Unital anti-unification: type and algorithms

David M. Cerna
Johannes Kepler Univerisity Linz, Austria
david.cerna@risc.jku.at

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

Abstract

Unital equational theories are defined by axioms that assert the existence of the unit element for some function symbols. We study anti-unification (AU) in unital theories and address the problems of establishing generalization type and designing anti-unification algorithms. First, we prove that when the term signature contains at least two unital functions, anti-unification is of the nullary type by showing that there exists an AU problem, which does not have a minimal complete set of generalizations. Next, we consider two special cases: the linear variant and the fragment with only one unital symbol, and design AU algorithms for them. The algorithms are terminating, sound, complete, and return tree grammars from which the set of generalizations can be constructed. Anti-unification for both special cases is finitary. Further, the algorithm for the one-unital fragment is extended to the unrestricted case. It terminates and returns a tree grammar which produces an infinite set of generalizations. At the end, we discuss how the nullary type of unital anti-unification might affect the anti-unification problem in some combined theories, and list some open questions.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Tree languages; Theory of computation → Equational logic and rewriting

Keywords and phrases Anti-unification, tree grammars, unital theories, collapse theories

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

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

1 Introduction

We consider the equational theory of function symbols with unit element (also know as identity), U, which is defined by the axioms $f(x, \epsilon_f) \approx x$ and $f(\epsilon_f, x) \approx x$, where ϵ_f is a special constant, the unit element, associated with the function f. These axioms state that the function symbol f is *unital* and that its unit is ϵ_f. We refer to such theories, containing only these type of axioms, as *unital theories*. This property is ubiquitous in algebra, and is essential to the two basic arithmetic operations $+$ and $\cdot$ as well as the union ($\cup$) and intersection ($\cap$) operations on sets. Furthermore, it is an example of a regular collapse theory [16], which means that the variable sets of both sides of the defining axiom(s) are the same (the regularity property), and it contains an axiom of the form $t \approx x$, where t is a non-variable term and x is a variable (the collapse property). Besides idempotency [8, 10], it is the simplest well-known such theory.

Unification and matching in unital theories has been shown to be NP-complete [17]. Otherwise, investigations concerning unital unification mostly focused on its combination with well known equational theories such as associativity (A), commutativity (C), idempotency (I), see, e.g., [2] for a survey.

As for anti-unification in unital theories, one of the earliest examples is generalization in free monoids [7]. More recent work [1] considers problems over arbitrary term alphabets with some binary symbols being unital, and proposes a modular algorithm for anti-unification in A, C, U theories and their combinations. The set of generalizations computed by the unital

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

anti-unification algorithm there is not complete in general (as one can see from Example 16 below), but completeness would hold if one restricts the result to linear generalizations.

The problems we address in this paper concern the unital anti-unification type and algorithms. We prove that when the term signature contains at least two unital functions, anti-unification is of type zero (nullary) by showing that there exists an AU problem which does not have a minimal complete set of generalizations. Next, we consider two special cases: the linear variant and one-unital fragment and design algorithms for them incrementally: The one-unital fragment algorithm is obtained by extending the rule set used in the linear variant algorithm. The latter uses a modification of rules from [1]. The algorithms are terminating, sound, complete, and return tree grammars from which a set of generalizations can be constructed. For the linear variant, the language of generalizations generated by the grammar is finite. In the one-unital fragment, the language might be infinite, but it contains a finite minimal complete set of generalizations. It follows that both linear and one-unital anti-unification are finitary.

The algorithm for one-unital fragment is further extended for the unrestricted case. It terminates and returns a tree grammar which produces an infinite set of generalizations. It remains to be shown whether this set is always complete or not. At the end of the paper, we also discuss how the nullary type of unital anti-unification might affect the problems in theories that combine U with the properties such as A, C, or I.

Concerning applications, anti-unification has been used for recursion scheme detection in functional programs [4], inductive synthesis of recursive functions [15], learning fixes from software code repositories [3, 14], and for preventing bugs and misconfiguration [11], just to name a few. Given the prominence of algebraic structures, whose equational theory includes unit axioms, in programming language theory, understanding of anti-unification in the presence of such axioms is essential to future progress in this area. As an example of a possible application of this work, modern pure functional programming languages, such as Haskell, heavily rely on monads which are higher-order AU-functions. Clone analysis of code fragments which contain multiple monads used in conjunction would suffer from the nullary type of unital anti-unification. However, restricted procedures, especially for the linear variant, can provide useful substitutes to the less well behaved general procedure. Combining unit axioms with a higher-order term signature was partially address in [9].

The unital anti-unification algorithms described in the paper are implemented and can be accessed at `https://github.com/Ermine516/UnitAU`.

2 Preliminaries

We assume familiarity with the basic notions of unification theory, see, e.g., [2].

Terms and substitutions

We consider a ranked alphabet $\mathcal{A}$, consisting of the set $\mathcal{F}$ of function symbols with fixed arity and the set of variables $\mathcal{V}$. A term t over $\mathcal{A}$ is defined as $t ::= x \mid f(t_1, \dots, t_n)$, where $x \in \mathcal{V}$ and $f \in \mathcal{F}$ with the arity $n \geq 0$. The set of terms over the alphabet $\mathcal{A}$ is denoted by $\mathcal{T}(\mathcal{A})$. Nullary function symbols are called constants. We denote variables by x, y, z, u, v, constants by a, b, c, d, function symbols f, g, h, and terms by s, t, r. We denote the set of variables appearing in a term t by $var(t)$. The depth of a term t is defined inductively as $dep(x) = dep(a) = 1$ for variables and constants, and $dep(f(t_1, \dots, t_n)) = \max\{dep(t_1), \dots, dep(t_n)\}+1$ otherwise. The number of occurrences of s in t is defined inductively as follows $occ(s, s) = 1$, $occ(s, a) = occ(s, x) = 0$ if $x \neq a$ and $s \neq x$, $occ(s, f(t_1, \dots, t_n)) = \sum_i occ(s, t_i)$.

The set of *positions* of a term t, denoted by $pos(t)$, is the set of strings of positive integers, defined as $pos(x) = \{\epsilon\}$ and $pos(f(t_1, \dots, t_n)) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i.p \mid p \in pos(t_i)\}$, where ϵ stands for the empty string. If p is a position in a term s and t is a term, then $s|_p$ denotes the subterm of s at position p and $s[t]_p$ denotes the term obtained from s by replacing the subterm $s|_p$ with t. The *head* of a term t is defined as $head(x) = x$ and $head(f(t_1, \dots, t_n)) = f$.

A *substitution* is a mapping from variables to terms such that all but finitely many variables are mapped to themselves. Lower case Greek letters are used to denote them, except the identity substitution, which is denoted by Id. They are extended to terms in the usual way and we use the postfix notation for that, writing $t\sigma$ for an *instance* of a term t under a substitution σ. The *composition* of substitutions σ and ϑ, written as juxtaposition $\sigma\vartheta$, is the substitution defined as $x(\sigma\vartheta) = (x\sigma)\vartheta$ for all variables x.

The *domain* of a substitution σ is the set of variables which are not mapped to themselves by σ: $dom(\sigma) := \{x \mid x\sigma \neq x\}$. The restriction of σ to a set of variables X, denoted $\sigma|_X$, is the substitution defined as $x(\sigma|_X) = x\sigma$ if $x \in X$ and $x(\sigma|_X) = x$ otherwise.

A *binding* is a pair of a variable and a term, written as $x \mapsto t$. To explicitly write substitutions, we use the standard convention representing a substitution σ as a finite set of bindings $\{x \mapsto x\sigma \mid x \in dom(\sigma)\}$. *Application* of σ to a set of bindings B, written $B\sigma$, is defined as $B\sigma = \{x \mapsto t\sigma \mid x \mapsto t \in B\}$.

Equational anti-unification

Every function symbol f will have an associated set of axioms, denoted by $Ax(f)$. If $Ax(f)$ is empty, then f does not have any associated properties and is called *free*. Otherwise, $Ax(f) \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}, \mathsf{I}\}$ where A is *associativity*, i.e., $f(t_1, f(t_2, t_3)) \equiv f(f(t_1, t_2), t_3)$ for all t_1, t_2, t_3; C is *commutativity*, i.e., $f(t_1, t_2) \equiv f(t_2, t_1)$ for all t_1, t_2; U is *unital*, i.e., $f(t, \epsilon_f) \equiv f(\epsilon_f, t) \equiv t$ for all t, where ϵ_f is the unique unit element associated with the function constant f; and I is *idempotency*, i.e., $f(t, t) \equiv t$ for all t. Note that in these cases, only binary function symbols have equational properties. In the case of unit element, only function constants with arity 0 can be ϵ_f. For each $\mathcal{E} \subseteq \{\mathsf{A}, \mathsf{C}, \mathsf{U}, \mathsf{I}\}$ we denote the equational theory generated by $\mathcal{E}$ by $\approx_{\mathcal{E}}$. For particular equational theories such as U we can denote which function constants have this property, writing, e.g., $\approx_{\mathsf{U}(f,g,\dots)}$. The majority of this paper focuses on *unital equational theories*. However, in later sections we consider combinations between unital theories and the other above mentioned theories.

In the rest of the paper, every non-unital function symbol is free unless otherwise specified.

We say that a term is in *unital normal form* (*U-normal form*) if it does not contain a subterm of the form $f(t, \epsilon_f)$ or $f(\epsilon_f, t)$ for any unital symbol f. To get an U-normal form of a term, all the subterms of the form $f(t, \epsilon_f)$ and $f(\epsilon_f, t)$ are replaced by t repeatedly as long as possible, for each unital symbol f. We write $nf_U(s)$ for the U-normal form of s, and for a set of terms S, $nf_U(S)$ denotes the set $nf_U(S) := \{nf_U(s) \mid s \in S\}$.

A term r is *more general* than s modulo $\mathcal{E}$ (r is an $\mathcal{E}$*-generalization* of s) if there exists a substitution σ such that $r\sigma \approx_{\mathcal{E}} s$. It is written as $r \preceq_{\mathcal{E}} s$. The relation $\preceq_{\mathcal{E}}$ is a quasi-ordering. Its strict part is denoted by $\prec_{\mathcal{E}}$, and the equivalence relation it induces by $\simeq_{\mathcal{E}}$.

Given two terms t and s, and their generalization r, we say that it is their *least general generalization* modulo $\mathcal{E}$ ($\mathcal{E}$-lgg or just lgg in short), if there is no generalization r' of t and s which satisfies $r \prec_{\mathcal{E}} r'$.

A *minimal and complete set of* $\mathcal{E}$*-generalizations* of two terms t and s is the set G with the following three properties:

1. Each element of G is an $\mathcal{E}$-generalization of t and s (soundness of G).

2. For each $\mathcal{E}$-generalization r' of t and s, there exists $r \in G$ such that $r' \preceq_{\mathcal{E}} r$, i.e., r is less general than r' modulo $\mathcal{E}$ (completeness of G).
3. No two distinct elements of G are $\preceq_{\mathcal{E}}$-comparable: If $r_1, r_2 \in G$ such that $r_1 \preceq_{\mathcal{E}} r_2$, then $r_1 = r_2$ (minimality of G).

We write $mcsg_{\mathcal{E}}(t, s)$ for the minimal complete set of $\mathcal{E}$-generalizations of t and s if it exists.

Often we just say generalization, lgg, etc. instead of $\mathcal{E}$-generalization, $\mathcal{E}$-lgg and so on when the equational theory being discussed is clear from context.

The *Anti-unification type* of equational theories are defined similarly (but dually) to unification type, based on the existence and cardinality of a minimal complete set of generalizations. We assume here no restriction on the signature, i.e., the problems and generalizations may contain arbitrary function symbols. Then the types are defined as follows:

- Unitary type: Any anti-unification problem in the theory has a singleton *mcsg*.
- Finitary type: Any anti-unification problem in the theory has an *mcsg* of finite cardinality, for at least one problem having it greater than 1.
- Infinitary type: For any anti-unification problem in the theory there exists an *mcsg*, and for at least one problem this set is infinite.
- Nullary type (or type zero): There exists an anti-unification problem in the theory which does not have an *mcsg*, i.e., every complete set of generalizations for this problem contains two distinct elements such that one is more general than the other.

For each of these types, there exists a corresponding instance of an equational theory. The syntactic first-order anti-unification [12,13] is unitary; commutative anti-unification [1] is finitary; idempotent anti-unification is infinitary [8]; nominal anti-unification with infinitely many atoms is nullary [5,6]. In this paper we illustrate that unital anti-unification is nullary over a term alphabet with as least two unital function symbols, and study anti-unification type for some other theories, which are combined with the unital one.

We represent anti-unification problems in the form of $\mathcal{E}$-*anti-unification triples* ($\mathcal{E}$-AUTs). An $\mathcal{E}$-AUT is a triple of a variable and two terms, written as $x : t \triangleq_{\mathcal{E}} s$. Here x is a fresh variable which stands for the most general $\mathcal{E}$-generalization of t and s. Any $\mathcal{E}$-generalization r of t and s is then an instance of x, witnessed by a substitution σ such that $x\sigma \approx_{\mathcal{E}} r$.

Sometimes, when we want to anti-unify s and t, we simply say that we have an *anti-unification problem* (AUP) modulo $\mathcal{E}$, $s \triangleq_{\mathcal{E}} t$.

In all the notations, we omit $\mathcal{E}$ when it is clear from the context.

Regular tree grammars

A *regular tree grammar* is a tuple $\langle \alpha, N, T, R \rangle$, where the symbol α is called the *axiom*, N is the set of *non-terminal* symbols with arity 0 such that $\alpha \in N$, T is the set of terminal symbols with $T \cap N = \emptyset$, and R is the set of production rules of the form $\beta \mapsto t$ where $\beta \in N$ and $t \in \mathcal{T}(T \cup N)$. Given a regular tree grammar $\mathcal{G} = \langle \alpha, N, T, R \rangle$, the *derivation relation* $\rightarrow_G$ is a relation on pairs of terms of $\mathcal{T}(T \cup N)$ such that $s \rightarrow_G t$ if and only if there exists a position p in s and a rule $\nu \rightarrow r \in R$ such that $s|_p = \nu$ and $t = s[r]_p$. The *language generated by* $\mathcal{G}$ from the nonterminal β is the set of terms $\mathcal{L}(\mathcal{G}, \beta) := \{t \mid t \in \mathcal{T}(T) \text{ and } \beta \rightarrow^+_G t\}$, where $\rightarrow^+_G$ is the transitive closure of the relation $\rightarrow_G$. The language generated by $\mathcal{G}$ is defined as the language generated bt $\mathcal{G}$ from α: $\mathcal{L}(\mathcal{G}) := \mathcal{L}(\mathcal{G}, \alpha)$. Given a grammar $\mathcal{G}$, the set of nonterminals of $\mathcal{G}$ that appear in a syntactic object (term, rule, AUT, etc.) O is denoted by $nter(\mathcal{G}, O)$. For a grammar $\mathcal{G}$, the set of nonterminals that *can be reached* from a nonterminal

ν, denoted by $reach(\mathcal{G},\nu)$, is defined as $reach(\mathcal{G},\nu) := \{\mu \mid \nu \to_{\mathcal{G}}^{*} t$ and $\mu \in nter(\mathcal{G},t)\}$, where $\to_{\mathcal{G}}^{*}$ is reflexive and transitive closure of $\to_{\mathcal{G}}$. When the grammar is clear from the context, we write $\to$ instead of $\to_{\mathcal{G}}$.

Our next step is to connect sets of bindings and regular tree grammars, defining how to construct grammars from binding sets. The reasoning behind such a correspondence is the following: our goal is to represent complete sets of unital generalizations by finite means with the help of regular tree grammars. Hence, we want to develop a U-generalization algorithm which gives us such a representation. The mentioned correspondence will make this task easier, because it will allow us to design a simpler algorithm. It computes a set of bindings, from which one can directly construct the desired grammar, based on the correspondence we define below in Definition 1.

We assume that each nonempty set of bindings B contains a designated binding, which we call the *root binding*. Its left hand side is called *the root* of B. It is required that the root occurs only once in the grammar, in the left hand side of the root binding.

▶ **Definition 1** (Regular tree grammar corresponding to a set of bindings). *Given a (nonempty) set of bindings B, the* corresponding regular tree grammar $\mathcal{G}(B) = \langle \alpha, N, T, B\rangle$ *is defined by the following construction:*

- *The axiom α is the root of B.*
- $N = \{x \mid x \mapsto r \in B$ *for some* $r\}$.
- $T = F \cup V$, *where F is the set of all function symbols that appear in terms of the right hand sides of B, and* $V = \{var(r) \mid x \mapsto r \in B$ *for some* $x\} \setminus N$.

The language of a tree grammar $\mathcal{G}$ is denoted by $\mathcal{L}(\mathcal{G})$.

A motivating Example

Let us consider the term $g(f(a,c),a) \triangleq g(c,b)$ where $Ax(g) = \emptyset$ and $Ax(f) = \{\mathsf{U}\}$. Using the methods discussed in [1] the computed generalization is $g(f(x,c),y)$. This seems reasonable because after decomposing $g(f(a,c),a) \triangleq g(c,b)$ once, we get two AUPs $f(a,c) \triangleq c$ and $a \triangleq b$. The latter is solvable while the former can benefit from a single application of unit introduction, i.e. $f(a,c) \triangleq f(\epsilon_f,c)$, resulting in the AUPs $a \triangleq \epsilon_f$ and $c \triangleq c$. However, if we apply unit introduction to $a \triangleq b$ twice, resulting in $f(a,\epsilon_f) \triangleq f(\epsilon_f,b)$, we can merge variables and get the generalization $g(f(x,c),f(x,y))$ which is less general than $g(f(x,c),y)$. This observation motivated us to investigate the type in greater detail because it seems to imply the possibility of an arbitrary number of variable introductions and merges.

3 General case: unital anti-unification is nullary

We formulate the first main result of this paper: generalization in theories with at least two unital function symbols is of type zero.

In this section all terms are taken from the set $\mathcal{T}(\{f,g,\epsilon_f,\epsilon_g\},\mathcal{V})$, where both f and g are unital with units ϵ_f and ϵ_g respectively. That means, we have no other function symbols except f, g, ϵ_f, and ϵ_g. Furthermore, we will denote generalizations by bold face $\mathbf{g}$.

▶ **Definition 2.** *Let $\mathbf{g}$ be a generalization in U-normal form of $t \triangleq s$. We refer to σ_1 and σ_2 as* generalizing substitutions of $\mathbf{g}$ *if $\mathbf{g}\sigma_1 \approx_U t$, $\mathbf{g}\sigma_2 \approx_U s$, and for every $\{x \mapsto u\} \in \sigma_i$, for $i \in \{1,2\}$, u is in U-normal form.*

▶ **Definition 3.** *Let $\mathbf{g}$ be a generalization in U-normal form of $t \triangleq s$, and let σ_1 and σ_2 be generalizing substitutions. We say that $\mathbf{g}$ is in* reduced form *if the following conditions hold:*

1. *For every* $x \in var(\mathbf{g})$, $x\sigma_1 \not\approx_{\mathsf{U}} x\sigma_2$.
2. *For all* $x, y \in var(\mathbf{g})$ *either* $x = y$, *or for some* $\theta \in \{\sigma_1, \sigma_2\}$, $x\theta \not\approx_{\mathsf{U}} y\theta$.

▶ **Theorem 4.** *There exists a reduced generalization* $\mathbf{g}$ *of* $\epsilon_f \triangleq \epsilon_g$ *such that* $\mathbf{g}$ *is not equal modulo* U *to a variable.*

Proof. Take $\mathbf{g} = f(x, g(x, y))$. Then $\sigma_1 = \{x \mapsto \epsilon_f, y \mapsto \epsilon_g\}$ and $\sigma_2 = \{x \mapsto \epsilon_g, y \mapsto \epsilon_f\}$ are the generalizing substitutions. Obviously, $\mathbf{g}$ is not equal modulo U to a variable. ◀

▶ **Theorem 5.** *Any reduced generalization of* $\epsilon_f \triangleq \epsilon_g$ *is either a variable or contains two distinct variables (maybe with multiple occurrences).*

Proof. Let $\mathbf{g}$ be a reduced generalization of $\epsilon_f \triangleq \epsilon_g$, and σ_1 an σ_2 be generalizing substitutions. If $\mathbf{g}$ is a variable, the theorem trivially holds. By Theorem 4, there exist also nonvariable reduced generalizations of $\epsilon_f \triangleq \epsilon_g$. Notice that for all $x \in var(\mathbf{g})$ we have either (a) $x\sigma_1 = \epsilon_f$ and $x\sigma_2 = \epsilon_g$, or (b) $x\sigma_1 = \epsilon_g$ and $x\sigma_2 = \epsilon_f$, for otherwise either $\mathbf{g}$ would not be a generalization of $\epsilon_f \triangleq \epsilon_g$ (we would be introducing new symbols not occurring in the initial terms), or for some $\{x \mapsto s\} \in \sigma_i$, $i \in \{1, 2\}$, s would not be in U-normal form. If the latter is the case we may just replace the offending binding by $\{x \mapsto s'\}$ where s' is the U-normalized version of s. But since $\mathbf{g}$ is reduced, we do not have two distinct $x, y \in var(\mathbf{g})$ with $x\sigma_i \approx_{\mathsf{U}} y\sigma_i$. Hence, when $\mathbf{g}$ is not a variable, then it must contain two distinct variables: one that satisfies (a), and the other one that satisfies (b). ◀

▶ **Theorem 6.** *For every generalization* $\mathbf{g}$ *in U-normal form of* $\epsilon_f \triangleq \epsilon_g$ *there exists a substitution* ϑ *such that* $\mathbf{g}\vartheta$ *is a reduced generalization of* $\epsilon_f \triangleq \epsilon_g$.

Proof. Let σ_1 and σ_2 be its generalizing substitutions. If $\mathbf{g}$ is reduced, then the theorem trivially holds and $\vartheta = Id$. Assume $\mathbf{g}$ is not in reduced form. (Therefore, it can not be a variable.) We will construct ϑ as a composition of two substitutions ϑ_1 and ϑ_2, which we define below. Since $\mathbf{g}$ is not reduced, it violates one of the two conditions of Definition 3.

If $\mathbf{g}$ does not violate the first condition, we take $\vartheta_1 = Id$ and continue with checking the second one. If $\mathbf{g}$ violates the first condition, then there exists $x \in var(\mathbf{g})$ such that $x\sigma_1 = x\sigma_2$, i.e., x is an overgeneralization. We can assume that $x\sigma_1 = x\sigma_2 = \epsilon_w$, where w is either f or g, because if $dep(x\sigma_1) > 1$, then either $x\sigma_1$ is not in U-normal form or $\mathbf{g}\sigma_1 \not\approx_{\mathsf{U}} \epsilon_w$.

Assume $\{x_1, \ldots, x_n, y_1, \ldots, y_m\} \subseteq var(\mathbf{g})$ are all those variables in $\mathbf{g}$ that violate the first condition of Definition 3 such that $x_i\sigma_1 = x_i\sigma_2 = \epsilon_f$ for all $1 \le i \le n$, and $y_j\sigma_1 = y_j\sigma_2 = \epsilon_g$ for all $1 \le j \le m$. Then we take $z_1, z_2 \notin var(\mathbf{g})$ and consider three substitutions

$$\vartheta_1 = \{x_1 \mapsto g(z_1, z_2)\} \cdots \{x_n \mapsto g(z_1, z_2)\}\{y_1 \mapsto f(z_1, z_2)\} \cdots \{y_m \mapsto g(z_1, z_2)\},$$
$$\sigma_1' = \{z_1 \mapsto \epsilon_f, z_2 \mapsto \epsilon_g\}\sigma_1, \qquad \sigma_2' = \{z_1 \mapsto \epsilon_g, z_2 \mapsto \epsilon_f\}\sigma_2.$$

$\mathbf{g}\vartheta_1$ is a generalization of $\epsilon_f \triangleq \epsilon_g$ and σ_1' and σ_2' are generalizing substitutions, because

$$\mathbf{g}\vartheta_1\sigma_1' = \mathbf{g}\{x_1 \mapsto \epsilon_f\} \cdots \{x_n \mapsto \epsilon_f\}\{y_1 \mapsto \epsilon_g\} \cdots \{y_m \mapsto \epsilon_g\}\sigma_1 = \mathbf{g}\sigma_1 = \epsilon_f.$$
$$\mathbf{g}\vartheta_1\sigma_2' = \mathbf{g}\{x_1 \mapsto \epsilon_f\} \cdots \{x_n \mapsto \epsilon_f\}\{y_1 \mapsto \epsilon_g\} \cdots \{y_m \mapsto \epsilon_g\}\sigma_2 = \mathbf{g}\sigma_2 = \epsilon_g.$$

However, in $\mathbf{g}\vartheta_1$ we do not have variables that violate the first condition of Definition 3: all such variables from $\mathbf{g}$ are now replaced by terms containing z_1 and z_2 only, and these new variables do not violate the condition as one can see from σ_1' and σ_2'.

Hence, we got $\mathbf{g}\vartheta_1$ that does not violate the first condition of Definition 3. If $\mathbf{g}\vartheta_1$ fulfills the second one too, then we take $\vartheta_2 = Id$ and obtain $\vartheta = \vartheta_1$. Otherwise there exist two distinct variables $x, y \in var(\mathbf{g}\vartheta_1)$ such that $x\sigma_i \approx_{\mathsf{U}} y\sigma_i$, $i = 1, 2$. We take the renaming substitution $\{x \mapsto y\}$ and obtain $\mathbf{g}\vartheta_1\{x \mapsto y\}$, which is obviously a generalization again, but replaces the violating variable pair by a single variable. We can repeat this process iteratively for all variable pairs violating the second condition of Definition 3. Let ϑ_2 be the composition of all renaming substitutions used in this process. The obtained generalization $\mathbf{g}\vartheta_1\vartheta_2$ is in reduced form. Taking $\vartheta = \vartheta_1\vartheta_2$ finishes the proof. ◀

From this proof we see that if $\mathbf{g}$ is a reduced generalization of $\epsilon_f \triangleq \epsilon_g$ with variables $var(\mathbf{g}) = \{x, y\}$, then $\sigma_1 = \{x \mapsto \epsilon_f, y \mapsto \epsilon_g\}$ and $\sigma_2 = \{x \mapsto \epsilon_g, y \mapsto \epsilon_f\}$ can be taken as the generalizing substitutions.

▶ **Theorem 7.** *Let $\mathbf{g}$ be a reduced generalization of $\epsilon_f \triangleq \epsilon_g$. Then there exists a reduced generalization $\mathbf{g}'$ of $\epsilon_f \triangleq \epsilon_g$ such that $\mathbf{g} \prec_U \mathbf{g}'$.*

Proof. By Theorem 5, since $\mathbf{g}$ is reduced, it is either a single variable x, or contains exactly two variables x and y.

First assume $\mathbf{g} = x$. Then $\mathbf{g}' = \mathbf{g}\{x \mapsto f(x, g(x, y))\} = f(x, g(x, y))$ is also a reduced generalization of $\epsilon_f \triangleq \epsilon_g$. However, for no θ we have $\mathbf{g}'\theta \approx_{\mathrm{U}} \mathbf{g}$. Hence, $\mathbf{g} \prec_U \mathbf{g}'$ in this case.

Now let $\mathbf{g}$ be such that $\{x, y\} = var(\mathbf{g})$ and $\mathbf{g}' = \mathbf{g}\{x \mapsto f(x, g(x, y))\}$. Furthermore, let $occ(x, \mathbf{g}) = n$ and $occ(y, \mathbf{g}) = m$. Then we get $occ(x, \mathbf{g}') = 2n$ and $occ(y, \mathbf{g}') = n + m$. By the proof of Theorem 5, $n > 0$ and $m > 0$. Assume by contradiction that $\mathbf{g} \not\prec_U \mathbf{g}'$, i.e. there exists $\theta = \{x \mapsto t, y \mapsto s\}$ such that $\mathbf{g}\{x \mapsto f(x, g(x, y))\}\theta = \mathbf{g}$.

If $x \in var(\mathbf{g}'\theta|_x)$ then $x \in var(t)$ implies that $occ(x, \mathbf{g}'\theta|_x) \geq 2n$. Thus, $x \notin var(t)$. This implies that $x \in var(\mathbf{g}'\theta)$ iff $x \in var(s)$. Therefore, $occ(x, \mathbf{g}'\theta) \geq n + m$. On the other hand, $occ(x, \mathbf{g}'\theta) = occ(x, \mathbf{g}) = n$ and from $n \geq n + m$ we get $m = 0$. But it is a contradiction with $m > 0$.

We can apply similar reasoning to the case when $\mathbf{g}' = \mathbf{g}\{y \mapsto f(y, g(y, x))\}$. Hence, $\mathbf{g} \prec_U \mathbf{g}'$ also when $\mathbf{g}$ contains exactly two variables. ◀

▶ **Theorem 8.** *Let $\mathcal{C}$ be a complete set of generalizations of $\epsilon_f \triangleq \epsilon_g$ which are in U-normal form. Then $\mathcal{C}$ contains $\mathbf{g}$ and $\mathbf{g}'$ such that $\mathbf{g} \prec_U \mathbf{g}'$.*

Proof. Let $\mathbf{g} \in \mathcal{C}$. By Theorem 6, $\mathbf{g}\vartheta$ a reduced generalization of $\epsilon_f \triangleq \epsilon_g$ for some ϑ. By Theorem 7 there exists a substitution φ such that $\mathbf{g}\vartheta \prec_U \mathbf{g}\vartheta\varphi$ and $\mathbf{g}\vartheta\varphi$ is a reduced generalization of $\epsilon_f \triangleq \epsilon_g$. By completeness of the set $\mathcal{C}$, there exists a substitution μ such that $\mathbf{g}\vartheta\varphi\mu \in \mathcal{C}$. Taking $\mathbf{g}' = \mathbf{g}\vartheta\varphi\mu$, we get $\mathbf{g}, \mathbf{g}' \in \mathcal{C}$ and $\mathbf{g} \prec_U \mathbf{g}'$. ◀

▶ **Corollary 9.** *Unital anti-unification is nullary.*

Proof. Follows from Theorem 8. ◀

In the rest of the paper we consider two special cases of unital anti-unification for which minimal complete set of generalizations exist, i.e., which are not nullary. These special cases and the linear variant and the fragment with one unital symbol.

4 Linear variant

In linear variant we are looking for unital generalizations in which no variable occurs more than once. Input is not restricted. In particular, the language may contain one or more unital function symbols.

We start by formulating the rules of an algorithm which is supposed to compute linear U-generalizations. The rules transform configurations into configurations. A *configuration* is a quadruple $A; S; L; B$, where A is a set of anti-unification triples to be solved, S is a set of already solved anti-unification triples (called the store), L is a set of pairs of an anti-unification triple and a set of unit elements denoting the start of cycles in B, and B is a set of bindings, representing the generalizations "computed so far". The intuitive idea is to take the obtained B at the end and construct from it a regular tree grammar, from which one can read off each generalization. The set L is not used in the linear variant, but we will need it in later cases when introducing cycles into the constructed grammar. We elaborate on the details later, after the rules are formulated. Configurations are denoted by $\mathbf{C}$.

It is assumed that all terms in A and S are in U-normal form and if $\mathsf{U} \in Ax(f)$ then ϵ_f is the unit element of f. Also, when bindings of the form $\{x \mapsto x\}$ occur in B they will automatically be dropped. The rules are defined as follows ($\uplus$ stands for disjoint union):

Dec: **Decomposition**

$\{x : f(s_1, \dots, s_n) \triangleq f(t_1, \dots, t_n)\} \uplus A;\ S;\ L;\ B \Longrightarrow$
$\quad \{y_1 : s_1 \triangleq t_1, \dots, y_n : s_n \triangleq t_n\} \cup A;\ S;\ L;\ B\{x \mapsto f(y_1, \dots, y_n)\}$

where $n \geq 0$, and $y_1, \dots, y_n$ are fresh variables.

Exp-U-Both: **Expansion for Unit, Both**

$\{x : t \triangleq s\} \uplus A;\ S;\ L;\ B \Longrightarrow$
$\quad \{x_1 : g(t, \epsilon_g) \triangleq s,\ x_2 : g(\epsilon_g, t) \triangleq s,\ y_1 : t \triangleq f(s, \epsilon_f), y_2 : t \triangleq f(\epsilon_f, s)\} \cup A;\ S;\ L;$
$\quad B \cup \{x \mapsto x_1\} \cup \{x \mapsto x_2\} \cup \{x \mapsto y_1\} \cup \{x \mapsto y_2\},$

where $head(t) = f \neq g = head(s)$, $\mathsf{U} \in Ax(f) \cap Ax(g)$, and x_1, x_2, y_1, y_2 are fresh variables.

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

$\{x : t \triangleq f(s_1, s_2)\} \uplus A;\ S;\ L;\ B \Longrightarrow$
$\quad \{x_1 : f(t, \epsilon_f) \triangleq f(s_1, s_2),\ x_2 : f(\epsilon_f, t) \triangleq f(s_1, s_2)\} \cup A;\ S;\ L;\ B \cup \{x \mapsto x_1\} \cup \{x \mapsto x_2\},$

where $f \neq head(t)$, $\mathsf{U} \in Ax(f)$, $\mathsf{U} \notin Ax(head(t))$, and x_1, x_2 are fresh variables.

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

$\{x : f(t_1, t_2) \triangleq s\} \uplus A;\ S;\ L;\ B \Longrightarrow$
$\quad \{x_1 : f(t_1, t_2) \triangleq f(s, \epsilon_f), x_2 : f(t_1, t_2) \triangleq f(\epsilon_f, s)\} \cup A;\ S;\ L;\ B \cup \{x \mapsto x_1\} \cup \{x \mapsto x_2\},$

where $f \neq head(s)$, $\mathsf{U} \in Ax(f)$, $\mathsf{U} \notin Ax(head(s))$, and x_1, x_2 are fresh variables.

Solve: **Solve**

$\{x : s \triangleq t\} \uplus A;\ S;\ L;\ B \Longrightarrow A;\ \{x : s \triangleq t\} \cup S; L;\ B,$

where $head(s) \neq head(t)$ and $\mathsf{U} \notin Ax(head(t)) \cup Ax((head(s)))$.

We denote this set of rules by $\mathcal{R}_{\mathsf{lin}}$. In order to compute linear U-generalizations of two terms t and s, we create an initial configuration $\{x : t \triangleq s\}; \emptyset; \emptyset; \{x_{\text{root}} \to x\}$, where x_{root} and x are fresh variables, and apply the following strategy as long as possible:

- Select an AUT $\mathbf{a}$ arbitrarily from the first component of the configuration.
- Apply a rule in $\mathcal{R}_{\mathsf{lin}}$, applicable to $\mathbf{a}$. (There is only one such rule for each $\mathbf{a}$ in $\mathcal{R}_{\mathsf{lin}}$.)
- If the applied rule is Exp-U-Both, transform all four new AUTs by the Dec rule.
- If the applied rule is Exp-U-L or Exp-U-R, transform both new AUTs by the Dec rule.

This strategy, called Step, will be used in other algorithms below as well. Therefore, we describe it in Algorithm 1. It takes a configuration and an AUT, and returns back a new configuration. In the algorithm, instead of writing "apply rule R to the configuration $\mathbf{C} = A; S; L; B$ with the AUT $\mathbf{a}$ selected in A", we simply write "apply rule R to $\mathbf{a}$".

Algorithm 1 Procedure Step

Require: A configuration $\mathbf{C} = A; S; L; B$ and an AUT $\mathbf{a} = x : t \triangleq s \in A$.

```
if head(t) = head(s) then
    Apply Dec to a, resulting in C'. Update C ← C'
else if ∃f, g ∈ F : (U ∈ (Ax(f) ∩ Ax(g)) ∧ head(s) = f ≠ g = head(t)) then
    Apply Exp-U-Both to a resulting in C' = {a1, a2, a3, a4} ∪ A; S; L; B'
    Apply Dec to a1, a2, a1, a2 resulting in C''. Update C ← C''
else if head(t) ≠ head(s) ∧ ∃f ∈ F : (U ∈ Ax(f) ∧ head(s) = f) then
    Apply Exp-U-L to a resulting in C = {a1, a2} ∪ A; S; L; B'
    Apply Dec to a1, a2 resulting in C''. Update C ← C''
else if head(t) ≠ head(s) ∧ ∃f ∈ F : (U ∈ Ax(f) ∧ head(t) = f) then
    Apply Exp-U-R to a resulting in {a1, a2} ∪ A; S; L; B'
    Apply Dec to a1, a2 resulting in C''. Update C ← C''
else
    Apply Solve to a resulting in C'. Update C ← C'
end if
return C
```

The linear U-generalization algorithm, $\mathfrak{G}_{\mathsf{U\text{-}lin}}$, is then an iterative application of Step, as one can see in Algorithm 2.[1] However, in that work we refrained from using a tree grammar-based procedure. In Example 10 below, we apply $\mathfrak{G}_{\mathsf{U\text{-}lin}}$ to the AUP $x : g(f(a,c),a) \triangleq g(c,b)$ over the alphabet $\{f, g, a, b, c, \epsilon_f\}$, where a, b, and c are constants and g is a binary free function symbol.

Algorithm 2 Procedure $\mathfrak{G}_{\mathsf{U\text{-}lin}}$

Require: A configuration $\mathbf{C} = A; S; L; B$

```
while A ≠ ∅ do
    a ← x : t ≜ s ∈ A
    C ← Step(C, a) (See Algorithm 1)
end while
return C
```

▶ **Example 10.**

$$\{x : g(f(a,c),a) \triangleq g(c,b)\};\ \emptyset;\ \emptyset;\ \{x_{\text{root}} \mapsto x\} \Longrightarrow_{\mathsf{Dec}}$$

$$\{x_1 : f(a,c) \triangleq c, x_2 : a \triangleq b\};\ \emptyset;\ \emptyset;\ \{x_{\text{root}} \mapsto g(x_1,x_2)\} \Longrightarrow_{\mathsf{Exp\text{-}U\text{-}L},\,\mathsf{Dec}\times 2}$$

$$\{x_3 : a \triangleq \epsilon_f, x_4 : c \triangleq c, x_5 : a \triangleq c, x_6 : c \triangleq \epsilon_f, x_2 : a \triangleq b\};\ \emptyset; \emptyset;$$
$$\{x_{\text{root}} \mapsto g(x_1,x_2), x_1 \mapsto f(x_3,x_4), x_1 \mapsto f(x_5,x_6)\} \Longrightarrow_{\mathsf{Dec}}$$

$$\{x_3 : a \triangleq \epsilon_f, x_5 : a \triangleq c, x_6 : c \triangleq \epsilon_f, x_2 : a \triangleq b\};\ \emptyset;\ \emptyset;$$
$$\{x_{\text{root}} \mapsto g(x_1,x_2), x_1 \mapsto f(x_3,c), x_1 \mapsto f(x_5,x_6)\} \Longrightarrow_{\mathsf{Solve}\times 4}$$

$$\emptyset;\ \{x_3 : a \triangleq \epsilon_f, x_5 : a \triangleq c, x_6 : c \triangleq \epsilon_f, x_2 : a \triangleq b\};\ \emptyset;$$
$$\{x_{\text{root}} \mapsto g(x_1,x_2),\ x_1 \mapsto f(x_3,c),\ x_1 \mapsto f(x_5,x_6)\}$$

[1] Linear U-anti-unification is discussed in [9].

We refer to the final binding set as B. Thus, $\mathcal{L}(\mathcal{G}(B)) \approx_U \{g(f(x_3, c), x_2), g(f(x_5, x_6), x_2)\}$. Note that $g(f(x_5, x_6), x_2) \prec_{\mathsf{U}} g(f(x_3, c), x_2)$.

▶ **Theorem 11** (Termination). *The procedure $\mathfrak{G}_{\text{U-lin}}$ is terminating.*

Proof. Let the depth of an AUP be $dep(x : t \triangleq s) = dep(t) + dep(s)$, and the complexity measure of a configuration $A; S; L; B$ be the multiset of depths of AUPs in A. We compare measures by multiset extension of the standard ordering on natural numbers. The extension is well-founded. After each iteration of the loop in Algorithm 2, the complexity measure of $\mathbf{C}$ strictly decreases. Hence, the algorithm terminates. ◀

Termination of $\mathfrak{G}_{\text{U-lin}}$ means that any sequence of rule transformations, starting from the initial configuration, is finite: $\{x : t \triangleq s\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow^* \emptyset; S; L; B$. In the terminal configuration the first component is empty, for otherwise there is always an applicable rule. The set of bindings B at the end is called the $\mathfrak{G}_{\text{U-lin}}$-computed set of bindings.

▶ **Theorem 12** (Soundness). *If $\{x : t \triangleq s\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow^* \emptyset; S; L; B$ is a transformation sequence of $\mathfrak{G}_{\text{U-lin}}$, then for every $r \in \mathcal{L}(\mathcal{G}(B))$, $r \preceq_U t$ and $r \preceq_U s$.*

Proof. We can prove soundness by induction over the length of the derivation, based on the fact that if $\mathcal{L}(\mathcal{G}(B))$ is a set of generalizations of an AUT $x : t \triangleq s$ and $\{x : t \triangleq s\} \cup A; S; L; B \Longrightarrow A'; S'; L'; B'$ is a transformation step, then $\mathcal{L}(\mathcal{G}(B'))$ is also a set of generalizations of $x : t \triangleq s$. For a transformation with Dec rule the proof of this property is standard. For Solve rule it is obvious. For the expansion rules it follows from two facts: first, B' is obtained from B by bindings of a variable to a variable (e.g., x to x_1) and second, all new AUTs obtained by these rules are U-equivalent to the original one (e.g., an AUT whose generalization is x_1 is U-equivalent to the AUT whose generalization was x). ◀

For the set B computed by the procedure, we call $\mathcal{L}(\mathcal{G}(B))$ the *set of generalizations computed by* $\mathfrak{G}_{\text{U-lin}}$.

▶ **Theorem 13** (Completeness of $\mathfrak{G}_{\text{U-lin}}$). *Let s be a linear U-generalization of two terms t_1 and t_2. Then there exists a transformation sequence $\{x : t_1 \triangleq t_2\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow^* \emptyset; S; L; B$ in $\mathfrak{G}_{\text{U-lin}}$ such that for some term $r \in \mathcal{L}(\mathcal{G}(B))$, $s \preceq_U r$.*

Proof. See Appendix A. ◀

▶ **Theorem 14.** *The set $\mathcal{L}(\mathcal{G}(B))$ computed by $\mathfrak{G}_{\text{U-lin}}$ is finite for any input.*

Proof. At every step of $\mathfrak{G}_{\text{U-lin}}$ only one of the inference rules is applicable to the current configuration. None of the rules used in the $\mathfrak{G}_{\text{U-lin}}$ procedure introduce cycles into the grammar. Thus, the final set of bindings produces a tree grammar with a finite language. ◀

▶ **Theorem 15.** *Linear unital anti-unification is finitary.*

Proof. By Theorem 13 & 14. ◀

5 One-unital fragment

The next special case of U-anti-unification allows arbitrary generalizations (not only linear ones), but takes input from a language with only one unital function. We call this special case a one-unital fragment, and the corresponding alphabet one-unital alphabet.

Lifting the linearity restriction leads to an extension of the rule system. If two variables generalize the same AUTs, they should be merged. Besides, cycles should be permitted in the grammar. These changes are reflected in the set of rules $\mathcal{R}_{\mathsf{one(f)}}$ given below. They will be used together with the $\mathcal{R}_{\mathsf{lin}}$ rules to solve generalization problems with one unital symbol.

One will probably notice that the cycle rules allows the construction of a grammar with an infinite language, however, as shown in Theorem 20, only a finite number of these terms are least general generalization. In some sense the cycle rules allow for the construction of more expressive tree grammars than necessary for finding the minimal complete set of generalizations. It is reasonable to expect that less expressive versions of the rules may be developed specifically for the one-unital fragment. However, as presented here we highlight the relationship between this fragment and the algorithm we present for the general procedure. Essentially in the one-unital fragment only a finite portion of the terms generated by the cycles are least general generalizations where in the general case all the terms resulting from a cycle may be ordered by generality.

Start-Cycle-U: **Cycle introduction for Unit**

$\{x : t \triangleq s\} \uplus A;\ S;\ L;\ B \Longrightarrow \{y_1 : f(t, \epsilon_f) \triangleq f(\epsilon_f, s),\ y_2 : f(\epsilon_f, t) \triangleq f(s, \epsilon_f), y_3 : t \triangleq s\} \cup A;$
$\quad S;\ \{(\{x : t \triangleq s\}, \{\epsilon_f\})\} \cup L;\ B \cup \{x \mapsto y_1\} \cup \{x \mapsto y_2\},$

where $\mathsf{U} \in Ax(f)$, $(\{y : t \triangleq s\}, Un) \notin L$ for any y and Un, $head(t) \neq \epsilon_f$ or $head(s) \neq \epsilon_f$, $\mathsf{U} \notin Ax(head(t)) \cup Ax(head(s))$, and y_1 and y_2 are fresh variables.

Sat-Cycle-U: **Cycle Saturation for Unit**

$\{x : t \triangleq s\} \uplus A;\ S;\ \{(\{y : t \triangleq s\}, Un)\} \cup L;\ B \Longrightarrow$
$\quad \{x : t \triangleq s\} \cup A;\ S;\ (\{y : t \triangleq s\}, Un) \cup L;\ B\{x \mapsto y\} \cup \{y \mapsto x\},$

where $x \neq y$ and $\{y \mapsto x\} \notin B$.

Merge: **Merge**

$\emptyset;\ \{x_1 : s_1 \triangleq t_1,\ x_2 : s_2 \triangleq t_2\} \uplus S;\ L;\ B \Longrightarrow \emptyset;\ \{x_1 : s_1 \triangleq t_1\} \cup S;\ L;\ B\{x_2 \mapsto x_1\},$

where $s_1 \approx_{\mathsf{U}} s_2$ and $t_1 \approx_{\mathsf{U}} t_2$.

For a given AUT, the Start-Cycle-U rule adds two new AUTs, which are U-equivalent to the given one. The original AUT is still present, just with a renamed generalization variable. It will be used for saturation. In Algorithm 3, we define a strategy for applying the new cycle rules. We 'exhaustively' (see line 6) apply Sat-Cycle-U because applying Dec to the AUPs resulting from Start-Cycle-U may result in AUPs present in the cycle set L.

Algorithm 3 Procedure Cycle($\mathbf{C}, \mathbf{a}$)

Require: A configuration $\mathbf{C} = A; S; L; B$, an AUT $\mathbf{a} = x : t \triangleq s$
1: **if** $\exists f \in \mathcal{F} : (\mathsf{U} \in Ax(f) \wedge (\{y : t \triangleq s\}, Un) \notin L)$ **then**
2: Apply Start-Cycle-U to $\mathbf{a}$ resulting in $\mathbf{C}' = \{\mathbf{a}_1, \mathbf{a}_2, x' : t \triangleq s\} \cup A; S; L'; B'$
3: Apply Dec to $\mathbf{a}_1, \mathbf{a}_2$ resulting in $\mathbf{C}''$. Update $\mathbf{C} \leftarrow \mathbf{C}''$ and $\mathbf{a} \leftarrow x' : t \triangleq s$
4: **end if**
5: Exhaustively apply Sat-Cycle-U to $\mathbf{C}$ resulting in $\mathbf{C}^*$. Update $\mathbf{C} \leftarrow \mathbf{C}^*$
6: **return** $(\mathbf{C}, \mathbf{a})$

The one-unital-function anti-unification algorithm $\mathfrak{G}_{\mathsf{U(f)}}$ is a strategy of applying the rules in $\mathcal{R}_{\mathsf{lin}} \cup \mathcal{R}_{\mathsf{one(f)}}$ as defined in Algorithm 4.

▶ **Example 16.** Observe that the AUP addressed in Example 10 is solved over an alphabet with a single unital function symbol. Now we try to solve it using $\mathfrak{G}_{\mathsf{U(f)}}$.

$$\{x : g(f(a, c), a) \triangleq g(c, b)\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow_{\mathsf{Start\text{-}Cycle\text{-}U}}$$

Algorithm 4 Procedure for $\mathfrak{G}_{\mathsf{U}(f)}$

Require: A configuration $\mathbf{C} = A; S; L; B$
  while $A \neq \emptyset$ **do**
    $\mathbf{a} \leftarrow x : t \triangleq s \in A$
    $(\mathbf{C}, \mathbf{a}) \leftarrow \mathsf{Cycle}(\mathbf{C}, \mathbf{a})$ (See Algorithm 3)
    $\mathbf{C} \leftarrow \mathsf{Step}(\mathbf{C}, \mathbf{a})$ (See Algorithm 1)
    Exhaustively apply Sat-Cycle-U to $\mathbf{C}$ resulting in $\mathbf{C}^*$. Update $\mathbf{C} \leftarrow \mathbf{C}^*$
  end while
  Exhaustively apply Merge to $\mathbf{C}$ resulting in $\mathbf{C}^*$. Update $\mathbf{C} \leftarrow \mathbf{C}^*$
  return $\mathbf{C}$

$$\{x_1 : g(f(a,c),a) \triangleq g(c,b), x_2 : f(g(f(a,c),a),\epsilon_f) \triangleq f(\epsilon_f, g(c,b)),$$
$$x_3 : f(\epsilon_f, g(f(a,c),a)) \triangleq f(g(c,b),\epsilon_f)\}; \emptyset; \{(x : g(f(a,c),a) \triangleq g(c,b), \{\epsilon_f\})\};$$
$$\{x_{\text{root}} \mapsto x, x \mapsto x_2, x \mapsto x_3\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{x_1 : g(f(a,c),a) \triangleq g(c,b), x_2 : f(g(f(a,c),a),\epsilon_f) \triangleq f(\epsilon_f, g(c,b)), x_4 : \epsilon_f \triangleq g(c,b),$$
$$x_5 : g(f(a,c),a) \triangleq \epsilon_f\}; \emptyset; \{(x : g(f(a,c),a) \triangleq g(c,b), \{\epsilon_f\})\};$$
$$\{x_{\text{root}} \mapsto x, x \mapsto x_2, x \mapsto f(x_4, x_5)\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{x_1 : g(f(a,c),a) \triangleq g(c,b), x_4 : \epsilon_f \triangleq g(c,b), x_5 : g(f(a,c),a) \triangleq \epsilon_f,$$
$$x_6 : g(f(a,c),a) \triangleq \epsilon_f, x_7 : \epsilon_f \triangleq g(c,b)\}; \emptyset; \{(x : g(f(a,c),a) \triangleq g(c,b), \{\epsilon_f\})\};$$
$$\{x_{\text{root}} \mapsto x, x \mapsto f(x_6, x_7), x \mapsto f(x_4, x_5)\}, \Longrightarrow_{\mathsf{Sat\text{-}Cycle\text{-}U}}$$
$$\{x_1 : g(f(a,c),a) \triangleq g(c,b), x_4 : \epsilon_f \triangleq g(c,b), x_5 : g(f(a,c),a) \triangleq \epsilon_f,$$
$$x_6 : g(f(a,c),a) \triangleq \epsilon_f, x_7 : \epsilon_f \triangleq g(c,b)\}; \emptyset; \{(x : g(f(a,c),a) \triangleq g(c,b), \{\epsilon_f\})\};$$
$$\{x_{\text{root}} \mapsto x, x \mapsto f(x_6, x_7), x \mapsto f(x_4, x_5), x \mapsto x_1\} \Longrightarrow_{\mathsf{Dec}}$$
$$\{x_4 : \epsilon_f \triangleq g(c,b), x_5 : g(f(a,c),a) \triangleq \epsilon_f, x_6 : g(f(a,c),a) \triangleq \epsilon_f, x_7 : \epsilon_f \triangleq g(c,b),$$
$$x_8 : f(a,c) \triangleq c, x_9 : a \triangleq b\}; \emptyset; \{(x : g(f(a,c),a) \triangleq g(c,b), \{\epsilon_f\})\};$$
$$\{x_{\text{root}} \mapsto x, x \mapsto f(x_6, x_7), x \mapsto f(x_4, x_5), x \mapsto g(x_8, x_9)\} \Longrightarrow_{\mathsf{Start\text{-}Cycle\text{-}U}}$$
$$\dots$$
$$\emptyset; \{x_{10} : \epsilon_f \triangleq g(c,b), x_{17} : g(f(a,c),a) \triangleq \epsilon_f, x_{33} : a \triangleq b, x_{40} : \epsilon_f \triangleq c, x_{76} : \epsilon_f \triangleq b,$$
$$x_{83} : a \triangleq \epsilon_f, x_{146} : a \triangleq c, x_{153} : c \triangleq \epsilon_f\}; L; \{x_{\text{root}} \mapsto x,\ x \mapsto g(f(x_{28}, x_{61}), f(x_{83}, x_{76})),$$
$$x \mapsto g(f(x_{83}, f(x_{153}, x_{40})), x_{33}), \dots, \underline{x \mapsto g(f(x_{83}, c), f(x_{76}, x_{83}))},$$
$$x \mapsto g(f(x_{70}, x_{28}), x_{33}), x \mapsto g(f(x_{83}, f(x_{40}, x_{153})), x_{33}), \dots,$$
$$x \mapsto g(f(x_{61}, x_{28}), f(x_{76}, x_{83})), \underline{x \mapsto g(f(x_{83}, c), f(x_{83}, x_{76}))}, \dots\}.$$

The complete derivation contains 217 rule applications. Here we skipped most of them. The final binding set, after removing useless bindings, has 26 bindings together with a single non-terminal.[2] However, the majority of the generalizations contained in the language of this grammar are comparable. We underline the two incomparable generalizations produced by the algorithm, and refer to them as $\mathbf{g}_1$ and $\mathbf{g}_2$. In fact, the set $\{\mathbf{g}_1, \mathbf{g}_2\}$ forms $mcsg_{\mathsf{U}}(g(f(a,c),a), g(c,b))$.[3] Observe that they are *less general* than the terms computed in

[2] See Section C for the grammar generated by our implementation.
[3] The algorithm in [1] computes generalizations that are more general than $\mathbf{g}_1$ and $\mathbf{g}_2$.

Example 10, indicating that the expansion rules are not enough to construct all non-linear generalizations even when only one function symbol is unital. We did not even need the Merge rule to obtain those *nonlinear* generalizations. The cycle rules created them.

▶ **Theorem 17** (Termination). *$\mathfrak{G}_{\mathsf{U(f)}}$ is terminating for AUPs over an one-unital alphabet.*

Proof. To a given AUT, Cycle can apply only once, because afterwards the AUT is put in the set L. To each of the AUTs obtained by the application of the Start-Cycle-U the same rule can apply again at most once, since the further obtained AUTs are either of the form $x : \epsilon_f \triangleq \epsilon_f$, or are already placed in L. The saturation rule applies once to each element in L. Hence, the cycle rules can apply only finitely many times. The other rules strictly decrease the measure as defined in the proof of Theorem 11. It implies that $\mathfrak{G}_{\mathsf{U(f)}}$ terminates. ◀

▶ **Theorem 18** (Soundness). *If $\{x : t \triangleq s\}; \emptyset; \emptyset; \{x_{\mathrm{root}} \mapsto x\} \Longrightarrow^* \emptyset; S; L; B$ is a transformation sequence of $\mathfrak{G}_{\mathsf{U(f)}}$ for AUPs over an one-unital alphabet, then for every $g \in \mathcal{L}(\mathcal{G}(B))$, $g \preceq_U t$ and $g \preceq_U s$.*

Proof. Similar to Theorem 12. For the cycle rules, the argument is the same as for the expansion rules. ◀

The notion of computed grammar is defined for $\mathfrak{G}_{\mathsf{U(f)}}$ in the same way as for $\mathfrak{G}_{\mathsf{U\text{-}lin}}$.

▶ **Theorem 19** (Completeness of $\mathfrak{G}_{\mathsf{U(f)}}$). *Let t_1, t_2, and s be terms over an one-unital alphabet such that s is a U-generalization of t_1 and t_2. Then there exists a transformation sequence $\{x : t_1 \triangleq t_2\}; \emptyset; \emptyset; \{x_{\mathrm{root}} \mapsto x\} \Longrightarrow^* \emptyset; S; L; B$ using the procedure $\mathfrak{G}_{\mathsf{U(f)}}$ such that for some term $r \in \mathcal{L}(\mathcal{G}(B))$, $s \preceq_U r$.*

Proof. We assume that t_1, t_2, and s are in U-normal form. We prove the theorem by induction on $dep(t_1) + dep(t_2)$ which we denote by n. Furthermore we will denote the unital function by f and its unit by ϵ_f.

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

a) The case $dep(s) = 1$ is handled in a similar way as case 1 a) of the proof of Theorem 13.

b) Now assume as the induction hypothesis that for every generalization s of t_1 and t_2 of depth at most k, either $s \preceq_{\mathsf{U}} t_1$ and $t_1 = t_2$, or $s \preceq_{\mathsf{U}} x$ and $t_1 \neq t_2$. We show that this holds for a generalization s' of depth $k+1$. By our assumptions, $s' = f(s_1, s_2)$ for some terms s_1 and s_2.

Let σ_1 and σ_2 be substitutions 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, by the induction hypothesis, $s_2 \preceq_{\mathsf{U}} t_1$ (resp., $s_1 \preceq_{\mathsf{U}} t_1$) when $t_1 = t_2$, or $s_2 \preceq_{\mathsf{U}} x$ (resp., $s_1 \preceq_{\mathsf{U}} x$) when $t_1 \neq t_2$. Without loss of generality, this implies that for every $x \in var(s_1)$, $x\sigma_1 = x\sigma_2 = \epsilon_f$, being that f is the only unital function. Thus, there exists a substitution ϑ such that $s_1\vartheta = \epsilon_f$ and $s_2\vartheta \approx_{\mathsf{U}} s_2'$ where s_2' is still a generalization of t_1 and t_2, i.e., $s'\vartheta = s_2'$ or $s' \prec_{\mathsf{U}} s_2'$.

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 the former case, without loss of generality.

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 $s_1 \preceq_{\mathsf{U}} t_1$ and $s_2 \preceq_{\mathsf{U}} t_1$. If $t_1 \neq t_2$ then we need to make a distinction:

b1. If neither t_1 nor t_2 is ϵ_f, 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 $y'\sigma_2 = t_2$. Note that if either t_1 or t_2 occurs in s' then s' is not a generalization $t_1 \triangleq t_2$. Let use assume that either y or y' occurs in s_2 or s_1, respectively. without loss of generality we assume that y occurs in s_2. However, this would imply that $s_2\sigma_1 = t_1$ resulting in the term $f(t_1, t_1)$ unless

$t_1 = \epsilon_f$, which contradicts our assumptions. Thus, y cannot occur in s_2. 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, $y\sigma'_1 = t_1$, $y\sigma'_2 = t_2$, $y'\sigma'_1 = y\sigma'_2 = \epsilon_f$. This implies that s_1 is a generalization of $t_1 \triangleq t_2$ which has depth $< k+1$. Thus, $s_1 \preceq_{\mathsf{U}} x$.

b2. Either t_1 or t_2 is ϵ_f. The proof is similar to the case b1 by showing that the variable generalizing the term which is not equivalent to ϵ_f cannot occur in both s_1 and s_2.

Case 2: $n > 2$.

a) Assume that $t_1 = g(w_1, \dots, w_m)$ and $t_2 = g(r_1, \dots, r_m)$, such that $\mathsf{U} \notin \mathsf{Ax}(g)$. Then $\mathfrak{G}_{\mathsf{U(f)}}$ performs the following rule applications to the initial configuration:

$$\{x : t_1 \triangleq t_2\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow_{\textsf{Start-Cycle-U},\ (\textsf{Dec}\times 2)}$$
$$\{x_1 : t_1 \triangleq \epsilon_f, x_2 : \epsilon_f \triangleq t_2, y_1 : t_1 \triangleq \epsilon_f, y_2 : \epsilon_f \triangleq t_2, x_3 : t_1 \triangleq t_2\}; \emptyset;$$
$$\{(x : t_1 \triangleq t_2, \{\epsilon_f\})\}; \{x_{\text{root}} \mapsto x, x \mapsto f(x_1, x_2), x \mapsto f(y_2, y_1)\} \Longrightarrow_{\textsf{Sat-Cycle-U}}$$
$$\{x_1 : t_1 \triangleq \epsilon_f, x_2 : \epsilon_f \triangleq t_2, y_1 : t_1 \triangleq \epsilon_f, y_2 : \epsilon_f \triangleq t_2, x_3 : t_1 \triangleq t_2\}; \emptyset;$$
$$\{(x : t_1 \triangleq t_2, \{\epsilon_f\})\}; \{x_{\text{root}} \mapsto x, x \mapsto f(x_1, x_2), x \mapsto f(x_2, x_1), x \mapsto x_3\} \Longrightarrow_{\textsf{Dec}}$$
$$\{x_1 : t_1 \triangleq \epsilon_f, x_2 : \epsilon_f \triangleq t_2, y_1 : t_1 \triangleq \epsilon_f, y_2 : \epsilon_f \triangleq t_2, z_1 : w_1 \triangleq r_1, \dots, z_m : w_1 \triangleq r_m\};$$
$$\emptyset; \{(x : t_1 \triangleq t_2, \{\epsilon_f\})\}; \{x_{\text{root}} \mapsto x, x \mapsto f(x_1, x_2), x \mapsto f(x_2, x_1), x \mapsto g(z_1, \dots, z_m)\}$$

The case when $s = g(s_1, \dots, s_m)$ is handled in a similar fashion as in case 2a) of the proof of Theorem 13, though we may need to apply additional Merges.
If $s = f(s_1, s_2)$ then it may be the case, without loss of generality, that s_1 generalizes $t_1 \triangleq \epsilon_f$ and s_2 generalizes $\epsilon_f \triangleq t_2$. This case may also be handled in a similar fashion as in case 2a) of the proof of Theorem 13, though we may need to apply additional Merges.
The final case to consider is $s = f(s_1, s_2)$ and, without loss of generality, s_2 generalizes $\epsilon_f \triangleq \epsilon_f$. This implies that for all $x \in var(s_2)$, $x\sigma_1 = x\sigma_2 = \epsilon_f$. Similar to case 1b) above we can reconstruct the substitutions such that $s \preceq_{\mathsf{U}} s_1$.

b) Assume that $t_1 = f(w_1, w_2)$ and $t_2 = f(r_1, r_2)$, such that $\mathsf{U} \in \mathsf{Ax}(f)$. We can proceed in a similar fashion as in case 2a).

c) Assume that $t_i = f(w_1, w_2)$ and $t_{(i+1 \ mod\ 2)} = g(r_1, \dots, r_k)$, where $i \in \{1, 2\}$. we can proceed in a similar fashion as in case 2b) except that we apply Exp-U-Both, Exp-U-L or Exp-U-R prior to applying Dec.

◀

▶ **Theorem 20.** *The set $\mathcal{L}(\mathcal{G}(B))$ computed by $\mathfrak{G}_{\mathsf{U(f)}}$ contains only finitely many incomparable generalizations.*

Proof. Notice that in case 1 of Theorem 19 only one generalization exists for a given AUP whose left and right term are constant. In case 2 of Theorem 19 we show that the generalizations of a given AUP can be constructed from the generalizations of the direct subterms. The only point which makes reference to possibly infinite chains of generalizations comes at the end of case 2a). However, it was shown that this case is degenerate. Thus, we can redo the inductive construction of Theorem 19 to prove that $\mathcal{L}(\mathcal{G}(B))$ contains only finitely many non-comparable generalizations. To show that it is not unitary we need only to consider the $f(a, a) \triangleq a$ where $\mathsf{U} \in Ax(f)$, which has two generalizations. ◀

▶ **Theorem 21.** *Anti-unification over an one-unital alphabet is finitary.*

Proof. By Theorem 19 & 20. ◀

A problem one might have noticed concerning $\mathfrak{G}_{\mathsf{U(f)}}$ is that the computed bindings produce a verbose grammar. Most of the generalizations in the language of the grammar are comparable. However, prior to termination, it is not clear which paths may be pruned from the search. The binding set produced by $\mathfrak{G}_{\mathsf{U(f)}}$ almost always produces a tree grammar with an infinite language which contains a finite set of incomparable generalizations. Possible ways of pruning need further investigations.

6 An algorithm for unrestricted unital anti-unification

The unrestricted case generalizes one-unital anti-unification by permitting more than one unital symbol. To accommodate them in cycles, we need an extra rule, which resembles to Start-Cycle-U in that it extends the set L, but only for AUTs already existing there, by adding a new unit element.

Branch-Cycle-U: **Branching Cycle for Unit**

$$\{x : t \triangleq s\} \uplus A;\ S;\ \{(\{y : t \triangleq s\}, Un)\} \cup L;\ B \Longrightarrow$$
$$\{y_1 : f(t, \epsilon_f) \triangleq f(\epsilon_f, s), y_2 : f(\epsilon_f, t) \triangleq f(s, \epsilon_f), y_3 : t \triangleq s\} \cup A;\ S;$$
$$\{(\{y : t \triangleq s\}, \{\epsilon_f\} \cup Un)\} \cup L;\ B\{x \mapsto y\} \cup \{y \mapsto y_1\} \cup \{y \mapsto y_2\},$$

where $\mathsf{U} \in Ax(f)$, $\epsilon_f \notin Un$, $head(t) \neq \epsilon_f$ or $head(s) \neq \epsilon_f$, $\mathsf{U} \notin Ax(head(t)) \cup Ax(head(s))$, and y_1 and y_2 are fresh variables.

We get the set of all rules for unital generalization $\mathcal{R}_{\mathsf{U}} := \mathcal{R}_{\mathsf{lin}} \cup \mathcal{R}_{\mathsf{one(f)}} \cup \{\text{Branch-Cycle-U}\}$, and the procedure that is based on them is denoted by $\mathfrak{G}_{\mathsf{U}}$. It is formulated in Algorithm 5.

Algorithm 5 Procedure $\mathfrak{G}_{\mathsf{U}}$

Require: A configuration $\mathbf{C} = A; S; L; B$
while $A \neq \emptyset$ **do**
  $\mathbf{a} \leftarrow x : t \triangleq s \in A$
  $(\mathbf{C}, \mathbf{a}) \leftarrow \mathsf{Cycle}(\mathbf{C}, \mathbf{a})$ (See Algorithm 3)
  if $\exists f \in \mathcal{A} : (\mathsf{U} \in Ax(f) \wedge (\{y : t \triangleq s\}, Un) \in L \wedge \epsilon_f \notin Un)$ **then**
    repeat
      Apply Branch-Cycle-U to $\mathbf{a}$ resulting in $\mathbf{C}' = \{\mathbf{a}_1, \mathbf{a}_2, x' : t \triangleq s\} \cup A; S; L'; B'$
      Apply Dec to $\mathbf{a}_1, \mathbf{a}_2$ resulting in $\mathbf{C}''$. Update $\mathbf{C} \leftarrow \mathbf{C}''$ and $\mathbf{a} \leftarrow x' : t \triangleq s$
      Exhaustively apply Sat-Cycle-U to $\mathbf{C}$ resulting in $\mathbf{C}^*$. Update $\mathbf{C} \leftarrow \mathbf{C}^*$
    until $\forall f \in \mathcal{A} : (\mathsf{U} \in Ax(f) \wedge (\{y : t \triangleq s\}, Un) \in L) \Rightarrow \epsilon_f \in Un)$
  end if
  $\mathbf{C} \leftarrow \mathsf{Step}(\mathbf{C}, \mathbf{a})$ (See Algorithm 1)
  Exhaustively apply Sat-Cycle-U to $\mathbf{C}$ resulting in $\mathbf{C}^*$. Update $\mathbf{C} \leftarrow \mathbf{C}^*$
end while
Exhaustively apply Merge to $\mathbf{C}$ resulting in $\mathbf{C}^*$. Update $\mathbf{C} \leftarrow \mathbf{C}^*$
return $\mathbf{C}$

Note that at each step in the procedures outlined in Algorithms 2, 4, and 5, there is only one rule applicable to the current configuration. Thus, each procedure produces a single tree grammar whose language is the computed generalizations of the initial AUP. Termination and soundness of $\mathfrak{G}_{\mathsf{U}}$ depends on termination and soundness of Branch-Cycle-U, which can be established similarly to Start-Cycle-U. Completeness of $\mathfrak{G}_{\mathsf{U}}$ needs further study.

▶ **Theorem 22.** *The algorithm $\mathfrak{G}_{\mathsf{U}}$ is terminating and sound.*

We have seen in Section 3 that unital anti-unification with two unital symbols is nullary, based on the AUPs $\epsilon_f \triangleq \epsilon_g$. Such AUPs can be generated with the help of Branch-Cycle-U even from such trivial problems as, e.g., $a \triangleq a$.

7 Combined theories

In this section we consider the combination of unit element theories with other common equational theories such as A (Associativity), C (Commutativity), and I (Idempotency).

Observe that the anti-unification problems used to prove Theorem 9, i.e., $\epsilon_f \triangleq \epsilon_g$ and $\epsilon_g \triangleq \epsilon_f$, are still problematic when considering the combined theories CU, AU, ACU. For example, modulo CU, AU, and ACU, $f(x, g(x, y)) \not\preceq_u x$, for $u \in \{\mathsf{CU}, \mathsf{AU}, \mathsf{ACU}\}$, when $\mathsf{U} \in Ax(f)$ and $\mathsf{U} \in Ax(g)$. Thus, the argument outlined in Section 3 still applies to these cases. However, for UI we have $f(x, g(x, y)) \preceq_{\mathsf{UI}} x$, i.e., $f(x, g(x, y))\{y \mapsto x\} = f(x, g(x, x)) \preceq_{\mathsf{UI}} f(x, x) \preceq_{\mathsf{UI}} x$ where $\mathsf{U}, \mathsf{I} \in Ax(f)$ and $\mathsf{U}, \mathsf{I} \in Ax(g)$. Thus, our proof of nullarity for unital theories cannot be extended to UI. As it was shown in [8], a theory with a single idempotent function is infinitary if there is an AUP with a so called base set of generalizations of size at least two. It is not completely clear that a similar result will hold for UI and ACUI.

Concerning the special cases, since C, A, and AC are finitary [1], we expect that their linear variant and one-unital fragment remain finitary, despite the fact that the existing algorithms are not based on the tree grammar representation and would require reworking. This can be done in a straightforward manner similar to our handling of the U-decomposition rules we define above. When using the tree grammar formulation described in this paper or as described in [8], one either needs to describe how to join tree grammars as in [8], or write rules in such a way that all possibilities are exhausted by a single rule application. Notice the U-decomposition rules introduce all possible decompositions modulo U into the configuration. The existing rules for C, A, and AC can be adjusted to our framework in a similar way, i.e., it would require writing a rule which adds all decomposition paths simultaneously to the current configuration.

8 Discussion

In this work we showed that unital anti-unification is of type zero. We also distinguished two cases the problem is finitary: linear variant and one-unital fragment. We provided procedures for solving those special cases, and proved their termination, soundness, and completeness. Besides, we provide a terminating and sound general procedure for computing unrestricted unital generalizations. These procedures are based on tree grammar construction in a similar fashion as in earlier work on idempotent equational theories [8]. We also briefly discussed generalization type in combined theories such as CU, AU, ACU, ACUI, and UI.

We end the paper with the following list of open questions:

- Is the general procedure $\mathfrak{G}_{\mathsf{U}}$ complete for arbitrary unital theories?
- Modify the one-unital procedure $\mathfrak{G}_{\mathsf{U(f)}}$ so that it produces less verbose tree grammars.
- Can the rules outlined in [1] be joined with the rules from $\mathcal{R}_{\mathsf{one(f)}}$ to produce minimal complete procedures for restrictions of CU, AU, ACU.
- Are unrestricted ACUI and UI infinitary or nullary?
- Can the techniques used here and [8] be generalized to AU for any collapse theory?
- Are there non-trivial collapse theories with unitary or finitary AU type?

References

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

2 Franz Baader and Wayne Snyder. Unification theory. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445 – 533. North-Holland, Amsterdam, 2001. doi:10.1016/B978-044450813-3/50010-2.

3 Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. Getafix: learning to fix bugs automatically. *Proc. ACM Program. Lang.*, 3(OOPSLA):159:1–159:27, 2019. URL: https://doi.org/10.1145/3360585, doi:10.1145/3360585.

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

5 Alexander Baumgartner. *Anti-Unification Algorithms: Design, Analysis, and Implementation.* PhD thesis, Johannes Kepler University Linz, 2015. Available from http://www.risc.jku.at/publications/download/risc_5180/phd-thesis.pdf.

6 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal anti-unification. In Maribel Fernández, editor, *RTA 2015, June 29 to July 1, 2015, Warsaw, Poland*, volume 36 of *LIPIcs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.RTA.2015.57.

7 Armin Biere. Normalisation, unification and generalisation in free monoids. Master's thesis, University of Karlsruhe, 1993. (in German).

8 David Cerna and Temur Kutsia. Idempotent anti-unification. *ACM Trans. Comput. Logic*, 21(2), November 2019. doi:10.1145/3359060.

9 David M. Cerna and Temur Kutsia. Higher-order pattern generalization modulo equational theories. *Mathematical Structures in Computer Science.* Accepted.

10 Stefan Kühner, Chris Mathis, Peter Raulefs, and Jörg H. Siekmann. Unification of idempotent functions. In Raj Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, page 528. William Kaufmann, 1977. URL: http://ijcai.org/Proceedings/77-1/Papers/092.pdf.

11 Sonu Mehta, Ranjita Bhagwan, Rahul Kumar, Chetan Bansal, Chandra Maddila, B. Ashok, Sumit Asthana, Christian Bird, and Aditya Kumar. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 435–448, Santa Clara, CA, February 2020. USENIX Association. URL: https://www.usenix.org/conference/nsdi20/presentation/mehta.

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

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

14 Reudismam Rolim, Gustavo Soares, Rohit Gheyi, and Loris D'Antoni. Learning quick fixes from code repositories. *CoRR*, abs/1803.03806, 2018. URL: http://arxiv.org/abs/1803.03806, arXiv:1803.03806.

15 Ute 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.

16 Jörg H. Siekmann. Unification theory. *J. Symb. Comput.*, 7(3/4):207–274, 1989. doi:10.1016/S0747-7171(89)80012-4.

17 Erik Tidén and Stefan Arnborg. Unification problems with one-sided distributivity. *J. Symb. Comput.*, 3(1/2):183–202, 1987. URL: https://doi.org/10.1016/S0747-7171(87)80026-3, doi:10.1016/S0747-7171(87)80026-3.

A Proof of Theorem 13

Proof. We assume that t_1, t_2, and s are in U-normal form. We prove the theorem by induction on $dep(t_1) + dep(t_2)$ which we denote by n.

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

a) First, assume that $dep(s) = 1$. If $t_1 = t_2$, then $s = t_1 = t_2$ and s is computed by the derivation $\{x : t_1 \triangleq t_2\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow_{\mathsf{Dec}} \emptyset; \emptyset; \emptyset; \{x_{\text{root}} \mapsto t_1\}$. If $t_1 \neq t_2$, then s must be a variable, computed by the derivation $\{x : t_1 \triangleq t_2\}; \emptyset; \emptyset; \{x_{\text{root}} \mapsto x\} \Longrightarrow_{\mathsf{Sol}} \emptyset; \{x : t_1 \triangleq t_2\}; \emptyset; \{x_{\text{root}} \mapsto x\}$. Note that, in both cases the resulting tree grammars are trivial, both have a language of size 1. Thus, we will refer to the members of these languages directly rather than evoking the tree grammar itself.

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 $\mathsf{U} \in \mathsf{Ax}(f)$ 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 $\leq k + 1$. 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 in case 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 $s_1\sigma_2 = \epsilon_{f_{t_1}}$. 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 $t_1 = f(w_1, \ldots, w_m)$ 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 $\leq n-1$. Thus, by the induction hypothesis, for each generalization s' generalizing $X_i : w_i \triangleq r_i$ there exists a generalization $s_i^* \in \mathcal{L}(\mathcal{G}(B_i))$, where B_i is the final set of bindings computed using $\mathfrak{G}_{\mathsf{U\text{-}lin}}$, such that, $s' \preceq s_i^*$. Now let S_i^* be the set of all such generalizations computed using $\mathfrak{G}_{\mathsf{U\text{-}lin}}$. We may now define the set of generalizations S^* as $S^* = \{f(s_1^*, \ldots, s_m^*) \mid s_i^* \in S_i^* \text{ for all } 1 \leq i \leq m\}$. Note that each term in S^* is a generalization of $X : t_1 \triangleq t_2$ computed using $\mathfrak{G}_{\mathsf{U\text{-}lin}}$ in is contained in $\mathcal{L}(\mathcal{G}(B))$, where B is the final set of bindings computed using $\mathfrak{G}_{\mathsf{U\text{-}lin}}$. Thus, any generalization s' of $X : t_1 \triangleq t_2$ such that $\mathit{head}(s') = f$ is more general than some generalization of S^*. Thus we need only to consider generalization s' such that $\mathit{head}(s') \neq f$. This implies that $\mathsf{U} \in \mathsf{Ax}(\mathit{head}(s'))$.

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 $\mathit{head}(s_1') = f$. 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 $\mathit{head}(s') \neq f$ there exists $s'' \in S^*$ such that $s' \preceq s''$.

b) 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 $\mathit{head}(s') = f$ 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 $\mathsf{U} \in \mathsf{Ax}(\mathit{head}(s'))$ 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 2a).

c) Let us assume that $t_1 = f(w_1, \ldots, w_m)$ and $t_2 = g(r_1, \ldots, r_k)$, where either $\mathsf{U} \in \mathsf{Ax}(f)$ or $\mathsf{U} \in \mathsf{Ax}(g)$, or both. By an application of Exp-U-Both, Exp-U-L, or Exp-U-R this case can be reduced to two (possibly four) instances of case 2b).

◀

B Example used for the proof of nullarity

Below is the tree grammar computed from the final configuration of $\mathfrak{G}_{\mathsf{U}}$ applied to $\epsilon_g \triangleq \epsilon_f$. Computation of the final binding set required the application of 86 rules to the initial configuration.

$$\mathcal{G} = \left(\{\mathbf{x}\}, \left\{ \begin{array}{c} \mathbf{x}, \mathbf{x}_1, \\ \mathbf{x}_5, \mathbf{x}_{11} \\ \mathbf{x}_{18}, \mathbf{x}_{29} \end{array} \right\}, \left\{ \begin{array}{c} f, g, \\ \epsilon_f, \epsilon_g, \\ x_8, x_{36} \end{array} \right\}, \left\{ \begin{array}{ll} \mathbf{x} \mapsto g(\mathbf{x}, \mathbf{x}_5), & \mathbf{x} \mapsto g(\mathbf{x}_5, \mathbf{x}) \\ \mathbf{x} \mapsto \mathbf{x}_1, & \mathbf{x} \mapsto x_8 \\ \mathbf{x}_1 \mapsto f(\mathbf{x}, \mathbf{x}_{11}), & \mathbf{x}_1 \mapsto f(\mathbf{x}_{11}, \mathbf{x}) \\ \mathbf{x}_5 \mapsto f(\mathbf{x}, \mathbf{x}_{18}), & \mathbf{x}_5 \mapsto f(\mathbf{x}_{18}, \mathbf{x}) \\ \mathbf{x}_5 \mapsto \epsilon_g, & \mathbf{x}_{11} \mapsto g(\mathbf{x}_{18}, \mathbf{x}) \\ \mathbf{x}_{11} \mapsto g(\mathbf{x}, \mathbf{x}_{18}), & \mathbf{x}_{11} \mapsto \epsilon_f \\ \mathbf{x}_{18} \mapsto \mathbf{x}_{29}, & \mathbf{x}_{18} \mapsto x_{36} \\ \mathbf{x}_{18} \mapsto g(\mathbf{x}_5, \mathbf{x}_{18}), & \mathbf{x}_{18} \mapsto g(\mathbf{x}_{18}, \mathbf{x}_5) \\ \mathbf{x}_{29} \mapsto f(\mathbf{x}_{18}, \mathbf{x}_{11}), & \mathbf{x}_{29} \mapsto g(\mathbf{x}_{11}, \mathbf{x}_{18}) \end{array} \right\} \right).$$

If we clean the grammar by removing redundant bindings we get the tree grammar $\mathcal{G}'$:

$$\mathcal{G}' = \left(\{\mathbf{x}\}, \left\{ \begin{array}{c} \mathbf{x}, \\ \mathbf{y} \end{array} \right\}, \left\{ \begin{array}{c} f, g, \\ \epsilon_f, \epsilon_g, \\ y, z \end{array} \right\}, \left\{ \begin{array}{ll} \mathbf{x} \mapsto g(\mathbf{x}, f(\mathbf{x}, \mathbf{y})), & \mathbf{x} \mapsto f(\mathbf{x}, g(\mathbf{x}, \mathbf{y})) \\ \mathbf{x} \mapsto f(g(\mathbf{y}, \mathbf{x}), \mathbf{x}), & \mathbf{x} \mapsto x \\ \mathbf{x} \mapsto g(\mathbf{x}, f(\mathbf{y}, \mathbf{x})), & \mathbf{x} \mapsto f(\mathbf{x}, g(\mathbf{y}, \mathbf{x})) \\ \mathbf{x} \mapsto f(g(\mathbf{x}, \mathbf{y}), \mathbf{x}), & \mathbf{x} \mapsto g(f(\mathbf{y}, \mathbf{x}), \mathbf{x}) \\ \mathbf{x} \mapsto g(f(\mathbf{x}, \mathbf{y}), \mathbf{x}), & \mathbf{y} \mapsto f(g(\mathbf{y}, \mathbf{x}), \mathbf{y}) \\ \mathbf{y} \mapsto g(\mathbf{y}, f(\mathbf{y}, \mathbf{x})), & \mathbf{y} \mapsto f(\mathbf{y}, g(\mathbf{y}, \mathbf{x})) \\ \mathbf{y} \mapsto g(f(\mathbf{y}, \mathbf{x}), \mathbf{y}), & \mathbf{y} \mapsto y \\ \mathbf{y} \mapsto f(\mathbf{y}, g(\mathbf{x}, \mathbf{y})), & \mathbf{y} \mapsto g(\mathbf{y}, f(\mathbf{x}, \mathbf{y})) \\ \mathbf{y} \mapsto f(g(\mathbf{x}, \mathbf{y}), \mathbf{y}), & \mathbf{y} \mapsto g(f(\mathbf{x}, \mathbf{y}), \mathbf{y}) \end{array} \right\} \right).$$

Some of the generalizations contained in the language of this grammar are x, $f(x, g(x, y))$, $f(x, g(y, x))$, $f(g(y, x), x)$, $f(g(y, x), f(x, g(x, y)))$, $f(g(y, f(x, g(x, y))), f(x, g(x, y)))$, and $f(f(x, g(x, y)), g(f(x, g(x, y)), y))$. Observe that some of these generalizations are comparable and form a subsequence of an infinite chain of less generality.

C Grammar generated for Example 16

Below is the tree grammar computed from the final configuration of $\mathfrak{G}_{\mathsf{U(f)}}$ applied to $g(f(a, c), a) \triangleq g(c, b)$. Note that g is non-unital and no unit elements show up in the initial AUP. Computation of the final binding set required the application of 217 rules to the initial configuration. We only provide the cleaned version of the tree grammar. Note that the language of the resulting tree grammar is finite.

$$\mathcal{G} = \left(\{\mathbf{x}\}, \{\ \mathbf{x}\ \}, \left\{ \begin{array}{c} f, g, \epsilon_f, a, b, \\ c, y, z, y', z' \end{array} \right\}, B \right),$$

where B is the set

$$\left\{ \begin{array}{lll}
\mathbf{x} \mapsto g(f(f(y,z),y'),z') & \mathbf{x} \mapsto g(f(y,z),f(y',z')) & \mathbf{x} \mapsto g(f(f(z,y'),y),f(z,z')) \\
\mathbf{x} \mapsto g(f(f(z,y),y'),f(z,z')) & \mathbf{x} \mapsto g(f(y,y'),z') & \mathbf{x} \mapsto g(f(f(y,z),y'),f(z',z)) \\
\mathbf{x} \mapsto g(f(y,f(z,y')),z') & \mathbf{x} \mapsto g(f(z,f(y,y')),z') & \mathbf{x} \mapsto g(f(z,f(y',y)),z') \\
\mathbf{x} \mapsto g(f(f(z,y),y'),f(z',z)) & \mathbf{x} \mapsto g(f(f(z,y'),y),f(z',z)) & \mathbf{x} \mapsto f(y,z) \\
\boxed{\mathbf{x} \mapsto g(f(z,c),f(y,z))} & \mathbf{x} \mapsto g(f(y,y'),f(z',z)) & \mathbf{x} \mapsto g(f(z,f(y,y')),f(z,z')) \\
\mathbf{x} \mapsto g(f(y,f(z,y')),f(z,z')) & \mathbf{x} \mapsto g(f(z,f(y',y)),f(z,z')) & \mathbf{x} \mapsto g(f(z,c),z') \\
\mathbf{x} \mapsto g(f(f(z,y'),y),z') & \mathbf{x} \mapsto g(f(z,f(y,y')),f(z',z)) & \mathbf{x} \mapsto g(f(y,f(z,y')),f(z',z)) \\
\mathbf{x} \mapsto g(f(f(y,z),y'),f(z,z')) & \boxed{\mathbf{x} \mapsto g(f(z,c),f(z,y))} & \mathbf{x} \mapsto g(f(z,f(y',y)),f(z',z)) \\
\mathbf{x} \mapsto f(y,z) & \mathbf{x} \mapsto g(f(f(z,y),y'),z') &
\end{array} \right\}.$$

Observe that of the 26 terms contained in $\mathcal{L}(\mathcal{G})$, there are only two incomparable terms, $g(f(z, c), f(y, z))$ and $g(f(z, c), f(z, y))$.