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 fragments 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 [2], 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 [16], which makes it unitary, and the single least general generalization can be computed in linear time by the algorithm proposed in [9]. 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 [9] by equational decomposition rules inspired by those from [2]. 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-, C-, U-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. Some application areas include proof generalization [17], higher-order term indexing [18], cognitive modeling and analogical reasoning [10, 21], recursion scheme detection in functional programs [4], inductive synthesis of recursive functions [20], learning fixes from software code repositories [19], just to name a few. Two higher-order anti-unification algorithms [7, 9] are included in an online open-source anti- unification library [5, 6]. First-order order-sorted equational generalization algorithms from [2] have also been implemented and are available online [1]. 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 [15] or in higher order program manipulation tools.

This paper is an extended and improved version of [11]. 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 [9]. In Section 4 we 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 fragments of anti-unification problems. The next sections describe the behavior of equational anti-unification algorithms on these fragments: In Section 7 we discuss A- and AU- generalization and speak

[†] We refer to the higher-order anti-unification algorithm from [9] as syntactic, although it works modulo $\beta\eta$-conversion.

about optimality. Sections 8 is about C- and CU-generalization. Section 10 is about AC- and ACU-generalization. Section 11 summarizes the results.

2. Preliminaries

This work builds upon the formulations and results of [8, 9]. 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 $X,Y,Z,x,y,z,a,b,\ldots$) as well as *constants* (typically $f,c,\ldots$) are assigned types from the set of types constructed using the above grammar. 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 $(\ldots(h\ t_1)\ldots t_m)$, where h is a constant or a variable, will be written as $h(t_1,\ldots,t_m)$, and terms of the form $\lambda x_1.\ldots.\lambda x_n.t$ as $\lambda x_1,\ldots,x_n.t$. We use $\vec{x}$ as a short-hand for $x_1,\ldots,x_n$. This basic language will be extended by higher-order constants satisfying equational axioms. When necessary, we write a λ-term t together with its type α as $t:\alpha$.

Every higher-order 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 properties 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. $f(t,\varepsilon_f) \equiv f(\varepsilon_f,t) \equiv t$, where ε_f is the unique unit element associated with the function constant f. Note that only functions 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 $f(t_1,\ldots,f(s_1,\ldots,s_m),\ldots t_n)$ 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 $\mathsf{A} \in Ax(f)$. 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 [3, 13]). By default, terms are assumed to be written in η-long β-normal form. Therefore, all terms have the form $\lambda x_1,\ldots,x_n.h(t_1,\ldots,t_m)$, where $n,m \geqslant 0$, h is either a constant or a variable, $t_1,\ldots,t_m$ have this form, and the term $h(t_1,\ldots,t_m)$ has a basic 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 $|h(t_1,\ldots,t_n)| = 1 + \sum_{i=1}^{n} |t_i|$ and $|\lambda x.t| = 1 + |t|$. The *depth* of a term t, denoted $depth(t)$ is defined recursively as $depth(h(t_1,\ldots,t_n)) = 1 + \max_{i\in\{1,\ldots,n\}} depth(t_i)$ and $depth(\lambda x.t) = 1 + depth(t)$. For a term $t = \lambda x_1,\ldots,x_n.h(t_1,\ldots,t_m)$ with $n,m \geqslant 0$, its *head* is defined as $head(t) = h$.

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 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 $s =_{\mathscr{E}} t$ if they are equivalent modulo $\mathscr{E} \subseteq \{\mathsf{A},\mathsf{C},\mathsf{U}\}$. 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 modulo an equational theory* $\mathscr{E} \subseteq \{\mathsf{A},\mathsf{C},\mathsf{U}\}$ than σ_2 written $\sigma_1 \preceq_{\mathscr{E}} \sigma_2$ if there exists ϑ such that $X\sigma_1\vartheta =_{\mathscr{E}} X\sigma_2$ for all $X \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)$ Note that $\prec$ and $\simeq$ and the term extension are generalized accordingly. Form 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 $\mathscr{E}$ of two terms t_1 and t_2 if $t \preceq_{\mathscr{E}} t_1$ and $t \preceq_{\mathscr{E}} t_2$. 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 $t \prec_{\mathscr{E}} s$. 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,
— t and 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 of $X(\vec{x}) : t \triangleq s$ is *least general* (or *most specific*) modulo an equational theory $\mathscr{E}$ if there is no anti-unifier ϑ of the same problem that satisfies $\sigma \prec_{\mathscr{E}} \vartheta$. 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: Higher-order terms t and s of the same type in η-long β-normal form and an equational theory $\mathscr{E} \subseteq \{\mathsf{A},\mathsf{C},\mathsf{U}\}$.
Find: A higher-order pattern generalization r of t and s modulo $\mathscr{E} \subseteq \{\mathsf{A},\mathsf{C},\mathsf{U}\}$.

Essentially, we are looking for r which is least general among all higher-order patterns which generalize t and s (modulo $\mathscr{E}$). There can still exist a term which is less general than r, gen-

eralizes both s and t, but is not a higher-order pattern. In [9] 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 $\mathscr{E}$ 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 in the AUPs of the form $X(\vec{x}) : t \triangleq s$ and 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 [9], which works for $\mathscr{E} = \varnothing$. 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**

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

where h is a free constant or $h \in \vec{x}$, 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 basic 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 $\equiv_{\mathscr{E}}$ instead of $=$.

We will refer to these generalization rules as $\mathscr{G}_{base}$. To compute generalizations for two simply typed lambda-terms in η-long β-normal form t and s, the algorithm from [9] starts with the *initial state* $\{X : t \triangleq s\}; \varnothing; \varnothing$, 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.

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 than compare the results. The details are explained below.

We assume that terms are written in *flattened form*, obtained by replacing all subterms of the form $f(t_1,\ldots,f(s_1,\ldots,s_m),\ldots t_n)$ 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. 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 fragments are discussed in the subsequent section.

4.1. *Associative Decomposition Rules*

We start from decomposition rules for associative generalization:

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

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

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 $\mathscr{G}_{base}$ by the above associativity rules as $\mathscr{G}_A$ and extend the termination, soundness and completeness results for $\mathscr{G}_{base}$ to $\mathscr{G}_A$.

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

Proof. Termination follows from the fact that $\mathscr{G}_{base}$ terminates [9] and the rules Dec-A-L and Dec-A-R can be applied finitely many times. □

Theorem 2 (Soundness). If $\{X : t \triangleq s\};\varnothing;\varnothing \Longrightarrow^* \varnothing;S;\sigma$ is a transformation sequence of $\mathscr{G}_A$, 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 [9] that $\mathscr{G}_{base}$ is sound. Let us assume as a base case that all occurrences of associative function symbols 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 function symbols 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 $\max\{k,m\} \leqslant (n+1)$ and let associative function symbols 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 function symbols 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 $\{X(\vec{x}) : t_1 \triangleq t_2\};\varnothing;\varnothing \Longrightarrow^* \varnothing;S;\sigma$ in $\mathscr{G}_A$ such that $\lambda\vec{x}.s \preceq X\sigma$.

Proof. We can reason similarly to the previous proof. It was shown in [9] that $\mathscr{G}_{base}$ is complete. Let us assume as a base case that all occurrences of associative function symbols 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 decompositions rules Dec-A-L and Dec-A-R allow one to consider all groupings. □

The addition of associative function symbols allows for more than one decomposition and thus more than one lgg in contrast to higher-order pattern generalization which results in a unique lgg. 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 $\mathscr{G}_{base}$ by the commutativity rule as $\mathscr{G}_C$. We can easily extend the termination, soundness, and completeness results to $\mathscr{G}_{\mathsf{C}}$. Notice that also for commutative generalization, the lgg is not necessarily unique.

4.3. *Associative-Commutative Decomposition Rules*

Unlike commutativity, which considers a fixed number of terms, and associativity, which enforces an ordering on terms, AC function symbols 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**

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

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

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

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

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

We refer to the extension of $\mathscr{G}_{base}$ by the AC decomposition rules as $\mathscr{G}_{\mathsf{AC}}$. 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 function symbol 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, and $t' \in \{f(t,\varepsilon_f), f(\varepsilon_f,t)\}$.

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, and $s' \in \{f(s,\varepsilon_f), f(\varepsilon_f,s)\}$.

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 1. Let $t = g(a,a)$ and $s = f(g(b,\varepsilon_f),b)$, where $Ax(f) = \{\mathsf{U}\}$ and ε_f is the unit element of f. Note that these are first-order terms. Then we have a derivation:

$$\begin{aligned}
&\{X : g(a,a) \triangleq f(g(b,\varepsilon_f),b)\};\ \varnothing;\ Id \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}L}}\\
&\{X : f(g(a,a),\varepsilon_f) \triangleq f(g(b,\varepsilon_f),b)\};\ \varnothing;\ Id \Longrightarrow_{\mathsf{Dec}}\\
&\{X_1 : g(a,a) \triangleq g(b,\varepsilon_f),\ X_2 : \varepsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(X_1,X_2)\} \Longrightarrow_{\mathsf{Dec}}\\
&\{Y_1 : a \triangleq b,\ Y_2 : a \triangleq \varepsilon_f,\ X_2 : \varepsilon_f \triangleq b\};\ \varnothing;\ \{X \mapsto f(g(Y_1,Y_2),X_2)\}.
\end{aligned}$$

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

$$\begin{aligned}
&f(g(f(Z_1,Z_2),Z_1),h(Z_2))\{Z_1 \mapsto a, Z_2 \mapsto \varepsilon_f\} \equiv_{\mathsf{U}} f(g(a,a),h(\varepsilon_f)) = t\\
&f(g(f(Z_1,Z_2),Z_1),h(Z_2))\{Z_1 \mapsto \varepsilon_f, Z_2 \mapsto a\} \equiv_{\mathsf{U}} f(g(b,\varepsilon_f),h(b)) = s.
\end{aligned}$$

The problem highlighted in Example 1 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 be shared. 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. 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 $\mathscr{G}_{\mathsf{U\text{-}lin}}$.

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**

$$\begin{aligned}
&\{X(\vec{x}) : t \triangleq s\} \uplus A;\ S;\ \sigma \Longrightarrow\\
&\qquad \{Y_1(\vec{x}) : t \triangleq \varepsilon_f,\ Y_2(\vec{x}) : \varepsilon_f \triangleq s\} \uplus A;\ S;\ \sigma\{X \mapsto f(Y_1(\vec{x}),Y_2(\vec{x}))\},
\end{aligned}$$

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

Extending $\mathscr{G}_{\mathsf{U\text{-}lin}}$ with DH-U, we would be able to compute the lgg for terms in Example 1, continuing the derivation that stopped there:

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

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 symbol with the unit element, it will generate an infinite branch in the derivation tree:[‡]

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

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-L 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 $\mathscr{G}_{\mathsf{U\text{-}lin}}$ when the equational axioms involve U. Hence, we will be interested in computing linear generalizations for those theories.

[‡] A similar behavior can be observed in a related theory of idempotence, studied in [12].

5.2. *Generalization Modulo* AU

When an associative function symbol 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 in one side and a non-first argument of its counterpart in 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 $Y(\vec{x}) : \varepsilon_f \triangleq \varepsilon_f$. This is what the condition about nontriviality 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, $f(t_0) = f(t_{n+1}) = \varepsilon_f$, 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, $f(s_0) = f(s_{m+1}) = \varepsilon_f$, 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 an 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 of the one to which the unit expansion rule applied, which implies termination.

Note that if we did put the trivial AUP restriction condition in the AU-decomposition rules, we could get an infinite derivation of the form $\{X : f(a,b) \triangleq a\};\varnothing;Id \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}R}} \{X : f(a,b) \triangleq f(a,\varepsilon_f)\};\varnothing;Id \Longrightarrow_{\mathsf{Dec\text{-}AU\text{-}L}} \{Y : f(a,b) \triangleq a,\ Z : \varepsilon_f \triangleq \varepsilon_f\};\varnothing;\{X \mapsto f(Y,Z)\} \Longrightarrow \cdots$.

Hence, to compute linear generalizations modulo AU we extend $\mathscr{G}_{\mathsf{U\text{-}lin}}$ by AU-decomposition rules. We call this algorithm $\mathscr{G}_{\mathsf{AU}}$. With the AU-decomposition rules we obtain all possible decompositions. The unit expansion rules repair head differences in AUPs for an AU-symbol symbol. Therefore, by $\mathscr{G}_{\mathsf{AU}}$ we will never miss an existing linear lgg of two terms.

5.3. *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 $\mathscr{G}_{\mathsf{U\text{-}lin}}$ the counterpart of Dec-C rule which is applied when $Ax(f) = \{\mathsf{C}, \mathsf{U}\}$, obtaining the algorithm $\mathscr{G}_{\mathsf{CU}}$. Its soundness, termination, and completeness properties are straightforward.

5.4. *Generalization Modulo* ACU

ACU-generalization is characterized by the properties of both AU- and CU- generalizations. From AU, it should inherit the condition that new AUPs are not trivial. It is similar to CU 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 Decomposition Left**

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

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

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

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

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

Extending $\mathscr{G}_{\mathsf{U\text{-}lin}}$ with Dec-ACU-L and Dec-ACU-R, we obtain an algorithm for linear ACU-generalization which we call $\mathscr{G}_{\mathsf{ACU}}$.Its soundness is straightforwards. 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 allows to reach all arguments, by adding the unit element whenever necessary. Therefore, no linear lgg will be missed.

5.5. *Combining Different Theories*

Finally, we consider the general case when different function symbols satisfy different associativity and/or commutativity and/or identity axioms. Like in [2], we can use the rules above all together.

(In the presence of the unit element, we can restrict ourselves with computing linear generalizations to avoid imposing an extra control on rule applications to guarantee termination.) All rules, except DH-U, are local in the sense of [2]: they are local to the given top function symbol in the given AUP they are acting upon, irrespective of what other function symbols 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 function symbols 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 Fragments

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.

In order to have a "reasonable" complexity, we should be able to choose such an optimal state from "reasonably" many alternatives in "reasonable" time. For this, our idea is to treat all the alternative states obtained by an equational decomposition step as syntactic anti-unification problems, compute lggs for each of them (which can be done in linear time), choose the best one among those lggs (e.g., less general than the others, or, if there are several such results, use some heuristics), and restart equational anti-unification algorithm from the state which led to the computation of that best syntactic lgg. When the branching factor is constant, this leads to a quadratic algorithm, and when it is linearly bounded, we get a cubic algorithm. These are the cases we consider below. We would also need to decompose in a more clever way than in the rules above, where the decomposition was based on an arbitrary choice of a subterm.

Hence, we need to identify fragments of equational anti-unification problems which would have the decomposition branching factor constant or linearly bounded. We start by introducing the following concepts.

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

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

Note that every syntactic generalization is also an $\mathscr{E}$-generalization. A direct consequence of this definition is that every element of the minimal complete set of $\mathscr{E}$-generalizations (in our equational theories) of two terms is an $\mathscr{E}$-refined generalization of t and s. However, there might exist $\mathscr{E}$-refined generalizations which do not belong to the minimal complete set of generalizations.

Looking back at the informal description of the construction above, we can say that at each branching point we will be aiming at choosing the alternative that would lead to "the best" $\mathscr{E}$-refined generalization.

The concept of E-refined allows us to compute better generalizations than the base procedure would do, without concerning ourselves with certain difficult 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 terms.

Definition 2 (Alignment, Rigidity Function). Let w_1 and w_2 be strings of symbols. Then the sequence $a_1[i_1, j_1] \cdots a_n[i_n, j_n]$, for $n \geqslant 0$ and a_k are not variables, is an *alignment* if

— is and js are integers such that $0 < i_1 < \cdots < i_n < |w_1|$ and $0 < j_1 < \cdots < j_n < |w_2|$, and
— $a_k = w_1|_{i_k} = w_2|_{j_k}$, for all $1 \leqslant k \leqslant n$. An alignment of the form $a_1[i, j]$ will be referred to as a *singleton alignment*, where $t|_\alpha$ denote the subterm at position α.

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

Definition 3 (Pair of argument head sequences and multisets). Let $t = f(t_1, \ldots, t_n)$ and $s = f(s_1, \ldots, s_m)$. Then the *pair of argument head sequences* and the *pair of argument head multisets* of t and s, denoted respectively as $pahs(t,s)$ and $pahm(t,s)$, are defined as follows:

$$pahs(t,s) = \langle (head(t_1), \ldots, head(t_n)),\ (head(s_1), \ldots, head(s_m)) \rangle.$$
$$pahm(t,s) = \langle \{\!\{head(t_1), \ldots, head(t_n)\}\!\},\ \{\!\{head(s_1), \ldots, head(s_m)\}\!\} \rangle.^{\S}$$

These notions extend to AUPs: A pair of argument head sequences (resp. multisets) of an AUP $X(\vec{x}) : t \triangleq s$ is the pair of argument head sequences (resp. multisets) of the terms t and s.

There is a subset of AUPs, referred to as 1-*Determined AUPs*, which contain associative function symbols and have interesting $\mathscr{E}$-refined generalizations are computable in linear time. The more general r-determined AUPs allow a bounded number of possible choices, that is r choices, whenever associative decomposition may be applied. Even for 2-determined AUPs computing the set of lggs is of exponential complexity. Therefore, we introduce the notion of $(\mathscr{R}, C, \mathscr{G})$-*optimal generalization* where $\mathscr{R}$ is a so called rigidity function [14] and C is a choice function picking one of available decompositions. Under such optimality conditions, we are able to compute an $\mathscr{E}$-refined generalization in quadratic time for k-determined AUPs and in cubic time for arbitrary AUPs with associative function symbols.

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 head in new AUPs (as in [14]), there would still be alternatives. For instance, consider the following AUP where f is associative: $X(\vec{x}) : f(t_1, \ldots t_i, \ldots, t_j, \ldots, t_n) \triangleq f(s_1, \ldots s_i, \ldots, s_j, \ldots, s_m)$. Now let $head(t_i) = head(s_j)$, $head(s_i) = head(t_j)$, and for every other term comparison whose index is $\leqslant j$ the head symbols are not equivalent. Under these assumptions there is not enough information

to decide which decomposition is less general. Furthermore, this can be generalized from two possible decompositions to k possibilities.

Under certain conditions we can force a term to have a single decomposition path, what we will refer to as a 1-*determined* condition which is equivalent to unique longest common subsequence of head symbols. We formally define k-determined AUPs using the following sequence of definitions:

Definition 4 (k-determinate set). Given the pair of sequences of symbols $\langle s_1, s_2 \rangle$ with $s_1 = (a_1, \ldots, a_n)$ and $s_2 = (b_1, \ldots, b_m)$, and a positive integer k, the *(strict) k-determinate set* of s_1 and s_2, denoted $det\,(k, s_1, s_2)$ ($det_s\,(k, s_1, s_2)$), is defined as follows:

— If $n = 0$ and $m \neq 0$ or vice versa, then $det\,(k, s_1, s_2) = \varnothing$.

— Otherwise, let $1 \leqslant i \leqslant \min(n,m)$ be a number such that for the multiset $M_i = (\{\!\{a_1\}\!\} \cap \{\!\{b_1\}\!\}) \cup (\{\!\{a_2, \ldots, a_i\}\!\} \cap \{\!\{b_2, \ldots, b_i\}\!\}) \neq \varnothing$ we have $M_i \cap \{\!\{b_{i+1}, \ldots, b_m\}\!\} = M_i \cap \{\!\{a_{i+1}, \ldots, a_n\}\!\} = \varnothing$. Let K (K_s) be the set of pairs $\{a_{j_1}[j_1, j_2] \mid a_{j_1} = b_{j_2}$ and $j_1 = 1$ iff $j_2 = 1\}$ ($\{a_{j_1}[j_1, j_2] \mid a_{j_1} = b_{j_2}\}$). If K has at most k elements, then

$$det(k, s_1, s_2) := \bigcup_{a_{j_1}[j_1, j_2] \in K} add(a_{j_1}[j_1, j_2], det\left(k, (a_{j_1+1}, \ldots, a_n), (b_{j_2+1}, \ldots, b_m)\right)).$$

$$add(a, A) = \begin{cases} \{(a, A)\} & A \neq \varnothing \\ \varnothing & otherwise \end{cases}$$

— Otherwise, $det(k, s_1, s_2) = \{\varnothing\}$.

Note that $det_s\,(k, s_1, s_2)$ is defined analogously using K_s instead of K. This distinction is important for extending the results of Section 7 to AU-Equational theories because choosing the first position does not cause a misalignment of the two terms (neither does choosing the last position). We will refer to the pairs (a, A) where a is a singleton alignment and A a k-determinate set as *blocks*.

We will use $det_s\,(k, s_1, s_2)$ when considering commutativity in Subsection 7.5 and Section 8.

Example 2. We illustrate the previous definition:

— $det\,(1, (a,b), (a,b)) = \{(a[1,1]\ ;\ \{(b[1,1]\ ;\ \{\varnothing\})\})\}$.
— $det\,(1, (a,a), (b,a)) = \{(\{a[2,2]\ ;\ \{\varnothing\})\}$.
— $det_s\,(1, (a,a), (b,a)) = \{(\{a[1,2]\ ;\ \{\varnothing\})\}$.
— $det\,(1, (a,c,c,b,a,c), (a,d,b,a,c)) = \{(a[1,1]\ ;\ \{(b[3,2]\ ;\ \{(a[1,1]\ ;\ \{(c[1,1]\ ;\ \{\varnothing\})\})\})\})\}$.
— $det\,(1, (a,b,a), (c,a,c,b)) = \{\varnothing\}$
— $det\,(1, (a,b,d), (c,a,b,c)) = \{(b[2,3]\ ;\ \{\varnothing\})\}$
— $det\,(2, (a,b,a), (c,a,b,c)) = \{(b[2,3]\ ;\ \{\varnothing\})\}$
— $det_s\,(1, (a,b), (b,a)) = \{(a[1,2]\ ;\ \{\varnothing\}), (b[2,1]\ ;\ \{\varnothing\})\}$
— $det\,(2, (c,a,b,c), (d,b,a,d)) = \{(a[2,3]\ ;\ \{\varnothing\}), (b[3,2]\ ;\ \{\varnothing\})\}$.
— $det\,(3, (a,b,a,c,d), (c,a,b,a,d)) =$
$\{(b[2,3]\ ;\ \{(a[1,1]\ ;\ \{\varnothing\})\}), (a[3,2]\ ;\ \{(d[2,3]\ ;\ \{\varnothing\})\}), (a[3,4]\ ;\ \{\varnothing\})\}$.
— $det\,(k, (a,a), (b,c,d)) = \{\varnothing\}$.
— $det\,(k, (a,b), (a)) = \varnothing$.
— $det\,(k, (a,a), (a)) = \{\varnothing\}$.

Even though $det\,(k, (a,b), (a))$ and $det\,(k, (a,a), (a))$ are related the formalism does not handle them as similar. This merely makes the formalism a little more restricted. Notice that a unique

longest common subsequence of two symbol sequences is not equivalent to k-determined. Consider the following example:

— $det(k,(c,a,a,d),(c,a,b,a,d)) = \{(c[1,1]\ ;\ \{(a[1,1]\ ;\ \{(d[2,3]\ ;\ \{\varnothing\})\})\})\}$.

The alignment representing its longest common subsequence is

— $c[1,1]a[2,2]a[3,4]d[4,5]$

Definition 5 (k-determined term pairs). A pair of terms $\langle t,s\rangle$ is k-determined iff either $head(t) \neq head(s)$ or $head(t) = head(s) = f$ and $Ax(f) = \varnothing$, or $Ax(f) = \{\mathsf{A}\}$ and $det(k,pahs(t,s)) \neq \varnothing$. Furthermore we say that the pair $\langle t,s\rangle$ is *total k-determined* if $t = \lambda x_1,\ldots,x_n.t'$, $s = \lambda y_1,\ldots,y_n.s'$ and t' and s' are η-equivalent to t'' and s'' with $|t''| = |s''| = 1$, or for $(a[i,j],S) \in det(k,pahs(t,s))$ where t_i is the term at the i^{th} position of t and s_j is the term at the j^{th} position of s the term pair $\langle t_i,s_j\rangle$ is total k-determined.

Proposition 1. The complexity of checking if the terms of an AUP $X(\bar{x}) : \lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n) \triangleq \lambda y_1,\ldots,y_k.f(s_1,\ldots,s_m)$ is 1-determined is $O(n)$ and total 1-determined is $O(n^2)$, where n is maximum of the length of the two terms.

Checking k-determinedness of an AUP is a harder problem complexity-wise. For example, given the sequences $(a,\ldots,a)$ and $(a,\ldots,a)$ there are n^2 ways to align the terms which have to be checked. Moreover, if we want to check total k-determinedness we have to again do a quadratic check for each pair of aligned terms resulting in an $O(n^4)$ procedure.

7. Associative Generalization: Special Fragments and Optimality

7.1. *Associativity and* 1*-Determined AUPs*

We provide a linear time algorithm for higher-order $\{\mathsf{A}\}$-refined pattern generalization of AUPs which are 1-determined. Essentially, at every step there is a single decomposition choice which can be made.

Theorem 4. A higher-order $\{\mathsf{A}\}$-refined pattern generalizer for a *total* 1*-determined* AUP can be computed in linear time.

Proof. If the AUP does not contain an associative function symbol, then its E-refined generalization, which is also an lgg, can be computed in linear time [9]. If it does contain an associative function symbol, we have two alternatives: either every occurrence of the associative function symbol has two arguments (remember that our terms are in flattened form), or not. In the former case, the associative decomposition rules do not differ from the syntactic decomposition rule Dec and we can only apply the latter. It means that we can still use the linear algorithm from [9]. The rest of the proof is about the case when there are occurrences of associative function symbols with more than two arguments. The proof goes by induction on the maximal number of such arguments.

We assume for the induction hypothesis that if every instance of the associative function symbol in the AUP has at most n arguments, then it is solvable in linear time, and show that the same holds for $n+1$. Let us assume that the AUP we are currently considering has the following form $X(\vec{x}) : f(t_1,\ldots,t_m) \triangleq f(s_1,\ldots,s_k)$ where f is associative and $\max\{m,k\} = n+1$. Assume without

loss of generality that $k = n+1$. Also, assume that no other occurrence of f in the given AUP has more than n arguments. We make this assumption in order to reduce the complexity of associative decomposition in the AUP and thus, apply the induction hypothesis. If $head(t_1) = head(s_1)$,then their lgg should not be a variable. Therefore, we can apply Dec-A-L, which results in the AUPs $X(\vec{x}) : t_1 \triangleq s_1$ (whose further decomposition will make sure that they t_1 and s_1 are not generalized by a generalization variable) and $X(\vec{x}) : f(t_2,\ldots,t_m) \triangleq f(s_2,\ldots,s_{n+1})$. Notice that both of the resulting AUPs, by our assumptions, only contain f with not more than n arguments. Thus, by the induction hypothesis the theorem holds in this case.

For the next step we assume s and t are terms of the AUP and that $(h[l,l],S) \in det(1,pahs(t,s))$ s.t. $Ax(h) = \{A\}$. Therefore, we can perform Dec-A-L only on the first argument $l-1$ times, which gives the following new AUPs: $\{X_1(\vec{x}) : t_1 \triangleq s_1,\ldots,X_{l-1}(\vec{x}) : t_{l-1} \triangleq s_{l-1}, X_l(\vec{x}) : f(t_l\ldots,t_m) \triangleq f(s_l,\ldots,s_{n+1})\}$. All the resulting AUPs, by our assumptions, only contain f with not more than n arguments, thus by the induction hypothesis the theorem holds in this case.

For the next step we assume s and t are terms of the AUP and that $(h[i,j],S) \in det(1,pahs(t,s))$ s.t. $Ax(h) = \{A\}$ and $i \neq j$. This is similar to the previous case except there is more than one possible way to apply associative decomposition. More precisely, the number of possible ways is $F(l-j+1)$ where

$$F(0) = 1, \qquad F(r+1) = \sum_{w=1}^{r+1} F(r+1-w) \quad \text{for } r \geqslant 0.$$

which is roughly $F(r) = 2^{(r-1)}$. Note that $F(\cdot)$ is derived from the combinatorics of the associative decomposition rule and concerns the number of possible pairings with respect to 1-determinacy. However, being that none of the head symbols of obtained term-pairs are equivalent nor can their head symbols be equivalent to f, we know that none of the resulting AUPs will require further decomposition. Thus, we need to apply associative decomposition. This can be easily performed be performed by some heuristic. The result will be a set of AUPs containing $X(\vec{x}) : f(t_j\ldots t_m) \triangleq f(s_l,\ldots s_{n+1})$ and thus by the induction hypothesis and our assumptions, the theorem holds.

For the final step we just need to apply a simple induction argument on the number of times in a term the associative symbol f occurs with arity $n+1$. The above argument provides the step case and base case being that we prove the theorem for one occurrence and can use the proof for p occurrences. Thus, the theorem holds. □

In the next section we consider AUPs which are k-determined for $k > 1$. This will requires a new concept of optimality based on a choice function greedily applied during decomposition.

7.2. *Choice Functions and Optimality*

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

If we were to compute the set of E-refined generalizations for a total k-determined AUP by testing every decomposition, even for $k = 2$ the size of search space is too large to deal with efficiently. However, we can find a $(\mathscr{R},C,\mathscr{G})$-*optimal E-refined generalization* (precisely defined below) in quadratic time, where $\mathscr{R}$ is a singleton rigidity function, C a $\mathscr{R}$-choice function, $\mathscr{G}$ is a set of state transformation rules. Essentially, $(\mathscr{R},C,\mathscr{G})$-optimality means the $\mathscr{R}$-choice function

chooses the "right" computation path via $\mathscr{G}$ based on the singleton rigidity function $\mathscr{R}$. The effect is that we reduce the problem of total k-determined AUPs to the case of total 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 $\mathscr{G}_{base}$.

We will denote the set of all AUPs by $\mathbb{A}$. We will need the concept for the following definitions.

Definition 6 ((P,a)-decomposition). Let $P \equiv X(\bar{x}) : \lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n) \triangleq \lambda y_1,\ldots,y_k.f(s_1,\ldots,s_m)$, a is an alignment of $\langle w_1,w_2\rangle_P$ (see Definition 3). An *(P,a)-decomposition of P* is $dec(P,a) = \{Y_{(i,j)}(\vec{y}_{(i,j)}) : t_i \triangleq s_j \mid h[i,j] \in a\ \}$, where $Y_{(i,j)}$ are new variables of appropriate type and $\vec{y}_{(i,j)}$ are bound variables from $\vec{x}$, which appear in $t_i \triangleq s_j$.

Definition 7 ($\mathscr{G}$-feasible). Let $A;S;\sigma$ be a state s.t. $P \in A$ where $P \equiv X(\vec{x}) : \lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n) \triangleq \lambda y_1,\ldots,y_k.f(s_1,\ldots,s_m)$, a be an alignment of $\langle w_1,w_2\rangle_P$ and $\mathscr{G}_{base} \subseteq \mathscr{G}$ be a set of state transformation rules. We say that $dec(P,a)$ is *$\mathscr{G}$-feasible* if there exists $A;S;\sigma \Longrightarrow^* A';S';\sigma'$ using $\mathscr{G}$ such that $A' = (A\backslash P) \cup dec(P,a)$.

Definition 8 ($(\mathscr{R},P,\mathscr{G})$-branching). Let $P \equiv X(\vec{x}) : \lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n) \triangleq \lambda y_1,\ldots,y_k.\ f(s_1,\ldots,s_m)$, $\langle w_1,w_2\rangle_P$ be its pair of argument head sequences, $\mathscr{R}$ be a singleton rigidity function, and $\mathscr{G}_{base} \subseteq \mathscr{G}$ be a set of state transformation rules. An *$(\mathscr{R},P,\mathscr{G})$-branching* is a set $B(\mathscr{R},P) = \{dec(P,a) \mid a \in \mathscr{R}(w_1,w_2)$ and $dec(P,a)$ is $\mathscr{G}$-feasible$\}$.

Definition 9 ($\mathscr{R}$-Choice function). Let $\mathscr{R}$ be a singleton rigidity function and $\mathscr{G}_{base} \subseteq \mathscr{G}$ be a set of state transformation rules. An $\mathscr{R}$-choice function $C_{(\mathscr{R},\mathscr{G})} : \mathbb{A} \to \mathbf{A}$ is a partial function from AUPs to alignments such that if for some $P \in \mathbb{A}$, $C_{(\mathscr{R},\mathscr{G})}(P) = a$, then $dec(P,a) \in B(\mathscr{R},P)$.

Definition 10 ($(\mathscr{R},C,\mathscr{G})$-optimal generalization). Let A be $\{X(\bar{x}) : t \triangleq s\}$, $\mathscr{R}$ be a singleton rigidity function, C be an $\mathscr{R}$-choice function, and $\mathscr{G}_{base} \subseteq \mathscr{G}$ be a set of state transformation rules, which compute generalizations. We say that a generalization k of the terms t and s is an $(\mathscr{R},C,\mathscr{G})$-optimal generalization if $r = X\sigma$, where σ is resulting from the derivation $A;\varnothing;\varnothing \Longrightarrow^* \varnothing;S;\sigma$ using the rules of $\mathscr{G}$, in which every decomposition is either syntactic or respects C-equivalence.

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 Generalization*

Before defining our concrete choice function, we must define the singleton rigidity function we will use. Intuitively, it should select alignments from prefixes of involved sequences. The prefixes are of the same length and should be maximal among those that contain at most k common elements. Formally, it is defined as follows:

Definition 11. Let $w_1 = (a_1,\ldots,a_n)$ and $w_2 = (b_1,\ldots,b_m)$ be sequences of symbols and $k \geqslant 1$ be an integer. We define the singleton rigidity function $\mathscr{R}^k_{\mathsf{A}}$ as

$$\mathscr{R}^k_{\mathsf{A}}(w_1,w_2) = \left\{ \begin{array}{c} \{a_l[l,k] \mid (a_l[l,k],S) \in det(k,w_1,w_2)\} \\ \varnothing \end{array} \middle| \begin{array}{c} det(k,w_1,w_2) \neq \varnothing \\ otherwise \end{array} \right. \tag{1}$$

Now we define a choice function taking an arbitrary singleton rigidity function.

Definition 12. Let $P \equiv X(\vec{x}) : \lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n) \triangleq \lambda y_1,\ldots,y_k.\ f(s_1,\ldots,s_m)$ be an AUP and f a function symbol such that $Ax(f) \not\equiv \varnothing$. We define the choice function $C_{(\mathscr{R},\mathscr{G})}$, where $\mathscr{R}$ is a singleton rigidity function, and $\mathscr{G}$ is a set of state transformation rules containing $\mathscr{G}_{base}$, as follows:

$$C_{(\mathscr{R},\mathscr{G})}(P) = \left\{ \begin{array}{c|ll} a_{\min} & B(\mathscr{R},P) \not\equiv \varnothing \\ \textbf{undef} & otherwise \end{array} \right. \tag{2}$$

where $a_{\min}$ is an alignment of $(head(t_1),\ldots,head(t_n))$ and $(head(s_1),\ldots,head(s_m))$ such that

— $dec(P,a_{\min}) \in B(\mathscr{R},P)$,

— for $dec(P,a) \in B(\mathscr{R},P)$, let $D(a)$ be the derivation $D(a) = \{P\};\varnothing;\varnothing \Longrightarrow^*_{\mathscr{G}} dec(P,a);S';\sigma' \Longrightarrow^*_{\mathscr{G}_{base}} \varnothing;S;\sigma_a$.

Then for each $a \neq a_{\min}$, the corresponding $D(a)$ computes σ_a such that $X\sigma_a$ is more general than $X\sigma_{a_{\min}}$, where $\sigma_{a_{\min}}$ is computed by $D(a_{\min})$. If there are several such $a_{\min}$'s, $C_{(\mathscr{R},\mathscr{G})}(P)$ is defined as one of them (chosen by some heuristics).

The choice function outlined above uses the linear time procedure $\mathscr{G}_{base}$ to make a choice between the various possible alignments. Notice that we use associative decomposition for $\{P\};\varnothing;\varnothing \Longrightarrow^* dec(P,a);S';\sigma'$ and syntactic decomposition in the derivation $dec(P,a);S';\sigma' \Longrightarrow^* \varnothing;S;\sigma_a$.

Theorem 5. A $(\mathscr{R}^k_A, C_{(\mathscr{R}^k_A,\mathscr{G}_A)}, \mathscr{G}_A)$-optimal higher-order $\{\mathsf{A}\}$-refined pattern generalization for a *total k-determined* AUP $X(\vec{x}) : t \triangleq s$ can be computed in $O(n^2)$ where n is the size of the AUP.

Proof.

This follows from the existence of a linear algorithm for the computation of lggs using $\mathscr{G}_{base}$ and the linear time algorithm of theorem 4. Note that k is constant and thus does not show up in complexity statement. □

7.4. *Step Optimal Generalization for Full Associativity*

Completely dropping the determinedness restrictions on the AUPs containing associative function symbols is the same as considering $O(n)$-determined AUPs. We have already shown that this problem is naively solvable by an exponential procedure, even when we consider $O(1)$-determined AUPs. In this section we again consider the problem of finding a $(\mathscr{R}_A^{O(n)}, C_{(\mathscr{R}_A^{O(n)},\mathscr{G}_A)}, \mathscr{G}_A)$-optimal generalization where n in the Landau-notation refers to the maximum number of arguments of any subterms in the given AUP. However, this time the resulting algorithm is cubic in complexity being that r in r–determined is no longer a constant. By $\mathscr{R}_A^{O(n)}$ we mean the singleton rigidity function which instead of looking for an r-determined subsequence just considers the largest feasible multiset intersection.

Theorem 6. A $(\mathscr{R}_A^{O(n)}, C_{(\mathscr{R}_A^{O(n)},\mathscr{G}_A)}, \mathscr{G}_A)$-optimal higher-order $\{\mathsf{A}\}$-refined pattern generalization for an AUP $X(\vec{x}) : t \triangleq s$ can be computed in $O(n^3)$ time where n is the size the AUP.

Now that we have completed our analysis of associative function symbols, the simplest of the cases we consider, we move on to the more interesting cases of unit and commutative decomposition as well as the combinations of these algebraic properties.

7.5. *Associative-Unit Generalization: Eliminating Alignment Restrictions*

As we have made mention of earlier, AU-decomposition removes the restriction on the decomposition of the first and last position of the terms which forced us to develop the concept of k-determined for the development of an efficient algorithm. Thus, in the AU case we may use the less restrictive concept of strict k-determined which we will also use in the next section concerning commutative decomposition. The proof of the following theorem is similar to that of Theorem 4 except we remove any restrictions on the choice of index when performing AU-decomposition, i.e. we may choose the first and last position in the given term.

Theorem 7. A higher-order $\{\mathsf{A},\mathsf{U}\}$-refined pattern generalizer for a *total strict* 1*-determined* AUP can be computed in linear time.

We can further extend the results for A-decomposition to AU-decomposition by defining a rigidity function dependent on strict k-determined rather than k-determined.

Definition 13. Let $w_1 = (a_1,\ldots,a_n)$ and $w_2 = (b_1,\ldots,b_m)$ be sequences of symbols and $k \geqslant 1$ be an integer. We define the singleton rigidity function $\mathscr{R}^k_{\mathsf{AU}}$ as

$$\mathscr{R}^k_{\mathsf{AU}}(w_1,w_2) = \left\{ \begin{array}{c|c} \{a_l[l,k] \mid (a_l[l,k],S) \in det_s(k,w_1,w_2)\} & det_s(k,w_1,w_2) \neq \varnothing \\ \varnothing & otherwise \end{array} \right. \tag{3}$$

This allows us to extend both Theorem 5 & 6 to AU-decomposition. The proofs of these extension follow the same line of argumentation as the proofs of Theorem 5 & 6.

Theorem 8. A $(\mathscr{R}^k_{AU}, C_{(\mathscr{R}^k_{AU},\mathscr{G}_{AU})}, \mathscr{G}_{AU})$-optimal higher-order $\{\mathsf{A},\mathsf{U}\}$-refined pattern generalization for a *total strict k-determined* AUP $X(\vec{x}) : t \triangleq s$ can be computed in $O(n^2)$ where n is the size of the AUP.

Theorem 9. A $(\mathscr{R}^{O(n)}_{AU}, C_{(\mathscr{R}^{O(n)}_{AU},\mathscr{G}_{AU})}, \mathscr{G}_{AU})$-optimal higher-order $\{\mathsf{A},\mathsf{U}\}$-refined pattern generalization for an AUP $X(\vec{x}) : t \triangleq s$ can be computed in $O(n^3)$ time where n is the size the AUP.

8. Commutative Case

Notice that in the case of commutative decomposition if all four terms (or three terms) have the same head symbol we end up with similar issues as in the associativity case. We can use strict 2-determined to restrict the considered AUPs.

Theorem 10. A higher-order $\{\mathsf{C}\}$-refined pattern generalization, for a *total strict 1-determined* AUP can be computed in linear time.

Proof. Similar to the proof of Theorem 4. □

Note that the case $f(t_1,t_2) \triangleq f(s_1,s_2)$, where $head(t_1) = head(s_1)$ and $head(t_2) = head(s_2)$, is considered by the procedure of Theorem 10, but not $f(t_1,t_2) \triangleq f(s_2,s_1)$ This is an issue with the definition of total strict 1-determined. We can fix this problem by performing an addition check to see if a permutation of the terms on the left or right side results in a better alignment. We now present a procedure for full commutativity, that is without restrictions which has a quadratic complexity (see Theorem 5.

Definition 14. Let $w_1 = (a_1, \ldots, a_n)$ and $w_2 = (b_1, \ldots, b_m)$ be sequences of symbols and $k \geqslant 1$ be an integer. We define the rigidity function $\mathscr{R}_{\mathsf{C}}$ returning all alignments.

When the rigidity function $\mathscr{R}_{\mathsf{C}}$ is used all by our procedure there will be at most 4 alignments.

corollary 1. A $(\mathscr{R}_{\mathsf{C}}, C_{(\mathscr{R}_{\mathsf{C}}, \mathscr{G}_{\{\mathsf{C}\}})}, \mathscr{G}_{\{\mathsf{C}\}})$-optimal higher-order $\{C\}$-refined pattern generalization for an AUP can be computed in quadratic time.

9. Commutative-Unit Case: Extra Alignments

Unlike the case of AU-decomposition the CU-decomposition case is not much different from the Commutative case being that one had to already use the concept of strict k-determined. What does change in this case is how often one must check for the possibility of applying commutative decomposition. Whenever we have a term of type α and a function f of type $\alpha \to \alpha \to \alpha$ which is commutative and has a unit element, we must check the usefulness of performing decomposition with respect to $f(\alpha, \varepsilon_f)$. While in the case of total strict 1-determined terms this is not expensive, it adds a bit of overhead in the case of total k-determined terms.

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

Proof. Similar to the proof of Theorem 4. □

The rigidity function is similar that of the commutative case except we need to add in all of the possible unit elements for functions which are both commutative and have a unit element.

Definition 15. Let $w_1 = (a_1, \ldots, a_n, \varepsilon_{f_1}, \cdots \varepsilon_{f_r})$ and $w_2 = (b_1, \ldots, b_m, \varepsilon_{f_1}, \cdots \varepsilon_{f_r})$ be sequences of symbols, $k \geqslant 1$, $r \geqslant 0$, and $Ax(f_i) = \{\mathsf{C}, \mathsf{U}\}$, for $1 \leqslant i \leqslant r$. We define the rigidity function $\mathscr{R}_{\mathsf{CU}}$ returning all alignments.

corollary 2. A $(\mathscr{R}_{\mathsf{C,U}}, C_{(\mathscr{R}_{\mathsf{CU}}, \mathscr{G}_{\{\mathsf{CU}\}})}, \mathscr{G}_{\{\mathsf{CU}\}})$-optimal higher-order $\{\mathsf{C}, \mathsf{U}\}$-refined pattern generalization for an AUP can be computed in quadratic time.

10. Associative-Commutative Case

In this section we consider functions f such that $Ax(f) = \{\mathsf{A}, \mathsf{C}\}$. Unfortunately, when a function is both associative and commutative, the number of possible decomposition paths is even greater than the previously considered cases and thus we need to further restrict the term structure. To provide a better understanding of why this is the case, consider a k-determined AUP where the multiset intersection is of size $O(k)$ and only contains one function symbol. This implies that there are $O(k^2)$ possible decompositions of the terms in the first multiset intersection of the terms containing k alignments. This is not even considering that there might be more than one function symbol in the AUP. The problem is that the more terms with the same head symbol, the more combinations we must check. Unlike commutativity, which considers a fixed number of terms, and associativity, which enforces an ordering on terms, associative-commutativity allows an arbitrary number of arguments with no fixed ordering on the terms. We can get around this problem by

considering special cases of AUPs where arguments of an associative-commutativity symbol have distinct heads.

Unfortunately, the concept of (strict) k-determined AUPs does not lead to a linear algorithm in the case of AC-generalization. Actually, this concept is not even meaningful for such an equational theory, since terms are not ordered in any particular way. Instead, we need to consider so called (k,l)-distinct AUPs, which are defined as follows:

Definition 16. Let $P \equiv X(\vec{x}) : \lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n) \triangleq \lambda y_1,\ldots,y_k.\ f(s_1,\ldots,s_m)$, $pahm(f(t_1,\ldots,t_n),f(s_1,\ldots,s_m)) = \langle T,S\rangle$, and $Ax(f) = \{\mathsf{A},\mathsf{C}\}$. We say that P is (k,l)-*distinct* if each $h \in T \cap S$ occurs at most k times in w_1 and at most k times in w_2, the number of symbols in $T \cap S \leqslant l$ and $T\backslash(T\cap S) \equiv \varnothing$ iff $S\backslash(T\cap S) \equiv \varnothing$. We say $P \equiv X(\vec{x}) : \lambda x_1,\ldots,x_w.t \triangleq \lambda y_1,\ldots,y_r.s$ is *total* (k,l)-*distinct* if $|t| = |s| = 1$ or for every pair of subterms (t',s') of t and s such that $head(t') = head(s')$, the AUP $Y(\vec{y}) : t' \triangleq s'$ is total (k,l)-distinct.

This concept is much simpler than k-determined in that it basically splits the arguments of the left and right side of the given AUP into at most l sections dependent on the head symbols of the arguments. Also, for head function symbol, there should be at most k occurrences of it and the result of decomposition is an empty term iff the terms of the left and right side of the AUP are empty.

When an AUP is total $(1,l)$-distinct there is only one way to decompose the AUP, i.e. either a given symbol shows up in both w_1 and w_2 once and can be aligned, or it cannot be aligned and is generalized by a new variable.

This leads to the following results:

Theorem 12. A higher-order $\{\mathsf{A},\mathsf{C}\}$-refined pattern generalization for a *total* $(1,l)$-*distinct* AUP can be computes in linear time.

Proof. Similar to the proof of Theorem 4. □

If we attempt to relax these constraints the time complexity of the algorithm increases substantially, even when we consider the case of $(2,l)$-distinct AUPs under our restricted optimality condition.

Definition 17. Let $w_1 = (a_1,\ldots,a_n)$ and $w_2 = (b_1,\ldots,b_m)$ be sequences of symbols. We define the singleton rigidity function $\mathscr{R}^{(k,l)}_{AC}$ as follows

$$\mathscr{R}^{(k,l)}_{AC}(w_1,w_2) = \left\{ \begin{array}{c} \{a_l[i,j] \mid \ a_i = b_j, 1 \leqslant i \leqslant n 1 \leqslant j \leqslant m \ \} \\ \varnothing \end{array} \right| \begin{array}{c} \text{if } (w_1,w_2) \text{ is } (k,l)\text{-distinct} \\ \text{otherwise} \end{array} \tag{4}$$

Theorem 13. A $(\mathscr{R}^{(k,l)}_{AC}, C_{(\mathscr{R}^{(k,l)}_{AC},\mathscr{G}_{AC})}, \mathscr{G}_{AC})$-optimal higher-order $\{\mathsf{A},\mathsf{C}\}$-refined pattern generalization for a *total* (k,l)-*distinct* AUP is computed in $O(k^{2\cdot l}\cdot n^2)$ time where n is the input size.

Proof. There are $O(k^2)$ ways to pair the terms with the same head and there are l blocks thus there are $O(k^{2\cdot l})$ computations using $\mathscr{G}_{base}$ (complexity $O(n)$) to be performed on an AUP with size n. □

Obviously, computing the full set of E-refined generalizations from the results of Theorem 13 using a naive method would take in the order of $O(k^{2\cdot l\cdot n})$ time.

10.1. *Associative-Commutative-Unit: Forgetting about misalignments*

Unlike the concept of k-determined the counterpart for (k,l)-distinct, what we will refer to as strict (k,l)-distinct has less to do with positions and more to do with misalignments between the left and right side of an AUP. The concept of (k,l)-distinct enforces that $T\backslash(T\cap S)\equiv\varnothing$ iff $S\backslash(T\cap S)\equiv\varnothing$ which states that if something does not align on one side, then there is something on the other side which also does not align. However, when a unit element is present this is not necessary given that misalignments can always be aligned with unit. This observation leads to the following definition:

Definition 18. Let $P\equiv X(\vec{x}):\lambda x_1,\ldots,x_l.f(t_1,\ldots,t_n)\triangleq\lambda y_1,\ldots,y_k.\ f(s_1,\ldots,s_m)$, $pahm(f(t_1,\ldots,t_n),f(s_1,\ldots,s_m))=\langle T,S\rangle$, and $Ax(f)=\{\mathsf{A},\mathsf{C}\}$. We say that P is *strict* (k,l)*-distinct* if each $h\in T\cap S$ occurs at most k times in w_1 and at most k times in w_2 and the number of symbols in $T\cap S\leqslant l$. We say $P\equiv X(\vec{x}):\lambda x_1,\ldots,x_w.t\triangleq\lambda y_1,\ldots,y_r.s$ is *total strict* (k,l)*-distinct* if $|t|=|s|=1$ or for every pair of subterms (t',s') of t and s such that $head(t')=head(s')$, the AUP $Y(\vec{y}):t'\triangleq s'$ is total strict (k,l)-distinct.

From this concept we are able to present a linear time algorithm for a fragment of AUPs containing ACU function symbols:

Theorem 14. A higher-order $\{\mathsf{A},\mathsf{C},\mathsf{U}\}$-refined pattern generalization for a *total strict* $(1,l)$*-distinct* AUP can be solved in linear time.

Proof. Similar to the proof of Theorem 4. □

By relaxing these constraints the time complexity of the algorithm increases substantially, even when we consider the case of strict $(2,l)$-distinct AUPs under our restricted optimality condition.

Definition 19. Let $w_1=(a_1,\ldots,a_n)$ and $w_2=(b_1,\ldots,b_m)$ be sequences of symbols. We define the singleton rigidity function $\mathscr{R}^{(k,l)}_{ACU}$ as follows

$$\mathscr{R}^{(k,l)}_{ACU}(w_1,w_2)=\left\{\begin{array}{c|c}\{a_l[i,j]\mid\ a_i=b_j,1\leqslant i\leqslant n,1\leqslant j\leqslant m\ \} & \text{if } (w_1,w_2) \text{ is strict } (k,l)\text{-distinct}\\ \varnothing & \text{otherwise}\end{array}\right. \tag{5}$$

Theorem 15. A $(\mathscr{R}^{(k,l)}_{ACU},C_{(\mathscr{R}^{(k,l)}_{ACU},\mathscr{G}_{ACU})},\mathscr{G}_{ACU})$-optimal higher-order $\{\mathsf{A},\mathsf{C},\mathsf{U}\}$-refined pattern generalization for a *total strict* (k,l)*-distinct* AUP is computed in $O(k^{2\cdot l}\cdot n^2)$ time where n is the input size.

Proof. Similar to the proof of Theorem 13. □

As with Theorem 13 computing the full set of E-refined generalizations from the results of Theorem 15 using a naive method would take in the order of $O(k^{2\cdot l\cdot n})$ time.

11. 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, associative-commutative function 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.

References

M. Alpuente, D. Ballis, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. ACUOS2: A high-performance system for modular acu generalization with subtyping and inheritance. In *Proceedings of 16th European Conference on Logics in Artificial Intelligence (JELIA 2019)*, Lecture Notes in Computer Science. Springer, 2019. To appear.

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

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. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2014.

A. Baumgartner and T. Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017.

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*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl, 2013.

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.

T. R. Besold, K. Kühnberger, 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 fuer Informatik, 2018.

D. M. Cerna and T. Kutsia. Idempotent anti-unification. Technical report, RISC, 2018.
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.
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.
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.
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.
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.