigma'$

— in time $O(n^2)$, if all the max-uniform (resp. maximal) nonempty orderless alignments are given;

— in time $O(n^3)$, if the max-uniform (resp. maximal) nonempty orderless alignments are to be computed.

For the AC case we use the rigidity function $\mathcal{R}_{\mathsf{AC}}(w_1, w_2) := \mathsf{oless\text{-}max\text{-}unif\text{-}ne\text{-}align}(w_1, w_2)$. The preferred $(\mathcal{R}_{\mathsf{AC}}, \mathcal{G}_{\mathsf{AC}})$-choice function uses the linear time procedure $\mathcal{G}_{\mathrm{base}}$ to make a choice of $\mathfrak{o}_{\min}$ between the various possible alignments. We use associative-commutative decomposition for $\{P\}; \varnothing; Id \Longrightarrow^*_{\mathcal{G}_{\mathsf{A}}} dec(P, \mathfrak{a}); S'; \sigma'$ and syntactic decomposition in the derivation $dec(P, \mathfrak{a}); S'; \sigma' \Longrightarrow^*_{\mathcal{G}_{\mathrm{base}}} \varnothing; S; \sigma_{\mathfrak{a}}$.

For the ACU case, we remove the uniformity restriction and consider a different rigidity function: $\mathcal{R}_{\mathsf{ACU}}(w_1, w_2) := \mathsf{oless\text{-}max\text{-}ne\text{-}align}(w_1, w_2)$. The choice function is $(\mathcal{R}_{\mathsf{ACU}}, \mathcal{G}_{\mathsf{ACU}})$, where the algorithm $\mathcal{G}_{\mathsf{ACU}}$ is obtained by extending $\mathcal{G}_{\mathrm{base}}$ with Dec-ACU-L and Dec-ACU-R.

Then we get the following counterparts of Theorems 9 and 10:

Theorem 17. An $(\mathcal{R}_{\mathsf{AC}}, PC_{(\mathcal{R}_{\mathsf{AC}}, \mathcal{G}_{\mathsf{AC}})}, \mathcal{G}_{\mathsf{AC}})$-optimal higher-order $\{\mathsf{AC}\}$-refined pattern generalization for a total uniformly k-determined AUP with $k > 1$, when all the orderless alignments are given, can be computed in time $O(n^2)$, where n is the size of the AUP.

An $(\mathcal{R}_{\mathsf{ACU}}, PC_{(\mathcal{R}_{\mathsf{ACU}}, \mathcal{G}_{\mathsf{ACU}})}, \mathcal{G}_{\mathsf{ACU}})$-optimal higher-order $\{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$-refined pattern generalization for a total k-determined AUP with $k > 1$, when all the orderless alignments are given, can be computed in time $O(n^2)$, where n is the size of the AUP.

Theorem 18. An $(\mathcal{R}_{\mathsf{AC}}, PC_{(\mathcal{R}_{\mathsf{AC}}, \mathcal{G}_{\mathsf{AC}})}, \mathcal{G}_{\mathsf{AC}})$-optimal higher-order $\{\mathsf{AC}\}$-refined pattern generalization for a *total uniformly k-determined* AUP with $k > 1$ can be computed in time $O(n^3)$, where n is the size of the AUP.

An $(\mathcal{R}_{\mathsf{ACU}}, PC_{(\mathcal{R}_{\mathsf{ACU}}, \mathcal{G}_{\mathsf{ACU}})}, \mathcal{G}_{\mathsf{ACU}})$-optimal higher-order $\{\mathsf{A}, \mathsf{C}, \mathsf{U}\}$-refined pattern generalization for a *total k-determined* AUP with $k > 1$ can be computed in time $O(n^3)$, where n is the size of the AUP.

We can achieve the same results even if the AUP is not total (uniformly) k-determined, but our rigidity functions enforce computation of maximum k orderless alignments. For instance, we can define $\mathcal{R}_{\mathsf{AC}}(w_1, w_2)$ as a set consisting of at most k max-uniform nonempty orderless alignments of w_1 and w_2, which we denote by $\mathsf{oless\text{-}max\text{-}unif\text{-}ne\text{-}align}_k$: $\mathcal{R}_{\mathsf{AC}}(w_1, w_2) := \mathsf{oless\text{-}max\text{-}unif\text{-}ne\text{-}align}_k(w_1, w_2)$. Similarly, we can define $\mathcal{R}_{\mathsf{ACU}}(w_1, w_2)$ as $\mathcal{R}_{\mathsf{ACU}}(w_1, w_2) := \mathsf{oless\text{-}max\text{-}ne\text{-}align}_k(w_1, w_2)$. Then both Theorem 17 and Theorem 18 hold for such rigidity functions without requiring that the AUPs are k-determined. Moreover, if we take $k = 1$, then we can obtain counterparts of Theorem 14 and Theorem 15 without requiring that the AUPs there are total (uniformly) 1-determined.

10. Conclusion

The higher-order equational anti-unification algorithm presented in this paper combines higher-order syntactic anti-unification rules with the decomposition rules for associative, commutative,

AUPs	A	AU	C	CU	AC	ACU
Total 1-determined, uniform, given alignments	$O(n)$ Th. 6	–	$O(n)$ Th. 11	$O(n)$ Th. 11	$O(n)$ Th. 14	–
Total 1-determined, given alignments	–	$O(n)$ Th. 6	$O(n)$ Th. 11	$O(n)$ Th. 11	–	$O(n)$ Th. 14
Total 1-determined, uniform, computed alignments	$O(n^3)$ Th. 7	–	$O(n)$ Th. 11	$O(n)$ Th. 11	$O(n^3)$ Th. 15	–
Total 1-determined, computed alignments	–	$O(n^3)$ Th. 7	$O(n)$ Th. 11	$O(n)$ Th. 11	–	$O(n^3)$ Th. 15
Total k-determined, $k > 1$, uniform, given alignments	$O(n^2)$ Th. 9	–	$O(n^2)$ Th. 12	$O(n^2)$ Th. 12	$O(n^2)$ Th. 17	–
Total k-determined, $k > 1$, given alignments	–	$O(n^2)$ Th. 9	$O(n^2)$ Th. 12	$O(n^2)$ Th. 12	–	$O(n^2)$ Th. 17
Total k-determined, $k > 1$, uniform, computed alignments	$O(n^3)$ Th. 10	–	$O(n^2)$ Th. 12	$O(n^2)$ Th. 12	$O(n^3)$ Th. 18	–
Total k-determined, $k > 1$, computed alignments	–	$O(n^3)$ Th. 10	$O(n^2)$ Th. 12	$O(n^2)$ Th. 12	–	$O(n^3)$ Th. 18

Table 1: $\mathcal{E}$-refined generalizations: equational theories, problem restrictions, running times.

associative-commutative function constant symbols and expansion rules for unit element. This gives a modular algorithm, which can be used for problems with different symbols from different theories without any adaptation.

Higher order A-, C-, U-, AU-, CU-, AC-, and ACU-anti-unification problems are finitary, but in the presence of U, one needs a special control to obtain a terminating algorithm, unless linear generalizations are computed. In practice, often it is desirable to compute only one answer, which is the best one with respect to some predefined criterion. We defined such an optimality criterion, which basically means that an optimal equational solution should be at least a good as the syntactic lgg. We then identified problem forms for which optimal solutions can be computed fast (in linear or polynomial time) by a greedy approach. The results are summarized in Table 1. They remain the same, if we lift the determinedness restriction from the input, but make the rigidity function provide the number of alignments bounded by a predefined constant $k \geqslant 1$.

References

M. Alpuente, S. Escobar, J. Espert, and J. Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014. .

M. Alpuente, D. Ballis, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. ACUOS^2: A high-performance system for modular ACU generalization with subtyping and inheritance. In F. Calimeri, N. Leone, and M. Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2019. ISBN 978-3-030-19569-4. . URL https://doi.org/10.1007/978-3-030-19570-0_11.

H. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North Holland, 1984.

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. .

A. Baumgartner. *Anti-Unification Algorithms: Design, Analysis, and Implementation*. PhD thesis, Johannes Kepler University Linz, 2015.

A. Baumgartner and T. Kutsia. A library of anti-unification algorithms. In 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*, pages 543–557. Springer, 2014. ISBN 978-3-319-11557-3. . URL https://doi.org/10.1007/978-3-319-11558-0_38.

A. Baumgartner and T. Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255: 262–286, 2017. . URL https://doi.org/10.1016/j.ic.2017.01.005.

A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. A variant of higher-order anti-unification. In F. 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, 2013. ISBN 978-3-939897-53-8. . URL https://doi.org/10.4230/LIPIcs.RTA.2013.113.

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. . URL https://doi.org/10.1007/s10817-016-9383-3.

T. R. Besold, K. Kuehnberger, and E. Plaza. Towards a computational- and algorithmic-level account of concept blending using analogies and amalgams. *Connect. Sci.*, 29(4):387–413, 2017. .

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 für Informatik, 2018. ISBN 978-3-95977-077-4. . URL https://doi.org/10.4230/LIPIcs.FSCD.2018.12.

D. M. Cerna and T. Kutsia. A generic framework for higher-order generalizations. In H. Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany.*, volume 131 of *LIPIcs*, pages 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019a. ISBN 978-3-95977-107-8. . URL https://doi.org/10.4230/LIPIcs.FSCD.2019.10.

D. M. Cerna and T. Kutsia. Idempotent anti-unification. *ACM Trans. Comput. Logic*, 21(2): 10:1–10:32, 2019b. URL https://doi.org/10.1145/3359060. To appear.

G. Dowek. Higher-order unification and matching. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1009–1062. Elsevier and MIT Press, 2001. ISBN 0-444-50813-9.

S. Eberhard and S. Hetzl. Inductive theorem proving based on tree grammars. *Ann. Pure Appl. Logic*, 166(6):665–700, 2015. . URL https://doi.org/10.1016/j.apal.2015.01.002.

S. Eberhard, G. Ebner, and S. Hetzl. Algorithmic compression of finite tree languages by rigid acyclic grammars. *ACM Trans. Comput. Logic*, 18(4):26:1–26:20, 2017. ISSN 1529-3785. .

G. Ebner, S. Hetzl, A. Leitsch, G. Reis, and D. Weller. On the generation of quantified lemmas. *Journal of Automated Reasoning*, 63(1):95–126, 2019.

S. Hetzl, A. Leitsch, G. Reis, and D. Weller. Algorithmic introduction of quantified cuts. *Theor. Comput. Sci.*, 549:1–16, 2014. . URL https://doi.org/10.1016/j.tcs.2014.05.018.

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

T. Libal and A. Steen. Towards a substitution tree based index for higher-order resolution theorem provers. In P. Fontaine, S. Schulz, and J. Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning co-located with IJCAR 2016*, volume 1635 of *CEUR Workshop Proceedings*, pages 82–94. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1635/paper-08.pdf.

D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991. .

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

B. Pientka. Higher-order term indexing using substitution trees. *ACM TOCL*, 11(1), 2009. .

R. Rolim, G. Soares, R. Gheyi, and L. D'Antoni. Learning quick fixes from code repositories. *CoRR*, abs/1803.03806, 2018. URL http://arxiv.org/abs/1803.03806.

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

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. ISBN 978-3-642-54515-3. . URL https://doi.org/10.1007/978-3-642-54516-0_7.