Idempotent Anti-Unification

David Cerna and Temur Kutsia
RISC, Johannes Kepler University Linz, Austria

August 28, 2019

Abstract

In this paper we address two problems related to idempotent anti-unification. First, we show that there exists an anti-unification problem with a single idempotent symbol which has an infinite minimal complete set of generalizations. It means that anti-unification with a single idempotent symbol has infinitary or nullary generalization type, similar to anti-unification with two idempotent symbols, shown earlier by Loïc Pottier. Next, we develop an algorithm, which takes an arbitrary idempotent anti-unification problem and computes a representation of its solution set in the form of a regular tree grammar. The algorithm does not depend on the number of idempotent function symbols in the input terms. The language generated by the grammar is the minimal complete set of generalizations of the given anti-unification problem, which implies that idempotent anti-unification is infinitary.

1 Introduction

The equational theory of idempotence is defined by the axiom $f(x, x) \approx x$, stating that the function symbol f is idempotent. Many interesting algebraic structures have this property. It is an example of a regular collapse theory [41], 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). In fact, it is the simplest such theory. It makes idempotence interesting from the unification theory point of view. There are several works that studied unification modulo idempotence alone [25, 34, 40, 22, 19] or in combination with some other properties, see, e.g., [28, 40, 29, 2, 38, 37, 4]. They show that in the unification hierarchy [40, 6], idempotent unification is finitary (i.e., the minimal complete set of unifiers always exists and is finite). It remains finitary if idempotence is combined with commutativity, but becomes of type zero in combination with associativity. Type zero means that for some unification problems the minimal complete set of unifiers does not exist: minimality conflicts with completeness. However, if we put all three properties together, associativity, commutativity, and idempotence, then it is again finitary.

Anti-unification with idempotent functions has been studied with far less intensity. The work of [33] shows that in a theory with two idempotent symbols, there exists an anti-unification problem with infinitely many incomparable generalizations. Anti-unification in description logic $\mathcal{EL}$ [24] uses a combination of idempotence with associativity and commutativity, plus some other special properties.

In general, unification and anti-unification types do not correlate. For instance, associative unification is infinitary (minimal complete set of unifiers always exists and for some problems it is infinite) [32, 39], but associative anti-unification is finitary [1]. Nominal unification is unitary [42], but nominal anti-unification (with unrestricted number of atoms, as in nominal unification) is of type zero [7, 10]. There are similarities as well: both syntactic unification [36] and anti-unification [31, 35] are unitary. The same is true for higher-order patterns [30, 9, 11]. For theories with commutative symbols, both unification and anti-unification are finitary [1].

In this context, it is interesting to see how idempotent anti-unification behaves. Pottier's result implies that anti-unification with two idempotent symbols is at least infinitary, i.e., it is either

infinitary or of type zero. But which one is actually the case? Moreover, what happens if we allow any number of idempotent symbols?

These were the questions that motivated our work, and we showed that, in fact, two idempotent symbols are not necessary to have a problem with infinitely many least general generalizations: one suffices. Moreover, we showed that idempotent anti-unification is infinitary, not of type zero: the minimal complete set of generalizations always exists. This is true no matter how many idempotent symbols exist within the theory. It gives yet another dichotomy between unification and anti-unification: this time, finitary vs infinitary. Moreover, we have constructed an algorithm, which computes a finite representation of such sets in the form of regular tree grammar, for any number of idempotent symbols in the input.

The results of this paper contribute to the understanding of anti-unification for one of the most basic, yet nontrivial equational theories. In general, equational anti-unification problem is studied less intensively than its unification counterpart. We review some of the work related to this subject below in a separate section. Besides the theoretical importance, our results have a practical merit as well. Most importantly, showing that idempotent anti-unification is (at least) infinitary for one idempotent function symbol is encouraging from the perspective of the development of a combination method for anti-unification algorithms (in the style of such results for unification in the union of signature-disjoint theories [5]). If the theory with one idempotent symbol had the finitary generalization type, it would mean that finitary anti-unification algorithms cannot be combined, since from [33] it is known that anti-unification with two idempotent symbols is infinitary or of type zero. Also, the idea of representing generalization sets by regular tree grammars (which resembles to the idea of E-generalizations using grammars from [14]), on which we elaborate in this paper, might be worth to extend to other theories as well, aiming at representing preferably minimal, complete sets of generalizations compactly and developing terminating algorithms to compute them.

From the applicability perspective, idempotent generalization, in a restricted form or in combination with other equational properties, can be potentially useful for proof transformation methods such as schematic cut elimination [27], cut introduction [21, 20] and induction theorem proving based on tree grammar construction and minimization [17].

The paper is organized as follows: In Section 2 we introduce the main notions and establish the terminology. In Section 3, we give an example of an anti-unification problem with one idempotent symbol, which has an infinite complete set of generalizations. We prove that this set is also minimal, which implies that such a theory is either infinitary or of type zero. In Section 4 an algorithm for anti-unification with an arbitrary number of idempotent function symbols is described. The algorithm accepts two terms and computes a regular tree grammar, for which we prove that it generates a complete set of idempotent generalizations of the input terms. In Section 5 we prove that the algorithm is also minimal, i.e., the computed grammar generates a minimal complete set of generalizations, which implies that anti-unification with arbitrary number of idempotent function symbols is infinitary. Section 6 concludes.

1.1 Related Work

As we have already mentioned, equational anti-unification has not been studied as intensively as equational unification. [33] gave not only an example of idempotent anti-unification with an infinite set of solutions, but also an algorithm for anti-unification with one associative-commutative symbol. [3] studied how different instantiation preorders affect E-generalization problems, and addressed generalization in the class, called commutative equational theories. Examples of such theories are commutative monoids, commutative idempotent monoids, and abelian groups. Generalization in commutative theories is unitary.

In [13], the author studied word anti-unification (anti-unification in free monoids) together with its special case, so called ϵ-free generalization, and showed that both are finitary. Generalization for sequences of unranked terms has been investigated in [43] for a restricted case and in [26, 8] for the general first- and second-order cases. Unranked term-graph anti-unification has been studied in [12].

[1] described a modular rule-based anti-unification algorithm for order-sorted theories with associative, commutative, and unit function symbols and for their combinations, and showed that the generalization problem is finitary for any considered axiom combination. Associative, commutative, and associative-commutative anti-unification for higher-order patterns has been discussed and proved finitary in [16].

The problem of generalization for clauses studied in [31] can also be seen as a special equational anti-unification problem. Considering the disjunction as an associative, commutative, idempotent (ACI) symbol, taking the empty clause as the unit element and the subclause relation for comparison, one can say that the anti-unification problem for clauses is an ACIU (ACI with a unit element) anti-unification problem for terms where the ACI symbol appears only in the topmost position.

[14] and [18] (see also [15]) proposed an interesting idea to use regular tree grammars to compute a finite representation of a complete set of generalizations, provided that the equational theory leads to regular congruence classes. They first construct grammars for congruence classes of the input terms, and then compute a grammar representation of the solution set. We could have applied this technique to idempotent anti-unification considered in this paper, but we were interested in obtaining a description of a *minimal* complete set of generalizations. Computing first a grammar for a complete set of generalizations, and then trying to obtain from there a description of a minimal complete set seems to be a non-trivial task. Therefore, we developed an algorithm that directly computes the desired representation for the idempotent equational theory.

2 Preliminaries

We assume familiarity with the basic notions of unification theory, see, e.g., [6].

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 $t ::= x \mid f(t_1, \ldots, t_n)$, 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 $dep(x) = 1$ for variables and $dep(f(t_1, \cdots, t_n)) = \max\{dep(t_1), \cdots, dep(t_1)))\} + 1$ otherwise.

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 $pos(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i.p \mid p \in pos(t_i)\}$, 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 $head(f(t_1, \ldots, t_n)) = f$.

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 $B\sigma = \{x \mapsto t\sigma \mid x \mapsto t \in B\}$.

An *idempotent equational theory* is generated (in the usual way) from the axiom of idempotence for some function symbol, i.e., from an equality of the form $f(x, x) \approx x$, which states that f is idempotent. We denote this equational theory by $\approx_I$, or by $\approx_{I(f,g,\ldots)}$, if we want to make explicit the idempotent symbols.

In equational (anti-)unification, function symbols that do not occur in the equational axioms of the theory are called *free function symbols* (with respect to the theory). Since we consider only idempotent theories, in our case every non-idempotent function symbol is free.

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

A term r is *more general* than s modulo I (r is an *I-generalization* of s) if there exists a substitution σ such that $r\sigma \approx_I s$. It is written as $r \preceq_I s$. The relation $\preceq_I$ is a quasi-ordering. Its strict part is denoted by $\prec_I$, and the equivalence relation it induces by $\simeq_I$.

Given two terms t and s, we say that a term r is their *least general generalization* modulo idempotence (I-lgg or just lgg in short), if $r \preceq_I s$ and $r \preceq_I t$ hold and for all r' with the property $r' \preceq_I t$, $r' \preceq_I s$, and $r \preceq_I r'$ we have $r \simeq_I r'$.

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

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

We write $mcsg_I(t,s)$ for the minimal complete set of I-generalizations of t and s. An *anti-unification problem* is a pair of terms for which an *mcsg* is to be computed.

Often we just say generalization, lgg, etc. instead of I-generalization, I-lgg and so on.

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:

- Unary 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 [31, 35] is unitary; commutative anti-unification [1] is finitary; nominal anti-unification with infinitely many atoms is nullary [7, 10]. In this paper we illustrate that idempotent anti-unification is infinitary. This holds for an arbitrary number of idempotent function symbols involved in anti-unification problems.

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

Sometimes, when we want to anti-unify s and t, we simply say that we have an *anti-unification problem* (AUP) $s \triangleq_I t$.

In all the notations, we omit I when it is clear from the context.

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 $G = \langle \alpha, N, T, R\rangle$, the *derivation relation* $\to_G$ is a relation on pairs of terms of $\mathcal{T}(T \cup N)$ such that $s \to_G t$ if and only if there exists a position p in s and a rule $\nu \to r \in R$ such that $s|_p = \nu$ and $t = s[r]_p$. The *language generated by* G is the set of terms $L(G) := \{t \mid t \in \mathcal{T}(T) \text{ and } \alpha \to_G^+ t\}$, where $\to_G^+$ is the transitive closure of the relation $\to_G$. Given a grammar G, the set of nonterminals of G that appear in a syntactic object (term, rule, AUT, etc.) O is denoted by $nter(G, O)$. For a grammar G, the set of nonterminals that *can be reached* from a nonterminal ν, denoted by $reach(G, \nu)$, is defined as $reach(G, \nu) := \{\mu \mid \nu \to_G^* t \text{ and } \mu \in nter(G, t)\}$, where $\to_G^*$ is reflexive and transitive closure of $\to_G$. When the grammar is clear from the context, we write $\to$ instead of $\to_G$.

Given a set U and an equivalence relation $\equiv$ on the elements of U, we denote an $\equiv$-equivalence class of $u \in U$ by $[u]$. We assume to have a function $rep_\equiv$ which for each $\equiv$-equivalence class gives its representative. For a set $M \subseteq U$, we denote by $Rep_\equiv(M)$ the set of representatives of $\equiv$-equivalence classes of elements of M: $Rep_\equiv(M) := \{rep_\equiv([m]) \mid m \in M\}$.

We extend the relation $\preceq_I$ and $\approx_I$ to bindings: $x \mapsto t \preceq_I y \mapsto s$ iff $x = y$ and $t \preceq_I s$. Then we have $x \mapsto t \approx_I y \mapsto s$ iff $x = y$ and $t \approx_I s$. Obviously, it is an equivalence relation. The strict relation $\prec_I$ is extended to bindings straightforwardly.

Given a set M of terms (resp. bindings) and the corresponding equivalence relation $\approx_I$, the function *Minimize* keeps only those representatives of $\approx_I$-equivalence classes of elements of M, which are not more general than the other representatives:

$$\begin{aligned} &Minimize(M) := \\ &\quad Rep_{\approx_I}(M) \setminus \{m \in Rep_{\approx_I}(M) \mid \text{there exists } m' \in Rep_{\approx_I}(M) \text{ such that } m \prec_I m'\}. \end{aligned}$$

Example 1 *We illustrate how Minimize(M) works using the following set of terms:*

$$M = \{a, f(a,a), f(b,x), f(f(x,b),y), f(f(a,x), f(x,a))\}$$

over the equational theory $\approx_{I(f)}$ where x, y are variables. Let $rep_{\approx_{I(f)}}$ select the canonical representative from each $\approx_I$-equivalence class, i.e., the term in I-normal form. Then $Rep_{\approx_I}(M)$ is the following set of terms:

$$\begin{aligned} &Rep_{\approx_I}(M) = \\ &\quad \{rep_{\approx_{I(f)}}([a]), rep_{\approx_{I(f)}}([f(a,a)]), rep_{\approx_{I(f)}}([f(b,x)]), rep_{\approx_{I(f)}}([f(f(x,b),y)]), \\ &\quad\ rep_{\approx_{I(f)}}([f(f(a,x), f(x,a))])\} = \{a, f(b,x), f(f(x,b),y), f(f(a,x), f(x,a))\}. \end{aligned}$$

Notice that $f(f(x,b),y) \prec_{I(f)} f(b,x)$ and $f(f(a,x), f(x,a)) \prec_{I(f)} a$. Therefore,

$$Minimize(M) = Rep_{\approx_I}(M) \setminus \{f(f(x,b),y), f(f(a,x), f(x,a))\} = \{a, f(b,x)\}.$$

Example 2 *Here we illustrate Minimize(M) for a set of bindings:*

$$M = \{x \mapsto a,\ y \mapsto f(a,a),\ z \mapsto f(b,x),\ z \mapsto f((x,b),y),\ y \mapsto f(f(a,x), f(x,a))\},$$

over the equational theory $\approx_{I(f)}$. Let $rep_{\approx_{I(f)}}$ select representatives in I-normal form. Then $Rep_{\approx_I}(M)$ is the following set of bindings:

$$Rep_{\approx_I}(M) = \{x \mapsto a,\ y \mapsto a,\ z \mapsto f(b,x),\ z \mapsto f(f(x,b),y),\ y \mapsto f(f(a,x), f(x,a))\}.$$

Since $z \mapsto f(f(x,b),y) \prec_{I(f)} z \mapsto f(b,x)$ and $y \mapsto f(f(a,x), f(x,a)) \prec_{I(f)} y \mapsto a$, we get

$$\begin{aligned} Minimize(M) &= Rep_{\approx_I}(M) \setminus \{z \mapsto f(f(x,b),y),\ y \mapsto f(f(a,x), f(x,a))\} \\ &= \{x \mapsto a,\ y \mapsto a,\ z \mapsto f(b,x)\}. \end{aligned}$$

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 minimal complete sets of idempotent generalizations by finite means with the help of regular tree grammars. Hence, we want to develop an I-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 will be developed in Section 4. It computes a set of bindings, from which one can directly construct the desired grammar, based on the correspondence we define below in Definition 2.

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 in the left hand side of the root binding only.

Definition 1 (Base Regular Tree Grammar Corresponding to a Set of Bindings) *Given (nonempty) set of bindings B, the* corresponding base regular tree grammar $G_b(B) = \langle \alpha, N, T, R_b \rangle$ *is defined by the following construction:*

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

$$R_b := \{x \to r \mid x \mapsto r \in Minimize(B)\}.$$

  In matching needed to decide $\prec_I$ for minimization, only variables from T can match. Variables from N are treated as constants.

Note that the language generated by $G_b(B)$ is finite.

Definition 2 (Regular Tree Grammar Corresponding to a Set of Bindings) *Let B be a (nonempty) set of bindings, and $G_b(B) = \langle \alpha, N, T, R_b \rangle$ be the corresponding base regular tree grammar. The* regular tree grammar corresponding to B *is the grammar $G_b = \langle \alpha, N, T, R \rangle$, where α, N, T are as in $G_b(B)$, and the set of rules R is defined by adding to R_b some recursive duplication rules with idempotent symbols:*

$$R = R_b \cup \bigcup_{x, \exists r. x \to r \in R_b} Duplicate(x, G_b(B)),$$

where $Duplicate(x, G_b(B))$ is defined as

$$\begin{aligned} &Duplicate(x, G_b(B)) := \\ &\quad \{x \to f(x, x) \mid f \text{ is an idempotent symbol and } R_b \text{ contains} \\ &\quad\ \text{two different rules } y \to r_1 \text{ and } y \to r_2 \text{ for } y \in reach(G_b(B), x)\}. \end{aligned}$$

Note that Duplicate uses all idempotent function symbols. Usually we assume that they are those that appear in B.

Before illustrating the introduced notion with an example, we briefly explain it. The motivation of this definition, as we have already mentioned, comes from the desire of designing a relatively simple, terminating minimal and complete I-generalization algorithm, which would return a set of bindings instead of directly constructing a regular tree grammar. However, simplicity has its price and, as we will see in Section 4, the algorithm designed there computes a set of bindings B which is not necessarily minimal. This is why we make sure to minimize the set of rules obtained from such a B. Moreover, B only represents the syntactically derivable generalizations. As Pottier

pointed out in [33], there are many generalizations that are incomparable to the syntactically derivable ones. These generalizations can only be constructed by computing the transitive closure of the axiom of idempotence within the tree grammar. This is precisely the reason why we add the duplication rules.

Example 3 *Let the set of bindings B, where f is idempotent, be defined as follows:*

$$B = \{x_1 \mapsto y_1,\ y_1 \mapsto f(x,y),\ y_1 \mapsto f(f(x,b),f(a,y)),\ y_1 \mapsto f(f(x,a),f(b,y))\},$$

where x_1 is the root. Minimization of B and the construction of R_b gives

$$R_b = \{x_1 \to y_1,\ y_1 \to f(f(x,b),f(a,y)),\ y_1 \to f(f(x,a),f(b,y))\}.$$

The base grammar $G_b(B)$ has the form

$$G_b(B) = \langle x_1, \{x_1, y_1\}, \{f, a, b, x, y\}, R_b\rangle.$$

Duplication gives

$$\begin{aligned} &\mathit{Duplicate}(x_1, G_b(B)) \cup \mathit{Duplicate}(y_1, G_b(B)) = \{x_1 \to f(x_1,x_1)\} \cup \{y_1 \to f(y_1,y_1)\} = \\ &\quad \{x_1 \to f(x_1,x_1),\ y_1 \to f(y_1,y_1)\}. \end{aligned}$$

Hence, we get

$$\begin{aligned} R = \{\ &x_1 \to y_1,\ x_1 \to f(x_1,x_1),\ y_1 \to f(f(x,b),f(a,y)),\ y_1 \to f(f(x,a),f(b,y)),\\ &y_1 \to f(y_1,y_1)\} \end{aligned}$$

and the final grammar is

$$G(B) = \langle x_1, \{x_1, y_1\}, \{f, a, b, x, y\}, R\rangle.$$

3 Infinitely Many Generalizations with One Idempotent Symbol

In this section we give an example that shows that idempotent anti-unification problems might have infinitely many anti-unifiers even if the theory contains only one idempotent symbol. The investigations by [33] provided an example of an idempotent anti-unification problem using two idempotent function symbols which has infinitely many incomparable generalizations.

His example is as follows: consider an alphabet $\Sigma = \{f, g, a, b\}$ where both f and g are binary idempotent function symbols and a and b are constants. Now we consider the following generalization problem $f(a,b) \triangleq g(a,b)$. Notice that $g(f(a,x),f(y,b))$ and $f(g(a,x),g(y,b))$ are lggs of the given problem (we will refer to them as G_1 and G_2, respectively):

$$\begin{aligned} G_1\{x \mapsto b, y \mapsto a\} &= g(f(a,x),f(y,b))\{x \mapsto b, y \mapsto a\} = g(f(a,b),f(a,b)) \approx_I f(a,b).\\ G_1\{x \mapsto a, y \mapsto b\} &= g(f(a,x),f(y,b))\{x \mapsto a, y \mapsto b\} = g(f(a,a),f(b,b)) \approx_I g(a,b).\\ G_2\{x \mapsto a, y \mapsto b\} &= f(g(a,x),g(y,b))\{x \mapsto a, y \mapsto b\} = f(g(a,a),g(b,b)) \approx_I f(a,b).\\ G_2\{x \mapsto b, y \mapsto a\} &= f(g(a,x),g(y,b))\{x \mapsto b, y \mapsto a\} = f(g(a,b),g(a,b)) \approx_I g(a,b). \end{aligned}$$

Using them we can develop the following recursive construction:

$$\begin{aligned} S_0 &= G_1,\\ S_{n+1} &= f(G_1, g(S_n, G_2)). \end{aligned}$$

Notice that S_{n+1} is always incomparable to S_n. However, this is not the minimal complete set, because the construction is limited to repeated use of $\{G_1, G_2\}$. Later in this section we provide

a minimal complete construction, but first we address the issue of infinitely many generalizations with one idempotent symbol by encoding Pottier's example in a simpler language.

Consider the alphabet $\Sigma = \{h, a, b\}$ where h is an idempotent function symbol and a and b are constants. Note that we can encode Pottier's example, using this signature, by encoding $f(\cdot,\cdot)$ as $h(a, h(\cdot,\cdot))$ and $g(\cdot,\cdot)$ as $h(b, h(\cdot,\cdot))$. One can think of this as an encoding of two binary functions using a ternary function and two constants, though the precise motivation does not matter for the results which follow. Thus, the generalization problem is now $h(a, h(a, b)) \triangleq h(b, h(a, b))$. Also, like in the previous case, there are at least two solutions to this problem which, as we will see below, are lggs:

$$G'_1 = h(h(x, h(x, b)), h(a, h(x, b))),$$
$$G'_2 = h(h(x, h(a, x)), h(h(x, b), h(a, b))).$$

The fact that G'_1 and G'_2 are solutions of $h(a, h(a, b)) \triangleq h(b, h(a, b))$ can be illustrated by the following instantiations:

$$G'_1\{x \mapsto a\} = h(h(a, h(a, b)), h(a, h(a, b))) \approx_I h(a, h(a, b)).$$
$$G'_1\{x \mapsto b\} = h(h(b, h(b, b)), h(a, h(b, b))) \approx_I h(b, h(a, b)).$$
$$G'_2\{x \mapsto a\} = h(h(a, h(a, a)), h(h(a, b), h(a, b))) \approx_I h(a, h(a, b)).$$
$$G'_2\{x \mapsto b\} = h(h(b, h(a, b)), h(h(b, b), h(a, b))) \approx_I h(b, h(a, b)).$$

Lemma 1 *The terms G'_1 and G'_2 are lggs of $h(a, h(a, b)) \triangleq h(b, h(a, b))$.*

Proof.

We prove the lemma for G'_1. For G'_2 it is similar. Let us denote $s_1 = h(a, h(a, b))$ and $s_2 = h(b, h(a, b))$.

We need to show that for any substitution σ, if $G'_1 \prec_I G'_1\sigma$, then $G'_1\sigma$ is not a generalization of both s_1 and s_2. We can assume without loss of generality that the domain of σ is $\{x\}$, i.e., it has the form $\{x \mapsto t\}$, and prove the statement by induction on the depth of t. The I-normal form of t is assumed.

The assumption $G'_1 \prec_I G'_1\sigma$ suggests that we should consider only t which is not $\simeq_I$-equivalent to a variable. Otherwise we will have $G'_1 \simeq_I G'_1\sigma$, not $G'_1 \prec_I G'_1\sigma$.

Note that our alphabet contains only h, a, and b. Any term built over these symbols and variables is $\simeq_I$-equivalent to a variable iff the term is constructed only by the idempotent symbol h and variables. For instance, $h(x_1, h(x_2, x_3)) \simeq_I x_4$, because $h(x_1, h(x_2, x_3))\{x_1 \mapsto x_4, x_2 \mapsto x_4, x_3 \mapsto x_4\} \approx_I x_4$ and $x_4\{x_4 \mapsto h(x_1, h(x_2, x_3))\} \approx_I h(x_1, h(x_2, x_3))$.

Now we proceed by induction on the depth of t. The base case is when the depth is 1. This can happen only if t is a, t is b, or t is a variable. In the first two cases $G'_1\sigma$ is not a generalization of both $h(a, h(a, b))$ and $h(b, h(a, b))$. The third case is excluded because t is $\simeq_I$-equivalent to a variable. Hence, the base case is proved.

As the induction hypothesis, assume that if $G'_1 \prec_I G'_1\{x \mapsto t\}$, then $G'_1\{x \mapsto t\}$ is not a generalization of both s_1 and s_2 for any t of depth n, and prove the same statement for any t' of depth $n + 1$. For this, we take t' of depth $n + 1$ and assume $G'_1 \prec_I G'_1\{x \mapsto t'\}$. The depth $n + 1$ suggests that t' should have a form $h(t_1, t_2)$, where the depths of t_1, t_2 are at most n, and at least one of them, say t_1, has the depth necessarily n.

If t' is ground, then we should have $t' \approx_I a$ in order $G'_1\{x \mapsto t'\}$ to be a generalization of $h(a, h(a, b))$, and $t' \approx_I b$ in order $G'_1\{x \mapsto t'\}$ to be a generalization of $h(b, h(a, b))$. But $t' \approx_I a$ and $t' \approx_I b$ at the same time is not possible. Hence, $G'_1\{x \mapsto t'\}$ cannot be a generalization of both s_1 and s_2 for a ground t'.

For a non-ground t', it should contain at least one a or b. Otherwise it would be $\simeq_I$-equivalent to a variable, which is forbidden by the assumption. Now, assume that there exist substitutions σ'_1

and σ'_2 such that $G'_1\{x \mapsto h(t_1,t_2)\}\sigma'_1 \approx_I h(a,h(a,b))$ and $G'_1\{x \mapsto h(t_1,t_2)\}\sigma'_2 \approx_I h(b,h(a,b))$. Then it must be the case that $h(t_1,t_2)\sigma'_1 \approx_I a$ and $h(t_1,t_2)\sigma'_2 \approx_I b$. But this would imply that $t_1\sigma'_1 \approx_I a$, $t_2\sigma'_1 \approx_I a$, $t_1\sigma'_2 \approx_I b$, $t_2\sigma'_2 \approx_I b$ and thus $G'_1\{x \mapsto t_1\}$ and $G'_1\{x \mapsto t_2\}$ would be also generalizations. However, this contradicts the induction hypothesis. Hence, we get that $G'_1\{x \mapsto t'\}$ is not a generalization of s_1 and s_2 for a non-ground t' either.

□

We can use the same recursive construction as Pottier's to produce an infinite set of incomparable generalizations. Though, once again, this construction does not result into a minimal complete set of generalizations. However, it turns out that we can come up even with a simpler example, which has infinitely many incomparable generalizations and we could even construct an $mcsg_I$. Such a generalization problem is $h(a,b) \triangleq h(b,a)$, which has at least the following two lggs, which we denote by H_1 and H_2:

$$H_1 = h(h(x,b),h(a,y)),$$
$$H_2 = h(h(x,a),h(b,y)).$$

Notice that $H_1\{x \mapsto a, y \mapsto b\} = h(a,b)$ and $H_1\{x \mapsto b, y \mapsto a\} = h(b,a)$. Similar to the example handled by Lemma 1, to prove that H_1 is an lgg we need to show that a substitution σ with domain $\{x,y\}$, when applied to H_1, does not result in a generalization of $h(a,b) \triangleq h(b,a)$. This requires a slightly more complex induction but essentially the same proof of the base and step case would be used.

Now we modify Pottier's recursive construction to obtain infinite set of generalizations for $h(a,b) \triangleq h(b,a)$:

$$S_0 = \{H_1,\ H_2\}.$$
$$S_k = \{h(s_1,s_2) \mid s_1,s_2 \in S_{k-1}, s_1 \neq s_2\} \cup S_{k-1},\ k > 0.$$

For example, we have

$$\begin{aligned}
S_1 = {} & \{h(H_1,H_2),\ h(H_2,H_1)\} \cup S_0.\\
S_2 = {} & \{h(H_1,h(H_1,H_2)),\ h(H_1,h(H_2,H_1)),\ h(h(H_1,H_2),H_1),\ h(h(H_2,H_1),H_1)),\\
& h(H_2,h(H_1,H_2)),\ h(H_2,h(H_2,H_1)),\ h(h(H_1,H_2),H_2),\ h(h(H_2,H_1),H_2),\\
& h(h(H_1,H_2),h(H_2,H_1)),\ h(h(H_2,H_1),h(H_1,H_2))\} \cup S_1.\\
& \ldots
\end{aligned}$$

By S_∞ we denote the limit of this construction. Its elements are generalizations of $h(a,b)$ and $h(b,a)$, because for any $g \in S_\infty$ we have $g\{x \mapsto a, y \mapsto b\} \simeq_I h(a,b)$ and $g\{x \mapsto b, y \mapsto a\} \simeq_I h(b,a)$.

The theorem below shows that S_∞, in fact, is a minimal complete set of generalizations for $h(a,b) \triangleq h(b,a)$. However, enumerating the minimal complete set of generalizations for arbitrary AUTs is much more involved, as we will see in the next sections.

Theorem 1 $S_\infty = mcsg_I(h(a,b),h(b,a))$.

Proof.

We have already shown soundness above, when we proved that S_∞ is a set of generalizations of $h(a,b)$ and $h(b,a)$.

Minimality follows from the fact that the definition of S_∞ guarantees that any two terms in the constructed set are incomparable with respect to $\preceq_I$. It is not difficult to see: First, note that $H_1 = h(h(x,b),h(a,y))$ and $H_2 = h(h(x,a),h(b,y))$ are $\preceq_I$-incomparable. Moreover, H_1 and H_2 are $\preceq_I$-incomparable with both $h(H_1,H_2)$ and $h(H_2,H_1)$, and also $h(H_1,H_2)$ and $h(H_2,H_1)$ are $\preceq_I$-incomparable (i.e., S_1 does not contain $\preceq_I$-comparable elements). As the construction

of the S's shows, this observation extends to arbitrary S_k: if s_1 and s_2 are $\preceq_I$-incomparable in S_{k-1}, then so are (pairwise) s_1, s_2, $h(s_1, s_2)$, and $h(s_2, s_1)$ in S_k. Iterating this argument over the construction of the S's, we get that S_∞ does not contain $\preceq_I$-comparable elements.

Now we show completeness, taking an arbitrary generalization G of $h(a, b)$ and $h(b, a)$ and showing that S_∞ contains G' such that $G \preceq_I G'$. We prove the statement by case distinction on G. When it is a variable, the statement is trivial. Assume that it is not a variable. Then G must be of the form $h(g', g'')$, since our term signature is $\{h(\cdot, \cdot), a, b\}$. Let σ_1 and σ_2 be substitutions with domain $\{x_1, \ldots, x_m\}$ such that $h(g', g'')\sigma_1 \approx_I h(a, b)$ and $h(g', g'')\sigma_2 \approx_I h(b, a)$. Then one of the following cases must hold:

(a) $g'\sigma_1 \approx_I a$ and $g''\sigma_1 \approx_I b$ and $g'\sigma_2 \approx_I b$ and $g''\sigma_2 \approx_I a$.

(b) $g'\sigma_1 \approx_I a$ and $g''\sigma_1 \approx_I b$ and $g'\sigma_2 \approx_I g''\sigma_2 \approx_I h(b, a)$.

(c) $g'\sigma_1 \approx_I g''\sigma_1 \approx_I h(a, b)$ and $g'\sigma_2 \approx_I b$ and $g''\sigma_2 \approx_I a$.

(d) $g'\sigma_1 \approx_I g''\sigma_1 \approx_I h(a, b)$ and $g'\sigma_2 \approx_I g''\sigma_2 \approx_I h(b, a)$.

Case (a) If we think back to Lemma 1, there are restrictions on the construction of g' and g''. Note that if g' (without loss of generality) contains an occurrence of a, then $g'\sigma_2 \not\approx_I b$ because the occurrence will remain, even after idempotent normalization. The same goes for occurrences of b. Thus, the only non-variable symbol occurring in g' and g'' would be $h(\cdot, \cdot)$.

We prove this case by induction on the depth $n \geq 2$ of G. When $n = 2$, then both subterms of G, i.e., g' and g'' have term depth 1. Therefore none of them contains an instance of $h(\cdot, \cdot)$ and thus both g' and g'' must be variables. Furthermore, they must be different variables implying that $G \simeq_I h(x_1, x_2)$. However, it is obvious that $h(x_1, x_2)$ is more general than H_1 and H_2 from S_0. Hence, we have elements in S_∞ which are less general than $h(g', g'')$.

Assume now that when the depth of G is at most n, then there exists $G' \in S_\infty$ such that $G \preceq_I G'$, and prove the statement for the case when the depth of G is $n + 1$.

From the induction hypothesis it follows that if g' and g'' have their depths $< n$, then there exists $G' \in S_\infty$ such that $h(g', g'') \preceq_I G'$. Now, proving the induction step means that if at least one of them, say g', has depth n, then we can find $G' \in S_\infty$ such that $h(g', g'') \preceq_I G'$.

Let us consider a subterm of g' which has term depth 2 and thus is of the form $h(x_1, x_2)$. We can transform $h(g', g'')$ to some $h(g^*, g'')$ by applying the substitution $\{x_1 \mapsto y, x_2 \mapsto y\}$ to $h(g', g'')$ and I-normalize it. This decreases the term depth at least by 1 and by the induction hypothesis there exists a $G' \in S_\infty$ such that $h(g', g'') \preceq_I h(g^*, g'') \preceq_I G'$. If $h(g^*, g'')$ is a generalization of $h(a, b)$ and $h(b, a)$, we are done. Now assume that it is not the case. But then σ_1 and σ_2 would have had to map x_1 and x_2 to different values. However, this would imply that both a and b would show up in $g'\sigma_1$, contradicting our assumption for case (a).

Case (b) Note that case (b) with minor changes proves case (c) as well. In this case g' can contain instances of a and g'' of b. If they do not contain instances of a or b then G is trivially more general than generalizations in S_0. Also, the variables of g' are distinct from the variables of g'' by the assumptions of case (b). If g' and g'' shared variables then it would not be possible for $g'\sigma_1 \approx_I a$ and $g''\sigma_1 \approx_I b$ because either $g'\sigma_1 \approx_I a$ or $g''\sigma_1 \approx_I b$ would necessarily need to contain both an instance of a and b. This can be proven by a simple argument: assume that g' and g'' share a variable y, and σ_1 maps y to a term containing a then $g''\sigma_1$ will necessarily contain a contradicting our construction. The same argument holds for a term containing b.

By the assumptions of case (b) $g'\sigma_2 \approx_I h(b, a)$. Let the notation $s[t \leftarrow z]$ stand for the term obtained from s by replacing all occurrences of t in s by the variable z, i.e., abstracting the occurrences of the subterm t in s by z. Let us instead of considering σ_2 consider a substitution θ' defined as follows:

$$x\theta' = (x\sigma_2)[b \leftarrow z'] \text{ for all } x,$$

where z' is a fresh variable. Notice that $g'\theta' \approx_I h(z', a)$ holds, which does not violate the assumptions of case (b). The same construction can be carried out for g'' with respect to a and z'', resulting in $g''\theta'' \approx_I h(b, z'')$. Thus, $G \approx_I h(g'\theta', g''\theta'') \approx_I h(h(z', a), h(b, z'')) \in S_0$.

Case (d) We now consider the case when g' and g'' are both generalizations of $h(a, b) \triangleq h(b, a)$. We prove by induction on the depth $d \geq 4$ of G that there exists $G' \in S_\infty$ such that $G \preceq_I G'$.

For the base case, let $d = 4$. This implies that g' and g'' are terms of at most depth 3. The only terms of depth at most 3, which generalize $h(a, b) \triangleq h(b, a)$, are those from S_0. By construction of S_∞, we know $S_0 \subset S_\infty$. If $g' = g''$, then $G \approx_I g' \in S_\infty$ and we take $G' = g'$. If $g' \neq g''$, then $G \in S_\infty$ by construction and we take $G' = G$. This concludes the base case.

Now assume for the induction hypothesis that the statement is true for the depth $\leq d$ of G and show it for $d + 1$. Recall that $G = h(g', g'')$. Hence, the depths of g' and g'' are at most d. By the assumption of Case (d), g' and g'' are both generalizations of $h(a, b) \triangleq h(b, a)$. By the induction hypothesis, there exist $g_1 \in S_\infty$ and $g_2 \in S_\infty$ such that $g' \preceq_I g_1$ and $g' \preceq_I g_2$. If $g_1 \neq g_2$, then, by construction, there exists $h(g_1, g_2) \in S_\infty$ and we can take $G' = h(g_1, g_2)$, since $G = h(g', g'') \preceq_I h(g_1, g_2)$. If $g_1 = g_2$, then we have $G = h(g', g'') \preceq_I h(g_1, g_2) \approx_I g_1$ and we can take $G' = g_1 \in S_\infty$.

□

Hence, the S_n construction provides an enumeration of $mcsg_I(h(a, b), h(b, a))$ and can be seen as a finite algorithmic description of this set. We can describe its growth rate as a recurrence for a generalization of the S_n construction where S_0 can have an arbitrary size.

Theorem 2 *Let A be a finite set, $P(S, S') = \{(a, b) \mid a \in S,\ b \in S',\ a \neq b\}$, and F_n be the following recursive set construction:*

$$\begin{aligned} F_0 &= \{\emptyset\}, \\ F_1 &= A, \\ F_{n+1} &= F_n \cup P(F_n, F_n). \end{aligned}$$

Then for $n \geq 1$, $|F_{n+1}| = |F_n^2| - |F_{n-1}^2| + |F_{n-1}|$.

Proof.

We use induction on n. Let $n = 1$. We have $F_2 = F_1 \cup P(F_1, F_1)$. We know that $F_1 = A$ and that $|P(F_1, F_1)| = |F_1|^2 - |F_1|$, because $(a, a) \notin P(F_1, F_1)$ for $a \in F_1$. Thus, $|F_2| = |F_1|^2 = |A|^2$ which is precisely given by the formula in the theorem statement, i.e. $|F_2| = |F_1|^2 - |F_0|^2 + |F_0| = |A|^2 - 1 + 1 = |A|^2$.

For the induction hypothesis, let us assume the theorem holds for F_n and show that it holds for F_{n+1}. We know that F_n is at least as large as F_{n-1} by definition and thus we can consider the subsets of F_n, F_{n-1} and $F_n \setminus F_{n-1}$ when computing F_{n+1}. Note that the elements of $P(F_{n-1}, F_{n-1})$ are already members of $F_n \setminus F_{n-1}$ and, therefore, do not need to be considered. But we do need to consider the cases $P(F_{n-1}, F_n \setminus F_{n-1})$, $P(F_n \setminus F_{n-1}, F_{n-1})$, and $P(F_n \setminus F_{n-1}, F_n \setminus F_{n-1})$, which have sizes $|F_{n-1}|(|F_n| - |F_{n-1}|)$, $|F_{n-1}|(|F_n| - |F_{n-1}|)$, and $(|F_n| - |F_{n-1}|)^2 - (|F_n| - |F_{n-1}|)$, respectively. Thus, we can compute the size of F_{n+1}:

$$\begin{aligned} |F_{n+1}| = 2 \cdot |F_{n-1}|(|F_n| - |F_{n-1}|) + (|F_n| - |F_{n-1}|)^2 - (|F_n| - |F_{n-1}|) + |F_n| = \\ 2|F_{n-1}||F_n| - 2|F_{n-1}|^2 + (|F_n|^2 - 2|F_n||F_{n-1}| + |F_{n-1}|^2) + |F_{n-1}| = \\ |F_n|^2 - |F_{n-1}|^2 + |F_{n-1}|. \end{aligned}$$

It proves the induction step. See Figure 1 for a geometric proof of the theorem.

□

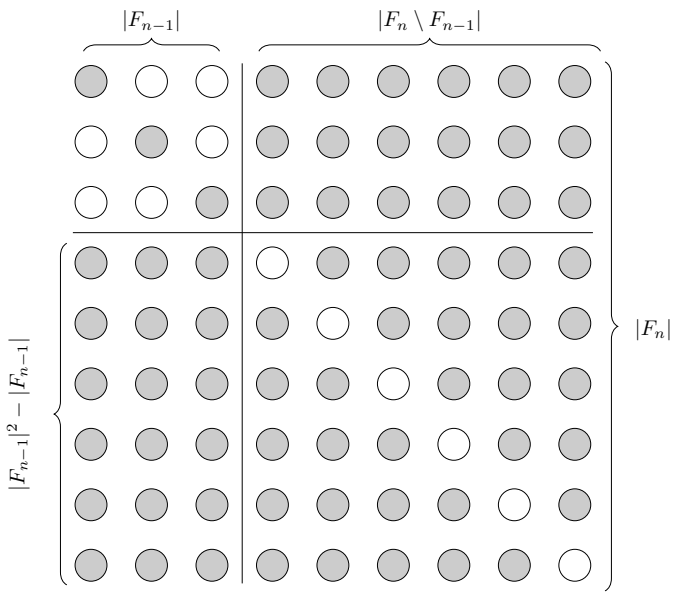

Figure 1: Geometric proof of Theorem 2 for $|F_1| = 3$.

This construct can be pushed a little further by considering the ratio between the large and small squares in Figure 1, where the area of the small square is $|F_{n-1}|^2$ and the area of the large square is $(|F_{n-1}|^2 - |F_{n-1}|)^2 = |F_{n-1}|^4 - 2|F_{n-1}|^3 + |F_{n-1}|^2$.

Theorem 3 *Let F_n be the construction of Theorem 2. Then $|F_n| \approx O(2^{2^n})$.*

Proof.

Consider the ratio between the large and small squares found in Figure 1:

$$\frac{(|F_{n-1}|^2 - |F_{n-1}|)^2}{|F_{n-1}|^2} = |F_{n-1}|^2 - 2 \cdot |F_{n-1}| + 1 = O(n^2) \tag{1}$$

Furthermore, the large square for case n is the small square for case $n+1$. Therefore, the ratio between the small square of case n and large square of case $n+1$ must be $O(n^4)$. The construction is a quadratic recurrence, which is known to have a complexity of $O(2^{2^n})$.

□

To summarize the results of this section, we note that one idempotent symbol is sufficient to construct an anti-unification problem, which has infinitely many $\preceq_I$-incomparable generalizations. A simple example of such a problem is $h(a, b) \triangleq h(b, a)$ for an idempotent h, for which we showed how to construct a minimal complete set of I-generalizations.

However, the construction cannot help us to construct minimal complete set of I-generalizations in the general case. Therefore, this example only says that idempotent generalization with one idempotent function symbol is either infinitary or of type zero. Still, it is an improvement over the result from [33], which says the same for two idempotent symbols.

In the next sections we prove that I-generalization is indeed infinitary, not only for one idempotent symbol, but for an arbitrary number of them. Instead of the S_n construction, tree grammars are used for finite algorithmic description of the minimal complete set of I-generalizations.

4 Generalization Algorithm

In this section we present an idempotent pre-generalization algorithm for construction of a finite representation of the infinite set of the generalizations of a given AUT with an arbitrary number of idempotent symbols.

Our algorithm is given in a rule-based manner, where the rules transform configurations into configurations. A *configuration* is a triple $A;\ S;\ B$, where A is a set of anti-unification problems to be solved, S is a set of already solved anti-unification problems (called the store), and B is a set of bindings, representing the generalization "computed so far". The intuitive idea is to collect such B's at the end and construct from them a regular tree grammar, from which one can read off each generalization. We elaborate on the details later, after the rules are formulated.

It is assumed that all terms in A and S are in I-normal form. The rules are defined as follows:

Dec: **Decomposition**

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

where f is a free function symbol, $n \geq 0$, and $y_1, \ldots, y_n$ are fresh variables.

Explanation: The decomposition rule applies when terms in the selected AUT have the same head. In this case, the generalization variable x, which is there as a (most general) generalization of $f(s_1, \ldots, s_n)$ and $f(t_1, \ldots, t_n)$, gets replaced in B by a more specific generalization $f(y_1, \ldots, y_n)$, where y_i is supposed to generalize s_i and t_i, $1 \leq i \leq n$.

Id-Left: **Idempotent symbol in the left**

$\{x : f(s_1, s_2) \triangleq t\} \cup A;\ S;\ B \Longrightarrow \{y_1 : s_1 \triangleq t,\ y_2 : s_2 \triangleq t\} \cup A;\ S;\ B\{x \mapsto f(y_1, y_2)\}$,

where f is an idempotent function symbol, $head(t)$ is not idempotent, and y_1 and y_2 are fresh variables.

Explanation: The Id-Left rule can be seen as a shorthand of two steps: first, replacing t by its I-equivalent term $f(t, t)$, and then decomposing the obtained AUT $x : f(s_1, s_2) \triangleq f(t, t)$, which leads to the replacement of x in B by a more specific generalization term $f(y_1, y_2)$. Note that here we require the head of t not to be idempotent. For AUTs with idempotent heads in both sides we have a separate rule.

Id-Right: **Idempotent symbol in the right**

$\{x : s \triangleq f(t_1, t_2)\} \cup A;\ S;\ B \Longrightarrow \{y_1 : s \triangleq t_1, y_2 : s \triangleq t_2\} \cup A;\ S;\ B\{x \mapsto f(y_1, y_2)\}$,

where f is an idempotent function symbol, $head(s)$ is not idempotent, and y_1 and y_2 are fresh variables.

Explanation: This rule is similar to the previous one, applying the idempotent expansion in the left-hand side of the selected AUT.

Id-Both 1: **Idempotent symbol in both sides 1**

$\{x : f(s_1, s_2) \triangleq f(t_1, t_2)\} \cup A;\ S;\ B \Longrightarrow$
$\{y_1 : s_1 \triangleq t_1,\ y_2 : s_2 \triangleq t_2\} \cup A;\ S;\ B \cup \{x \mapsto f(y_1, y_2)\}$,

where f is an idempotent function symbol and y_1 and y_2 are fresh variables.

Id-Both 2: **Idempotent symbol in both sides 2**

$\{x : f(s_1, s_2) \triangleq t\} \cup A;\ S;\ B \Longrightarrow$
$\{y_1 : s_1 \triangleq t,\ y_2 : s_2 \triangleq t\} \cup A;\ S;\ B \cup \{x \mapsto f(y_1, y_2)\}$,

where f is an idempotent function symbol, $head(t)$ is idempotent, and y_1 and y_2 are fresh variables.

Id-Both 3: **Idempotent symbol in both sides 3**

$\{x : s \triangleq f(t_1, t_2)\} \cup A;\ S;\ B \Longrightarrow$
$\{y_1 : s \triangleq t_1, y_2 : s \triangleq t_2\} \cup A;\ S;\ B \cup \{x \mapsto f(y_1, y_2)\},$

where f is an idempotent function symbol, $head(s)$ is idempotent, and y_1 and y_2 are fresh variables.

Explanation: These three rules, Id-Both 1, Id-Both 2, and Id-Both 3 are variations of Dec, Id-Left, and Id-Right rules. Rather then substituting the decomposed generalization into the existing bindings, each of them adds a new binding there. This is justified by the fact that these three rules are alternatives to each other. Later, when we will be collecting bindings computed in different ways by these rules, we would like each of these alternatives to be explicitly present in the united set (grammar), in order to be able to construct larger incomparable generalizations in all possible ways. These details will become clearer when we define the generalization algorithm.

Solve: **Solve**

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

where $head(s) \neq head(t)$ and neither $head(s)$ nor $head(t)$ is an idempotent symbol.

Explanation: The condition of the solving rule guarantees that it only applies when no other inference is applicable to the selected AUT. In this case, we move the AUT to the store, recording that for s and t we cannot find a more specific generalization than x. Essentially, this rule captures the mis-matches between the terms we are attempting to anti-unify.

Merge: **Merge**

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

where $s_1 \approx_I s_2$ and $t_1 \approx_I t_2$.

Explanation: Since we aim at computing least general generalizations, we should reuse a single generalization variable instead of two different ones, if they generalize the same (modulo idempotence) difference. In this way, the Merge rule is used to reduce the number of generalization variables to the necessary minimum.

The idempotent pre-generalization algorithm I-PreGen applies the rules exhaustively, starting from the initial state $\{x : s \triangleq t\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x\}$ for the given terms s and t. As we mentioned above, the purpose of the Id-Both rules is to introduce new bindings while Dec, Id-Left, and Id-Right only change the generality of existing bindings. The additional bindings added by the Id-Both rules are used to capture generalizations, which are obtained by combining some of the other generalizations under an idempotent head.

Theorem 4 (Termination) I-PreGen *terminates on any input.*

Proof.

All rules strictly reduce the number of symbols in the set of anti-unification problems to be solved. Hence, the rules can be applied finitely many times and the algorithm terminates.

□

Note that I-PreGen is a non-deterministic algorithm and thus results in a derivation tree. The tree provides us with a set of binding lists, which we can collect from its leaves, i.e., from the configurations of the form $\emptyset; S; B$. They are used in the algorithm I-Gen which we define now.

The idempotent generalization algorithm I-Gen takes as input two terms s and t and performs the following steps:

1. Apply I-PreGen on s and t. After I-PreGen terminates, collect the S's and B's computed in the branches of the derivation tree. Assume that there are n derivation branches, and S_i and B_i, $1 \leq i \leq n$, are respectively the sets of stores and bindings computed in the leaves of those branches.

2. The variable x_{gen} is the root of each B_i. By construction, in each B_i there are no two bindings with the same left-hand side. There is no recursion either. It means that $L(G(B_i))$ consists of a single term. Let it be denoted by $T(B_i)$. Let J be the maximal set of indices $J \subseteq \{1, \ldots, n\}$ such that $rep(T(B_j)) \in Minimize(\cup_{i=1}^{n}\{T(B_i)\})$ for all $j \in J$, i.e., they are terms retained after the minimization of the set $\cup_{i=1}^{n}\{T(B_i)\}$.

3. Apply the Merge rule starting from the configuration $\emptyset;\ \cup_{j\in J}S_j;\ \cup_{j\in J}B_j$ as long as possible. In this process, different choices of selecting AUTs to merge might give different final results, but they all will be the same modulo idempotence and variable renaming. This property is enough to justify referring to the resulting store and binding sets of the final configuration as the STORE and B, respectively. The final configuration is obtained after an exhaustive application of the Merge rule:

$$\emptyset;\cup_{j\in J}S_j;\cup_{j\in J}B_j \Longrightarrow^{*}_{\mathsf{Merge}} \emptyset;\ \mathsf{STORE};\ \mathsf{B}$$

and Merge does not apply to $\emptyset$; STORE; B anymore.

4. Return STORE and $G(\mathsf{B})$, which is a regular tree grammar that corresponds to B. For the given s and t, we will refer to this grammar as $G(s,t)$. Below we will also use $G_b(s,t)$, which is the base grammar $G_b(\mathsf{B})$. Note that unlike $G(s,t)$, the base grammar $G_b(s,t)$ does not contain duplication rules as discussed in Definition 2. Therefore, $L(G_b(s,t))$ contains only those generalizations, which are not represented as a combination of some of the other generalizations under an idempotent head. The latter need the duplication rules.

Based on Theorem 4 and the fact that the construction of $G(\mathsf{B})$ stops, we can conclude that I-Gen is terminating.

By the construction of stores, it is easy to see that STORE does not contain two AUTs $x : s_1 \triangleq t_1$ and $x : s_2 \triangleq t_2$ with the same generalization variable x.

Note that the occurrence of an AUT of the form $x : t \triangleq s$ within a given configuration such that both $head(t)$ and $head(s)$ are idempotent, guarantees that the final set of bindings B will contain at least two bindings with the variable x on the left. This occurs because Id-Both 1, Id-Both 2, and Id-Both 3 add new bindings to B. In such cases, for the same variable we may have two (when only Id-Both 2 and Id-Both 3 apply) or three bindings (when also Id-Both 1 applies) in B. This splitting is the essential aspect of our algorithms allowing us to capture the minimal complete set of least general generalizations.

Each variable that appears in the set of terminals of $G(\mathsf{B})$ is a generalization variable which occurs in the store in one of the leaves of the derivation tree of the algorithm above. To distinguish these variables and nonterminals syntactically, we use the bold face font for the latter.

STORE can tell us how to obtain the initial terms t_1 and t_2 from $r \in L(G(s,t))$: Assume r contains generalization variables $y_1, \ldots, y_n$, and for them STORE contains the AUTs $y_1 : s_1^1 \triangleq s_1^2, \ldots, y_n : s_n^1 \triangleq s_n^2$. Then, from the construction of the algorithm I-Gen we can easily see that $r\sigma_i \approx_I t_i$, where $\sigma_i = \{y_1 \mapsto s_1^i, \ldots, y_n \mapsto s_n^i\}$.

Example 4 *We illustrate how* I-Gen *works on the AUP* $f(a,b) \triangleq f(b,a)$*, where* f *is idempotent. First,* I-PreGen *is applied, which results in three derivations:*

1) $\{x : f(a,b) \triangleq f(b,a)\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x\} \Longrightarrow_{\mathsf{Id\text{-}Both\ 1}}$
$\{x_1 : a \triangleq b,\ x_2 : b \triangleq a\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(x_1,x_2)\} \Longrightarrow^{2}_{\mathsf{Solve}}$
$\emptyset;\ \{x_1 : a \triangleq b,\ x_2 : b \triangleq a\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(x_1,x_2)\}.$

2) $\{x : f(a,b) \triangleq f(b,a)\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x\} \Longrightarrow_{\mathsf{Id\text{-}Both\ 2}}$
$\{y_1 : a \triangleq f(b,a),\ y_2 : b \triangleq f(b,a)\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(y_1,y_2)\} \Longrightarrow_{\mathsf{Id\text{-}Right}}$
$\{z_1 : a \triangleq b,\ z_2 : a \triangleq a,\ y_2 : b \triangleq f(b,a)\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,z_2),y_2)\} \Longrightarrow_{\mathsf{Solve}}$
$\{z_2 : a \triangleq a,\ y_2 : b \triangleq f(b,a)\};\ \{z_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,z_2),y_2)\} \Longrightarrow_{\mathsf{Dec}}$

$\{y_2 : b \triangleq f(b,a)\};\ \{z_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),y_2)\} \Longrightarrow_{\mathsf{Id\text{-}Right}}$
$\{z_3 : b \triangleq b,\ z_4 : b \triangleq a\};\ \{z_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(z_3,z_4))\} \Longrightarrow_{\mathsf{Dec}}$
$\{z_4 : b \triangleq a\};\ \{z_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4))\} \Longrightarrow_{\mathsf{Solve}}$
$\emptyset;\ \{z_1 : a \triangleq b,\ z_4 : b \triangleq a\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4))\}.$

3) $\{x : f(a,b) \triangleq f(b,a)\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x\} \Longrightarrow_{\mathsf{Id\text{-}Both\ 3}}$
$\{u_1 : f(a,b) \triangleq b,\ u_2 : f(a,b) \triangleq a\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(u_1,u_2)\} \Longrightarrow_{\mathsf{Id\text{-}Left}}$
$\{v_1 : a \triangleq b,\ v_2 : b \triangleq b,\ u_2 : f(a,b) \triangleq a\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,v_2),u_2)\} \Longrightarrow_{\mathsf{Solve}}$
$\{v_2 : b \triangleq b,\ u_2 : f(a,b) \triangleq a\};\ \{v_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,v_2),u_2)\} \Longrightarrow_{\mathsf{Dec}}$
$\{u_2 : f(a,b) \triangleq a\};\ \{v_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,b),u_2)\} \Longrightarrow_{\mathsf{Id\text{-}Left}}$
$\{v_3 : a \triangleq a,\ v_4 : b \triangleq a\};\ \{v_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,b),f(v_3,v_4))\} \Longrightarrow_{\mathsf{Dec}}$
$\{v_4 : b \triangleq a\};\ \{v_1 : a \triangleq b\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,b),f(a,v_4))\} \Longrightarrow_{\mathsf{Solve}}$
$\emptyset;\ \{v_1 : a \triangleq b,\ v_4 : b \triangleq a\};\ \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,b),f(a,v_4))\}.$

Hence, the obtained store/bindings pairs are

$$S_1 = \{x_1 : a \triangleq b,\ x_2 : b \triangleq a\}, \qquad B_1 = \{x_{\text{gen}} \mapsto x,\ x \mapsto f(x_1,x_2)\}.$$
$$S_2 = \{z_1 : a \triangleq b,\ z_4 : b \triangleq a\}, \qquad B_2 = \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4))\}.$$
$$S_3 = \{v_1 : a \triangleq b,\ v_4 : b \triangleq a\}, \qquad B_3 = \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(v_1,b),f(a,v_4))\}.$$

From B_1, B_2 and B_3 we see that $T(B_1) = f(x_1,x_1)$, $T(B_2) = f(f(z_1,a),f(b,z_4))$, and $T(B_3) = f(f(v_1,b),f(a,v_4))$. Minimizing the set $\{T(B_1),T(B_2),T(B_3)\}$ gives $\{T(B_2),T(B_3)\}$. Hence, the set of indices of retained stores and bindings is $J = \{2,3\}$, and we should merge variables in the configuration $\emptyset;\ S_2 \cup S_3;\ B_2 \cup B_3$:

$\emptyset;\ S_2 \cup S_3;\ B_2 \cup B_3 =$
$\emptyset;\ \{z_1 : a \triangleq b,\ z_4 : b \triangleq a,\ v_1 : a \triangleq b,\ v_4 : b \triangleq a\};$
$\quad \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4)),\ x \mapsto f(f(v_1,b),f(a,v_4))\} \Longrightarrow_{\mathsf{Merge}}$
$\emptyset;\ \{z_1 : a \triangleq b,\ z_4 : b \triangleq a,\ v_4 : b \triangleq a\};$
$\quad \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4)),\ x \mapsto f(f(z_1,b),f(a,v_4))\} \Longrightarrow_{\mathsf{Merge}}$
$\emptyset;\ \{z_1 : a \triangleq b,\ z_4 : b \triangleq a\};$
$\quad \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4)),\ x \mapsto f(f(z_1,b),f(a,z_4))\}.$

From the last configuration we get

$$\mathsf{STORE} = \{z_1 : a \triangleq b,\ z_4 : b \triangleq a\}$$
$$\mathsf{B} = \{x_{\text{gen}} \mapsto x,\ x \mapsto f(f(z_1,a),f(b,z_4)),\ x \mapsto f(f(z_1,b),f(a,z_4))\}.$$

From B*, we extract first the base grammar $G_b(\mathsf{B}) = \langle \boldsymbol{x}_{\text{gen}}, N, T, R_b \rangle$, where*

$$N = \{\boldsymbol{x}_{\text{gen}}, \boldsymbol{x}\},$$
$$T = \{f, a, b, z_1, z_4\},$$
$$R_b = \{\boldsymbol{x}_{\text{gen}} \to \boldsymbol{x},\ \boldsymbol{x} \to f(f(z_1,a),f(b,z_4)),\ \boldsymbol{x} \to f(f(z_1,b),f(a,z_4))\}.$$

The base grammar gives two generalizations $f(f(z_1,b),f(a,z_4))$ and $f(f(z_1,a),f(b,z_4))$ of our initial problem $f(a,b) \triangleq f(b,a)$. However, $G_b(\mathsf{B})$ does not give generalizations, which are generated from the existing generalizations by taking into account the equational property of idempotence. For instance, the generalization $f(f(f(z_1,b),f(a,z_4)),f(f(z_1,a),f(b,z_4)))$ cannot be obtained from

$G_b(\mathsf{B})$. *For such kind of answers, we need the duplication rules. Since for* $\boldsymbol{x}$ *we have two rules, and* $\boldsymbol{x}$ *is reachable from* $\boldsymbol{x}_{\mathrm{gen}}$ *and from itself, we add two duplication rules to* R_b*: one for* $\boldsymbol{x}_{\mathrm{gen}}$ *and one for* $\boldsymbol{x}$*. It results in the final grammar*

$$G(f(a,b),\ f(b,a)) = G(\mathsf{B}) = \langle \boldsymbol{x}_{\mathrm{gen}}, N, T, R\rangle,$$

where N *and* T *are as in* $G_b(\mathsf{B})$ *and*

$$\begin{aligned} R = \{&\boldsymbol{x}_{\mathrm{gen}} \to \boldsymbol{x},\ \boldsymbol{x}_{\mathrm{gen}} \to f(\boldsymbol{x}_{\mathrm{gen}}, \boldsymbol{x}_{\mathrm{gen}}),\\ &\boldsymbol{x} \to f(f(z_1,b), f(a,z_4)),\ \boldsymbol{x} \to f(f(z_1,a), f(b,z_4)),\ \boldsymbol{x} \to f(\boldsymbol{x},\boldsymbol{x})\}. \end{aligned}$$

To obtain $f(a,b)$ *from any* $r \in L(G(f(a,b)\, f(b,a)))$*, we need to apply the substitution* $\{z_1 \mapsto a,\ z_4 \mapsto b\}$ *to* r*. This substitution is read off from* STORE*. Similarly, to obtain* $f(b,a)$ *from* r*, we need the substitution* $\{z_1 \mapsto b,\ z_4 \mapsto a\}$*, which can be also constructed from* STORE*.*

▶

Example 5 *Application of* I-Gen *to* $g(a, f(a,b)) \triangleq g(a, f(b,a))$*, where* g *is free and* f *is idempotent, starts with* I-PreGen *and performs two decomposition steps:*

$$\begin{aligned} &\{x : g(a, f(a,b)() \triangleq g(a, f(b,a))\};\ \emptyset;\ \{x_{\mathrm{gen}} \mapsto x\} \Longrightarrow_{\mathsf{Dec}}\\ &\{x_1 : a \triangleq a,\ x_2 : f(a,b) \triangleq f(b,a)\};\ \emptyset;\ \{x_{\mathrm{gen}} \mapsto x,\ x \mapsto g(x_1,x_2)\} \Longrightarrow^2_{\mathsf{Dec}}\\ &\{x_2 : f(a,b) \triangleq f(b,a)\};\ \emptyset;\ \{x_{\mathrm{gen}} \mapsto g(a,x_2)\}. \end{aligned}$$

From the obtained configuration, I-PreGen *produces three alternative derivations in the same way as in Example 4. From the leaves of the derivation tree, we get three store/bindings pairs:*

$$\begin{aligned} S_1 &= \{x_3 : a \triangleq b,\ x_4 : b \triangleq a\}, \qquad B_1 = \{x_{\mathrm{gen}} \mapsto g(a,x_2),\ x_2 \mapsto f(x_3,x_4)\}.\\ S_2 &= \{z_1 : a \triangleq b,\ z_4 : b \triangleq a\}, \qquad B_2 = \{x_{\mathrm{gen}} \mapsto g(a,x_2),\ x_2 \mapsto f(f(z_1,a), f(b,z_4))\}.\\ S_3 &= \{v_1 : a \triangleq b,\ v_4 : b \triangleq a\}, \qquad B_3 = \{x_{\mathrm{gen}} \mapsto g(a,x_2),\ x_2 \mapsto f(f(v_1,b), f(a,v_4))\}. \end{aligned}$$

After that, I-Gen *applies minimization, retains* S_2*,* S_3*,* B_2*, and* B_3*, performs merging, and gives*

$$\begin{aligned} \mathsf{STORE} &= \{z_1 : a \triangleq b,\ z_4 : b \triangleq a\}\\ \mathsf{B} &= \{x_{\mathrm{gen}} \mapsto g(a,x_2),\ x_2 \mapsto f(f(z_1,a), f(b,z_4)),\ x_2 \mapsto f(f(v_1,b), f(a,v_4))\}. \end{aligned}$$

From B*,* I-Gen *constructs the grammar*

$$G(g(a, f(a,b)),\ g(a, f(b,a))) = \langle \boldsymbol{x}_{\mathrm{gen}}, N, T, R\rangle,$$

where

$$\begin{aligned} N &= \{\boldsymbol{x}_{\mathrm{gen}},\ \boldsymbol{x}\},\\ T &= \{f, g, a, b, z_1, z_4\},\\ R = \{&\boldsymbol{x}_{\mathrm{gen}} \to g(a,\boldsymbol{x}),\ \boldsymbol{x}_{\mathrm{gen}} \to f(\boldsymbol{x}_{\mathrm{gen}}, \boldsymbol{x}_{\mathrm{gen}}),\\ &\boldsymbol{x} \to f(f(z_1,a), f(b,z_4)),\ \boldsymbol{x} \to f(f(z_1,b), f(a,z_4)),\ \boldsymbol{x} \to f(\boldsymbol{x},\boldsymbol{x})\}. \end{aligned}$$

We have, for instance, $f(g(a, f(f(z_1,b), f(a,z_4))),\ g(a, f(f(z_1,a), f(b,z_4)))) \in L(G(g(a, f(a,b)),\ g(a, f(b,a))))$*. To see that this term is indeed an I-generalization of* $g(a, f(a,b))$ *and* $g(a, f(b,a)))$*, we can read off the matching substitutions from* STORE*:* $\sigma_1 = \{z_1 \mapsto a,\ z_4 \mapsto b\}$ *and* $\sigma_2 = \{z_1 \mapsto b,\ z_4 \mapsto a\}$ *and check:*

$$\begin{aligned} &f(g(a, f(f(z_1,b), f(a,z_4))),\ g(a, f(f(z_1,a), f(b,z_4))))\sigma_1 =\\ &\quad f(g(a, f(f(a,b), f(a,b))),\ g(a, f(f(a,a), f(b,b)))) \approx_I\\ &\quad f(g(a, f(a,b)),\ g(a, f(a,b))) \approx_I g(a, f(a,b)). \end{aligned}$$

$$f(g(a, f(f(z_1, b), f(a, z_4))),\ g(a, f(f(z_1, a), f(b, z_4))))\sigma_2 =$$
$$f(g(a, f(f(b, b), f(a, a))),\ g(a, f(f(b, a), f(b, a)))) \approx_I$$
$$f(g(a, f(b, a)),\ g(a, f(b, a))) \approx_I g(a, f(b, a)).$$

▶

Example 6 *We apply* I-Gen *to* $f(a, f(a, b)) \triangleq f(a, f(b, a))$*, where* f *is idempotent.* I-PreGen *gives the following sets of bindings:*

$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(a, x_2),\ x_2 \mapsto f(f(y_1, a), f(b, y_2))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(a, x_2),\ x_2 \mapsto f(f(y_3, b), f(a, y_4))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(z_1, a)), x_3),\ x_3 \mapsto f(a, f(b, z_2))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(z_1, a)), x_3),\ x_3 \mapsto f(f(a, f(z_1, a)), f(z_3, f(b, z_3)))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(z_1, a)), x_3),\ x_3 \mapsto f(f(a, z_4), x_5),\ x_5 \mapsto f(z_1, z_4)\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(z_1, a)), x_3),\ x_3 \mapsto f(f(a, z_4), x_5),$$
$$x_5 \mapsto f(f(z_1, a), f(b, z_4))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(z_1, a)), x_3),\ x_3 \mapsto f(f(a, z_4), x_5),$$
$$x_5 \mapsto f(f(z_1, b), f(a, z_4))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(a, v_1)), x_4),\ x_4 \mapsto f(v_2, f(a, v_1))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(a, v_1)), x_4),\ x_4 \mapsto f(f(v_3, a), x_6),\ x_6 \mapsto f(v_3, v_1)\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(a, v_1)), x_4),\ x_4 \mapsto f(f(v_3, a), x_6),$$
$$x_6 \mapsto f(f(v_3, a), f(b, v_1))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(a, v_1)), x_4),\ x_4 \mapsto f(f(v_3, a), x_6),$$
$$x_6 \mapsto f(f(v_3, b), f(a, v_1))\},$$
$$\{x_{\text{gen}} \mapsto x_1,\ x_1 \mapsto f(f(a, f(a, v_1)), x_4),\ x_4 \mapsto f(f(v_4, f(v_4, b)), f(a, f(a, v_1)))\}.$$

From these sets, I-Gen *constructs the grammar*

$$G(f(a, f(a, b)),\ f(a, f(b, a))) = \langle \boldsymbol{x}_{\text{gen}}, N, T, R\rangle,$$

where

$$N = \{\boldsymbol{x}_{\text{gen}}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_6\},$$
$$T = \{f, a, b, y_1, y_2\},$$
$$R = \{\boldsymbol{x}_{\text{gen}} \to \boldsymbol{x}_1,\ \boldsymbol{x}_{\text{gen}} \to f(\boldsymbol{x}_{\text{gen}}, \boldsymbol{x}_{\text{gen}}),$$
$$\boldsymbol{x}_1 \to f(a, \boldsymbol{x}_2),\ \boldsymbol{x}_1 \to f(f(a, f(y_1, a)), \boldsymbol{x}_3),\ \boldsymbol{x}_1 \to f(f(a, f(a, y_2)), \boldsymbol{x}_4),$$
$$\boldsymbol{x}_1 \to f(\boldsymbol{x}_1, \boldsymbol{x}_1),$$
$$\boldsymbol{x}_2 \to f(f(y_1, a), f(b, y_2)),\ \boldsymbol{x}_2 \to f(f(y_1, b), f(a, y_2)),\ \boldsymbol{x}_2 \to f(\boldsymbol{x}_2, \boldsymbol{x}_2),$$
$$\boldsymbol{x}_3 \to f(a, f(b, y_2)),\ \boldsymbol{x}_3 \to f(f(a, f(y_1, a)), f(y_2, f(b, y_2))),\ \boldsymbol{x}_3 \to f(f(a, y_2), \boldsymbol{x}_5),$$
$$\boldsymbol{x}_3 \to f(\boldsymbol{x}_3, \boldsymbol{x}_3),$$
$$\boldsymbol{x}_4 \to f(f(y_1, a), \boldsymbol{x}_6),\ \boldsymbol{x}_4 \to f(f(y_1, f(y_1, b)), f(a, f(a, y_2))),\ \boldsymbol{x}_4 \to f(\boldsymbol{x}_4, \boldsymbol{x}_4),$$
$$\boldsymbol{x}_5 \to f(f(y_1, a), f(b, y_2)),\ \boldsymbol{x}_5 \to f(f(y_1, b), f(a, y_2)),\ \boldsymbol{x}_5 \to f(\boldsymbol{x}_5, \boldsymbol{x}_5),$$
$$\boldsymbol{x}_6 \to f(f(y_1, a), f(b, y_2)),\ \boldsymbol{x}_6 \to f(f(y_1, b), f(a, y_2)),\ \boldsymbol{x}_6 \to f(\boldsymbol{x}_6, \boldsymbol{x}_6)\}.$$

$\mathsf{STORE} = \{y_1 : a \triangleq b, y_2 : b \triangleq a\}$.

We can see that $f(a, f(f(f(y_1, a), f(b, y_2)), f(f(y_1, b), f(a, y_2)))) \in L(G(f(a, f(a, b)),\ f(a, f(b, a))))$*:*

$$\boldsymbol{x}_{\text{gen}} \to \boldsymbol{x}_1 \to f(a, \boldsymbol{x}_2) \to f(a, f(\boldsymbol{x}_2, \boldsymbol{x}_2)) \to f(a, f(f(f(y_1, a), f(b, y_2)), \boldsymbol{x}_2))$$

$$\to f(a, f(f(f(y_1, a), f(b, y_2)), f(f(y_1, b), f(a, y_2)))).$$

▶

Example 7 *Let f and g be idempotent symbols. Then for $f(a, b) \triangleq g(a, c)$ the regular tree grammar computed by* I-Gen *is $\langle \boldsymbol{x}_{\text{gen}}, N, T, R\rangle$, where*

$$\begin{aligned} N &= \{\boldsymbol{x}_{\text{gen}}, \boldsymbol{x}\},\\ T &= \{f, g, a, y_1, y_2, y_3\},\\ R &= \{\boldsymbol{x}_{\text{gen}} \to \boldsymbol{x},\ \boldsymbol{x}_{\text{gen}} \to f(\boldsymbol{x}_{\text{gen}}, \boldsymbol{x}_{\text{gen}}),\ \boldsymbol{x}_{\text{gen}} \to g(\boldsymbol{x}_{\text{gen}}, \boldsymbol{x}_{\text{gen}}),\\ &\quad \boldsymbol{x} \to f(g(a, y_1), g(y_2, y_3)),\ \boldsymbol{x} \to g(f(a, y_2), f(y_1, y_3)),\ \boldsymbol{x} \to f(\boldsymbol{x}, \boldsymbol{x}),\ \boldsymbol{x} \to g(\boldsymbol{x}, \boldsymbol{x})\}. \end{aligned}$$

STORE *is* $\{y_1 : a \triangleq c,\ y_2 : b \triangleq a,\ y_3 : b \triangleq c\}$. ▶

I-Gen is sound, as the following theorem shows:

Theorem 5 (Soundness) *Let s and t be two terms and $G(s, t)$ be the regular tree grammar computed by* I-Gen *for them. Then any term in $L(G(s, t))$ is an I-generalization of s and t.*

Proof.

First, notice that if $r_1, r_2 \in L(G(s, t))$ are I-generalizations of s and t, then $f(r_1, r_2)$ is also an I-generalization of s and t, where f is idempotent. Indeed, assume r_1 and r_2 generalize s. Then there exist substitutions σ_1 and σ_2 such that $r_i\sigma_i \approx_I s$, $i = 1, 2$. As we saw, we can actually read off these σ's from STORE. It implies that if $x \in dom(\sigma_1) \cap dom(\sigma_2)$, then $x\sigma_1 = x\sigma_2$. Let ϑ be a substitution defined as $x\vartheta = x\sigma_1$ if $x \in dom(\sigma_1)$ and $x\vartheta = x\sigma_2$ otherwise. Then we have $f(r_1, r_2)\vartheta = f(r_1\sigma_1, r_2\sigma_2) \approx_I f(s, s) \approx_I s$, which shows that $f(r_1, r_2)$ is an I-generalization of s. For t it is analogous.

Hence, it remains to prove that every term in $L(G_b(s, t))$ is an I-generalization of s and t. For the elements of $L(G(s, t))$ that are generated with the help of duplication rules, the theorem follows from what we just showed.

The terms of $L(G_b(s, t))$ are precisely those derived using I-PreGen (i.e., the right hand sides of bindings in binding sets computed by I-PreGen) and thus we can prove the theorem by well-founded induction over the length of the derivation of terms in $L(G_b(s, t))$.

Let us assume that the proposition holds for each r, which is derived by I-PreGen in at most n steps and prove it for the case of $n + 1$ steps. The next step depends on the rule of our algorithm, applicable to the initial configuration $\{x : s \triangleq t\}$; $\emptyset$; $\{x_{\text{gen}} \mapsto x\}$. Therefore, we handle each case independently.

Dec: If $s = a$ and $t = a$, the decomposition rule gives the final configuration $\emptyset$; $\emptyset$; $\{x_{\text{gen}} \mapsto a\}$, which implies $L(G_b(s, t)) = \{a\}$ and the theorem is true.

Now assume $s = g(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_m)$, where g is free. Take one of the derivations:

$$\begin{aligned} &\{x : g(s_1, \ldots, s_m) \triangleq g(t_1, \ldots, t_m)\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x\} \Longrightarrow_{\mathsf{Dec}}\\ &\{x_1 : s_1 \triangleq t_1, \ldots, x_m : s_m \triangleq t_m\};\ \emptyset;\ \{x_{\text{gen}} \mapsto g(x_1, \ldots, x_m)\} \Longrightarrow^n \emptyset;\ S;\ B. \end{aligned}$$

We have $L(G_b(B)) \subseteq L(G_b(s, t))$. The part of this derivation after the first step can be represented as a combination of m derivations of length at most n:

$$\{x_i : s_i \triangleq t_i\};\ \emptyset;\ \{y_i \mapsto x_i\} \Longrightarrow_{R_1^i} \cdots \Longrightarrow_{R_{n_i}^i} \emptyset;\ S_i;\ \{y_i \mapsto l_i'\} \cup B_i', \tag{2}$$

where $1 \le i \le m$, $n_1 + \cdots + n_m \le n$, and R_k^i is the rule name used in the ith derivation at kth step. The mentioned combination can be expressed, e.g., like this:

$$\{x_1 : s_1 \triangleq t_1, \ldots, x_m : s_m \triangleq t_m\};\ \emptyset;$$

$$
\begin{aligned}
&\{x_{\text{gen}} \mapsto g(y_1,\ldots,y_m), y_1 \mapsto x_1,\ldots,y_m \mapsto x_m\} \Longrightarrow_{R_1^1} \cdots \Longrightarrow_{R_{n_1}^1} \\
&\{x_2 : s_2 \triangleq t_2,\ldots,x_m : s_m \triangleq t_m\};\ S_1; \\
&\quad \{x_{\text{gen}} \mapsto g(y_1,\ldots,y_m), y_1 \mapsto l_1',\ldots,y_m \mapsto x_m\} \cup B_1' \\
&\Longrightarrow_{R_1^2} \cdots \Longrightarrow_{R_{n_2}^2} \cdots \Longrightarrow_{R_1^m} \cdots \Longrightarrow_{R_{n_m}^m} \\
&\emptyset;\ S_1 \cup \cdots \cup S_m; \\
&\quad \{x_{\text{gen}} \mapsto g(y_1,\ldots,y_m), y_1 \mapsto l_1',\ldots,y_m \mapsto l_m'\} \cup B_1' \cup \cdots \cup B_m' \Longrightarrow^*_{\mathsf{Merge}} \\
&\emptyset;\ S;\ \{x_{\text{gen}} \mapsto g(y_1,\ldots,y_m), y_1 \mapsto l_1,\ldots,y_m \mapsto l_m\} \cup B_1 \cup \cdots \cup B_m.
\end{aligned}
$$

Although this derivation and the derivation $\{x_1 : s_1 \triangleq t_1,\ldots,x_m : s_m \triangleq t_m\};\ \emptyset;\{x_{\text{gen}} \mapsto g(x_1,\ldots,x_m)\} \Longrightarrow^n \emptyset;\ S;\ B$ syntactically slightly differ from each other (the presence of y's in the first one is the difference), they generate equivalent tree grammars: the languages generated by them are the same (up to renaming of store variables).

By the induction hypothesis, the elements of the language $L(G_b(s_i,t_i)) = L(G_b(\{y_i \mapsto l_1'\} \cup B_i'))$ are I-generalizations of s_i and t_i. By the tree grammar definition, it implies that the elements of the language $L(G_b(\{x_{\text{gen}} \mapsto g(y_1,\ldots,y_m), y_1 \mapsto l_1',\ldots,y_m \mapsto l_m'\} \cup B_1' \cup \cdots \cup B_m'))$ are I-generalizations of $s = g(s_1,\ldots,s_m)$ and $t = g(t_1,\ldots,t_m)$. If there are two AUTs in $S_1 \cup \cdots \cup S_m$ which are subject of Merge, it indicates that in s there are two positions p_s' and p_s'' containing I-equivalent subterms, in t there are two positions p_t' and p_t'' containing I-equivalent subterms, the subterms $s|_{p_s'}$ and $t|_{p_t'}$ are generalized by a variable v', and the subterms $s|_{p_s''}$ and $t|_{p_t''}$ are generalized by another variable v''. However, since $s|_{p_s'} \approx_I s|_{p_s''}$ and $t|_{p_t'} \approx_I t|_{p_t''}$, one can just use v' in place of v'' in a generalization of s and t and still have their generalization. Hence, the sequence of Merge rule applications transforms generalizations into generalizations, meaning that every element of the set $L(G_b(x_{\text{gen}} \mapsto g(y_1,\ldots,y_m), y_1 \mapsto l_1,\ldots,y_m \mapsto l_m\} \cup B_1 \cup \cdots \cup B_m)) = L(G_b(B))$ is an I-generalization of s and t. Since the derivation ending with B was chosen arbitrarily, by construction of $L(G_b(s,t))$ we get that this set consists of I-generalizations of s and t.

Solve: If the AUT for s and t is transformed by this rule, the final configuration is $\emptyset;\ \{x : s \triangleq t\};\ \{x_{\text{gen}} \mapsto x\}$, from which we have $L(G_b(s,t)) = \{x\}$ and the theorem holds trivially.

The rules for idempotent heads: Id-Left, Id-Right, Id-Both 1, Id-Both 2, Id-Both 3. Reasoning for these cases is similar to the reasoning for the case with the decomposition rule above. There is an extra detail to be taken into account: for Id-Left, for instance, we should observe that the following fact holds:

Fact 1 If $s = f(s_1,s_2)$, f is idempotent, and r_i is an I-generalization for s_i and t, $i = 1,2$, then $f(r_1,r_2)$ is an I-generalization of s and t.

We spell out the Id-Left case in a bit more detail. Let f be idempotent, $s = f(s_1,s_2)$, $head(t)$ not be idempotent, and consider a derivation that starts with Id-Left rule:

$$
\begin{aligned}
&\{x : f(s_1,s_2) \triangleq t\};\ \emptyset;\ \{x_{\text{gen}} \mapsto x\} \Longrightarrow_{\mathsf{Id\text{-}Left}} \\
&\{x_1 : s_1 \triangleq t,\ x_2 : s_2 \triangleq t\};\ \emptyset;\ \{x_{\text{gen}} \mapsto f(x_1,x_2)\} \Longrightarrow^n \emptyset;\ S;\ B.
\end{aligned}
$$

We have $L(G_b(B)) \subseteq L(G_b(s,t))$. The part of this derivation after the first step can be represented as a combination of two derivations of length at most n:

$$
\{x_i : s_i \triangleq t\};\ \emptyset;\ \{y_i \mapsto x_i\} \Longrightarrow_{R_1^i} \cdots \Longrightarrow_{R_{n_i}^i} \emptyset;\ S_i;\ \{y_i \mapsto l_i'\} \cup B_i', \tag{3}
$$

where $1 \le i \le 2$, $n_1 + n_2 \le n$, and R_k^i is the rule name used in the ith derivation at kth step.

By the reasoning similar to the Dec rule case above, using the induction hypothesis, we conclude the elements of the language $L(G_b(s_i,t)) = L(G_b(\{y_i \mapsto l_1'\} \cup B_i'))$ are I-generalizations of s_i and t. By the tree grammar definition, using Fact 1, it implies that the elements of the language $L(G_b(\{x_{\text{gen}} \mapsto f(y_1,y_2), y_1 \mapsto l_1', y_2 \mapsto l_2'\} \cup B_1' \cup B_2'))$ are I-generalizations of $s = f(s_1,s_2)$ and t. The rest of the proof of this case is similar to the one for Dec.

As for the Id-Both rules, note that for $L(G_b(B))$ it does not make a difference whether B was computed by updates (i.e., rules applying a substitution to bindings as $B_i\{y \mapsto l\}$) or by expansion (i.e., rules adding a new binding as $B_i \cup \{y \mapsto l\}$). If nothing else is different, the languages $L(G_b(B_i\{y \mapsto l\}))$ and $L(G_b(B_i \cup \{y \mapsto l\})$ are the same. This argument we used in the proof of soundness for Dec rule, and the same argument can be used to make the proof of soundness for the Id-Both rules similar to those for Dec or Id-Left.

Merge: This rule transforms an I-generalization of s and t into another, less general I-generalization of s and t. It finishes the proof.

□

The algorithm I-Gen is also complete:

Theorem 6 (Completeness) *Let t_1 and t_2 be two terms and $G(t_1, t_2)$ be the regular tree grammar computed by* I-Gen *for them. Let r be an I-generalization of t_1 and t_2. Then there exists $s \in L(G(t_1, t_2))$ such that $r \preceq_I s$.*

Proof.

By structural induction on r. We assume that t_1 and t_2 are in I-normal form and do not share variables. The latter is not a restriction since we can treat variables in t_1 and t_2 as constants. Note also that the simplification involved in the construction of $G(t_1, t_2)$ by *Minimize* does not violate the claim of the theorem, since its effect is to remove a more general term from $L(G(t_1, t_2))$ while keeping less general ones there.

Case 1. r **is a variable** It implies that $head(t_1) \neq head(t_2)$ and none of those heads are idempotent. We use Solve and get that $L(G(t_1, t_2))$ contains a variable, which proves this case.

Case 2. r **is a constant** let r be a. Then both t_1 and t_2 should be a and by Dec rule we obtain $a \in L(G(t_1, t_2))$.

Case 3. $r = f(r_1, \ldots, r_n)$ **and** f **is free** Then t_1 and t_2 both have f as the head. Let $t_1 = f(t'_1, \ldots, t'_n)$ and $s_2 = f(t''_1, \ldots, t''_n)$. Then r_i is an I-generalization of t'_i and t''_i. It might happen that for some $1 \leq j, k, \leq n$, the terms r_j and r_k share some (terminal) variables, e.g., z. It indicates that t'_j and t'_k have a common subterm l', and t''_j and t''_k have a common subterm l''. The occurrence of l' in t'_j (resp. in t'_k) corresponds to the occurrence of l'' in t''_j (resp. in t''_k). These are those subterms that are generalized by the variable z shared between r_j and r_k.

The original anti-unification problem between t_1 and t_2 can be transformed by the Dec rule, which gives rise to new anti-unification problems $y_i : t'_i \triangleq t''_i$, $1 \leq i \leq n$. If we assume that each $y_i : t'_i \triangleq t''_i$ is a new AUT and run I-Gen for each of them, we obtain $G(t'_i, t''_i)$, $1 \leq i \leq n$.

By the induction hypothesis, there exists $s_i \in L(G(t'_i, t''_i))$ such that $r_i \preceq_I s_i$. Hence, each $G(t'_i, t''_i)$ contains a rule $y_{\text{gen}_i} \to q_i$, such that y_{gen_i} is the axiom and s_i is obtained from q_i, where q_i is either y_i or an instance of y_i. Then s_j contains a subterm l_j and s_k contains a subterm l_k, which are instances of z. Since at this stage s_j and s_k are independent, those subterms are not necessarily the same, but each of them is a computed generalization of l' and l''. We can choose the computing derivations in such a way that l_j and l_k are obtained by exactly the same applications of the rules. This is possible because they generalize the same terms, and the derivations are guided by those terms. Therefore, if l_j and l_k are not the same, they would differ by the names of fresh (terminal) variables generated during the two derivations.

Now we need to put those separate derivations together to show that there exists the desired $s \in L(G(t_1, t_2))$. Essentially, the only required addition would be to consider the necessity of the Merge rule, which would make sure that the subterms such as l_j and l_k above are generalized in the same way. Then the grammar $G(t_1, t_2)$ will contain a rule $x_{\text{gen}} \to f(q_1, \ldots, q_n)$, where the q_i's are as above with the difference that terms s_j and s_k they generate may have shared

(terminal) variables. Let $\hat{s}_i$, $1 \leq i \leq n$ be the instance of the corresponding s_i under such variable-sharing if there exists one. Otherwise, $\hat{s}_i = s_i$. Hence, for all terms in $L(G(t'_i, t''_i))$, and, in particular, for $\hat{s}_i$, $1 \leq i \leq n$, we have $s = f(\hat{s}_1, \ldots, \hat{s}_n) \in L(G(t_1, t_2))$. On the other hand, we have $r = f(r_1, \ldots, r_n) \preceq_I f(\hat{s}_1, \ldots, \hat{s}_n)$, which finishes the proof of this case.

Case 4. $r = f(r_1, r_2)$ **and** f **is idempotent** Then we have the following subcases:

1. $t_1 = f(t'_1, t'_2)$, $t_2 = f(t''_1, t''_2)$ and r_i is an I-generalization of t'_i and t''_i, $i = 1, 2$. In this case we use Id-Both 1 rule and proceed similarly to Case 3 above.
2. $t_1 = f(t'_1, t'_2)$ and r_i is a generalization of t'_i and t_2, $i = 1, 2$. Depending whether $head(t_2)$ is idempotent or not, we proceed either by Id-Left or Id-Both 1 and reason similarly to Case 3.
3. $t_2 = f(t''_1, t''_2)$ and r_i is a generalization of t_1 and t''_i, $i = 1, 2$. Depending whether $head(t_1)$ is idempotent or not, we proceed either by Id-Right or Id-Both 2 and reason similarly to Case 3.
4. Both r_1 and r_2 are I-generalizations of t_1 and t_2. (It means that there exist substitutions σ_1 and σ_2 such that $r_1\sigma_1 \approx_I r_2\sigma_1 \approx_I t_1$ and $r_1\sigma_2 \approx_I r_2\sigma_2 \approx_I t_2$.) By the induction hypothesis, we have $s_1, s_2 \in L(G(t_1, t_2))$ such that $r_i \preceq_I s_i$, $i = 1, 2$. By the construction of $G(t_1, t_2)$, if $s_1, s_2 \in G(t_1, t_2)$, then $f(s_1, s_2) \in G(t_1, t_2)$. Since $f(r_1, r_2) \preceq_I f(s_1, s_2)$, this case is also proved.

□

Let σ_{t_1} and σ_{t_2} be substitutions obtained from STORE after I-Gen has been applied to t_1 and t_2:

$$\sigma_{t_1} := \{x \mapsto l \mid x : l \triangleq r \in \mathsf{STORE} \text{ for some } r\},$$
$$\sigma_{t_2} := \{x \mapsto r \mid x : l \triangleq r \in \mathsf{STORE} \text{ for some } l\}.$$

Then the following theorem holds:

Theorem 7 *$L(G(t_1, t_2))\sigma_{t_i}$ is the I-equivalence class for the term t_i, $i = 1, 2$.*

Proof.

By Theorem 5, every element $t \in L(G(t_1, t_2))$ is an I-generalization of t_1 and t_2. The only instantiable variables in t are those which are the generalization variables in STORE, i.e., those in the domain of σ_{t_i}. By the Solve rule, if $x : l \triangleq r$ is in the store, x generalizes an occurrence of l in t_1 and an occurrence of r in t_2. Then we get $t\sigma_{t_i} \approx_I t_i$, and, hence, $L(G(t_1, t_2))\sigma_{t_i}$ is a subset of the I-equivalence class for t_i, $i = 1, 2$. The other inclusion follows from the Completeness Theorem (Theorem 6).

□

5 Idempotent Generalization is Infinitary

To show that idempotent generalization is infinitary, we prove that for any terms s and t the language $L(G(s, t))$ is a complete and minimal set of I-generalizations of s and t with respect to a given idempotent equational theory.

From Theorem 5 and Theorem 6, we know that $L(G(s, t))$ is a complete set of I-generalizations of s and t. Hence, it remains only to show that $L(G(s, t))$ is minimal.

We assume that $L(G(s, t))$ is I-normalized (i.e., $nf_I(L(G(s, t)) = L(G(s, t))$) and proceed as follows. First, we show that $L(G_b(s, t))$ is minimal (i.e., it does not contain distinct terms that are comparable by $\preceq_I$). Next, to each term r in $L(G(s, t))$ we associate a natural number $dra(r)$ which indicates the maximal number of applications of the duplicating rules (rules from $R \setminus R_b$) needed

to construct r, and define $L_n(G(s,t)) = \{r \mid r \in L(G(s,t)) \text{ and } dra(r) \leq n\}$. These are finite languages for all $n \geq 0$, and $L_0(G(s,t)) = L(G_b(s,t))$. Then we prove that for all $n \geq 0$, $L_n(G(s,t))$ is minimal. Since $L_n(G(s,t)) \subseteq L_{n+1}(G(s,t))$ for all $n \geq 0$ and $\cup_{n=0}^{\infty} L_n(G(s,t)) = L(G(s,t))$, it implies that $L(G(s,t))$ is minimal.

Minimality of $L(G_b(s,t))$ follows from the construction, since the *Minimize* function makes sure that no two terms generated by the same nonterminal are comparable with respect to $\preceq_I$. Before presenting our proof of minimality of $L(G(s,t))$, we provide the following helpful lemma, which highlights an interesting property of idempotent function symbols: if instances of generalizations are not generalizations anymore, one can not make a generalization out of those instances again by putting them under an idempotent symbol. It is an essential observation. While our base grammar is minimal, our final grammar would not be such if two generalizations were transformable via substitution into another generalization when both are direct subterms of an idempotent function symbol. Lemma 2 shows that it is impossible.

Lemma 2 *Let $s \triangleq t$ be an AUP over a signature Σ which may contain idempotent function symbols. Let t_1 and t_2 be $\preceq_I$-incomparable generalizations of $s \triangleq t$ and σ be a substitution such that $t_1\sigma$ and $t_2\sigma$ are not generalizations of $s \triangleq t$. Then for any idempotent function symbol $f \in \Sigma$, the term $f(t_1\sigma, t_2\sigma)$ is not a generalization of $s \triangleq t$.*

Proof.

We perform induction over the term depth of t_1 and t_2.

Case: ($\mathbf{BC}_1$) If $dep(t_1) = dep(t_2) = 1$, then one of the following must hold:

- They are the same constant, i.e. $t_1 = t_2$.
- They are both variables.
- One is a constant and the other one is a variable.

In each case the terms are not $\preceq_I$-incomparable and, thus, trivially satisfying the lemma.

Case: ($\mathbf{BC}_2$) Let us assume without loss of generality that the lemma holds when $dep(t_1) = n$ and $dep(t_2) = 1$. We show that it holds when $dep(t_1) = n+1$ and $dep(t_2) = 1$. Once again, the terms are not $\preceq_I$-incomparable and, thus, trivially satisfy the Lemma.

Case: ($\mathbf{IH}_1$) Let us assume that the lemma holds for $dep(t_1) = n$ and $dep(t_2) = m$ and show that it holds for $dep(t_1) = n+1$ and $dep(t_2) = m+1$.

Case 1 Assume that $t_1 = g(s_1, \cdots, s_k)$ and $t_2 = g(r_1, \cdots, r_k)$ for a free function symbol g. This implies that $s = g(s'_1, \cdots, s'_k)$ and $t = g(r'_1, \cdots, r'_k)$. Thus, for $f(t_1\sigma, t_2\sigma)$ to be a generalization of s and t, it is necessarily the case that $t_1\sigma = t_2\sigma$, implying, without loss of generality, that $f(t_1\sigma, t_2\sigma) = t_1\sigma$ and, thus, contradicting our assumption that $t_1\sigma$ is not a generalization.

Case 2 Assume that $t_1 = g(s_1, s_2)$ and $t_2 = g(r_1, r_2)$ where g is idempotent. This results in two distinct cases depending on s and t.

Case 2a Let us further assume that $s = g(s'_1, s'_2)$ and $t = g(r'_1, r'_2)$. We assume that $g(t_1\sigma, t_2\sigma)$ is a generalization of s and t and reach a contradiction. In order to show a contradiction, we consider the four possible ways t_1 and t_2 generalize $s \triangleq t$, given in the assumption table.

Assumption Table

- **Assumption 2aa:** s_1 generalizes $s'_1 \triangleq r'_1$, s_2 generalizes $s'_2 \triangleq r'_2$, r_1 generalizes $s'_1 \triangleq r'_1$, and r_2 generalizes $s'_2 \triangleq r'_2$.
- **Assumption 2ab:** s_1 generalizes $s'_1 \triangleq r'_1$, s_2 generalizes $s'_2 \triangleq r'_2$, and both r_1 and r_2 generalize $s \triangleq t$.
- **Assumption 2ac:** both s_1 and s_2 generalize $s \triangleq t$, r_1 generalizes $s'_1 \triangleq r'_1$, and r_2 generalizes $s'_2 \triangleq r'_2$.
- **Assumption 2ad:** both s_1 and s_2 generalize $s \triangleq t$, and both r_1 and r_2 generalize $s \triangleq t$.

Case 2aa In this case, given that $t_1\sigma$ and $t_2\sigma$ do not generalize $s \triangleq t$, it is necessarily the case that $s_1\sigma$ does not generalize $s'_1 \triangleq r'_1$, $s_2\sigma$ does not generalize $s'_2 \triangleq r'_2$, $r_1\sigma$ does not generalize $s'_1 \triangleq r'_1$, and $r_2\sigma$ does not generalize $s'_2 \triangleq r'_2$. Therefore, there does not exist θ_1 and θ_2 such that $g(t_1\sigma, t_2\sigma)\theta_1 = g(s, s) = s$ and $g(t_1\sigma, t_2\sigma)\theta_2 = g(t, t) = t$. But since we assumed that $g(t_1\sigma, t_2\sigma)$ generalizes s and t, there should exist θ_1 and θ_2 such that $g(t_1\sigma, t_2\sigma)\theta_1 = g(g(s'_1, s'_1), g(s'_2, s'_2)) = s$ and $g(t_1\sigma, t_2\sigma)\theta_2 = g(g(r'_1, r'_1), g(r'_2, r'_2)) = t$. However, This would imply that $t_1\sigma$ is a generalization of $s'_1 \triangleq r'_1$ and $t_2\sigma$ is a generalization of $s'_2 \triangleq r'_2$, contradicting **Assumption 2aa**.

Case 2ab In this case, given that $t_1\sigma$ and $t_2\sigma$ do not generalize $s \triangleq t$, it is necessarily the case that $s_1\sigma$ does not generalize $s'_1 \triangleq r'_1$, $s_2\sigma$ does not generalize $s'_2 \triangleq r'_2$, $r_1\sigma$ does not generalize $s \triangleq t$, and $r_2\sigma$ does not generalize $s \triangleq t$. Using a similar argument as the one used for (2aa), we can get a contradiction to **Assumption 2ab** by just considering $t_1\sigma$.

Case 2ac In this case, given that $t_1\sigma$ and $t_2\sigma$ do not generalize $s \triangleq t$, it is necessarily the case that $s_1\sigma$ does not generalize $s \triangleq t$, $s_2\sigma$ does not generalize $s \triangleq t$, $r_1\sigma$ does not generalize $s'_1 \triangleq r'_1$, and $r_2\sigma$ does not generalize $s'_2 \triangleq r'_2$. Using a similar argument as the one used for (2aa), we can get a contradiction to **Assumption 2ac** by just considering $t_2\sigma$.

Case 2ad In this case, given that $t_1\sigma$ and $t_2\sigma$ do not generalize $s \triangleq t$, it is necessarily the case that $s_1\sigma$ does not generalize $s \triangleq t$, $s_2\sigma$ does not generalize $s \triangleq t$, $r_1\sigma$ does not generalize $s \triangleq t$, and $r_2\sigma$ does not generalize $s \triangleq r$. This case is an instance of the induction hypothesis and thus contradicts to **Assumption 2ad**.

Case 2b Let us assume that $s = g(s'_1, s'_2)$ and $t = g(r'_1, r'_2)$. We assume that $f(t_1\sigma, t_2\sigma)$ with $f \neq g$ is a generalization of s and t and reach a contradiction. This case is similar to (2a) except that there are fewer subcases to consider. For instance, if θ is a substitution such that $f(t_1\sigma, t_2\sigma)\vartheta \approx_I s$, then the outermost occurrence of f does not occur in s after idempotent normalization of $f(t_1\sigma, t_2\sigma)\theta$. Thus, we would only need to consider an assumption analogous to **Assumption 2aa** for Case 2b and reach a contradiction in a similar fashion as it was reached for Case 2aa.

Case 3 Assume without loss of generality that $t_1 = g(s_1, \cdots, s_k)$ and $t_2 = f(r_1, r_2)$ such that g is a free function symbol and f is idempotent. This implies that $s = g(s'_1, \cdots, s'_k)$ and $t = g(r'_1, \cdots, r'_k)$. Furthermore, it implies that r_1 and r_2 must be generalizations of s and t, i.e., there must exist θ_1 and θ_2 such that $f(r_1, r_2)\theta_1 = f(s, s) = s$ and $f(r_1, r_2)\theta_2 = f(t, t) = t$. Thus, for $f(t_1\sigma, t_2\sigma)$ to be a generalization of s and t, it is necessarily the case that $r_1\sigma$ and $r_2\sigma$ to be generalizations of s and t. However, by the induction hypothesis this results in a contradiction.

We have covered all possible cases. Hence, the lemma is proved.

□

Hence, Lemma 2 showed that two generalizations cannot be transformable via substitution into another generalization when both are direct subterms of an idempotent function symbol. Lemma 3 and later Theorem 8 discuss the usage of this property for proving minimality.

In particular, Lemma 3 uses Lemma 2 to show that for any AUP $s \triangleq t$, any generalization in $L_{n+1}(G(s,t))$ is either incomparable to generalizations in $L_n(G(s,t))$ or it is I-equivalent to a generalization of $L_n(G(s,t))$. Thus, we can only introduce new generalizations when allowing an additional application of duplication, otherwise we produce I-equivalent generalizations. We can never produce a generalization which is less general than an existing one (in our construction).

Lemma 3 considers the case when a generalization g is constructed from existing generalizations g_1 and g_2 in such a way that the newly constructed generalization g is I-comparable to the generalization from which it was constructed, that is either $g_1 \prec_I g$ or $g_2 \prec_I g$ holds. The existence of such less general generalizations implies the existence of an infinite chain of strictly less generality among generalizations. If such a chain forms a complete subset of a set of lggs of some AUP, then idempotent generalization would be nullary rather than infinitary. The statement of Lemma 3 address this issue by considering the production of lggs from existing lggs. Under the conditions enforced by our grammar construction, neither of the above mentioned cases is possible, and thus, an infinite chain of strictly less generality cannot be produced. Note that it still does not imply non-nullarity, because it is just a refutation of a sufficient condition for it. But this property is used in the proof that a minimal complete set of I-generalizations always exits.

One additional note concerns the proof argument itself, which contains a not entirely trivial quadruple induction. The tree grammar structure provides easy access to the sub-AUPs generated during grammar construction. It is precisely these sub-AUPs which are the focus of our argument because changing the internal structure of a generalization may allow comparison with other generalizations. Our goal is to show that, by construction, sub-grammars of the given tree grammar do not allow infinite chains of less generality. In order to complete this argument we perform induction on the derivation length (within the sub-grammar), internal node complexity, leaf node complexity, and number of duplication rule applications.

Lemma 3 *Let $s \triangleq t$ be an AUP over a signature Σ which may contain idempotent function symbols. Assume that there exists $n \in \mathbb{N}$ such that for all $1 \leq k \leq n$, the sets $L_k(G(s,t))$ are minimal. Let $w \in L_{n+1}(G(s,t)) \setminus L_n(G(s,t))$, $r \in L_n(G(s,t))$, and p be a position in r such that $w = r\,[f(t_1,t_2)]_p$ for $t_1, t_2 \in L_n(G(s,t))$. Then neither $t_1 \prec_I f(t_1,t_2)$ nor $t_2 \prec_I f(t_1,t_2)$.*

Proof.

Without loss of generality, we can focus on proving $t_1 \not\prec_I f(t_1,t_2)$ and assume that f is idempotent. The statement is trivial when f is free. We consider two cases:

Case 1 Assume that for some position p of r, we have $r|_p = t_1$ and $r \in L_0(G(s,t))$. This case implies that t_1 is constructed without the use of duplication rules of $R \setminus R_b$. Now let us consider $x \to \hat{t}[\bar{x}] \in R_b$, where $\hat{t}[\bar{x}]$ indicates that $\bar{x}$ is a list of the non-terminals occurring in $\hat{t}$, such that $t_1 = \hat{t}[\bar{x}]$. In particular, let us consider $\bar{x}$ to be empty. This means that I-PreGen did not use any of the Id-Both rules when constructing t_1. Furthermore, as a final assumption, t_1 does not contain idempotent function symbols. Under these constraints t_1 is a syntactic part of the generalization and thus $t_1 \not\prec_I f(t_1,t_2)$ ($\mathbf{BC}_1$).

Now assume that $t_1 \not\prec_I f(t_1,t_2)$ for t_1 containing at most m idempotent function symbols. We refer to this induction hypothesis as ($\mathbf{IH}_2$). We show that $t_1 \not\prec_I f(t_1,t_2)$ when t_1 contains at most $m+1$ idempotent function symbols (induction step ($\mathbf{IS}_2$)).

There are only two interesting cases to consider, when $t_1 = f(t_1',t_2')$ and when $t_1 = g(t_1',t_2')$ where g is idempotent and $f \neq g$. The statement $t_1 \not\prec_I f(t_1,t_2)$ trivially holds when $head(t_1)$ is a free symbol. First, consider the case $t_1 = f(t_1',t_2')$. Let us now assume that there exists a substitution σ such that $t_1\sigma \approx_I f(t_1,t_2)$, $t_1'\sigma \approx_I f(t_1,t_2)$ and $t_2'\sigma \approx_I f(t_1,t_2)$. Notice that the existence of such a σ allows us to reformulate the statement we are proving as

$t'_1 \not\approx_I f(t_1, t_2)$. Given that t'_1 has m idempotent symbols, by $\mathbf{IH}_2$, $t'_1 \not\approx_I f(t_1, t_2)$ holds. If, instead, there exists σ such that $t_1\sigma \approx_I f(t_1, t_2)$ and $t'_1\sigma \approx_I t_1$ and $t'_2\sigma \approx_I t_2$, then we reformulate the statement we are proving as $t'_1 \not\approx_I f(t'_1, t'_2)$. Again, t'_1 has fewer idempotent symbols than $m+1$ and by $\mathbf{IH}_2$, we get $t'_1 \not\approx_I f(t'_1, t'_2)$. Hence, we are done with the case $t_1 = f(t'_1, t'_2)$. Now assume $t_1 = g(t'_1, t'_2)$, where g is idempotent (different from f). Then there exists a substitution σ such that $t_1\sigma \approx_I f(t_1, t_2)$, $t'_1\sigma \approx_I f(t_1, t_2)$ and $t'_2\sigma \approx_I f(t_1, t_2)$. This is equivalent to the case when we reformulate the statement we are proving as $t'_1 \not\approx_I f(t_1, t_2)$. This proves the case when the list of non-terminals in $x \to \hat{t}[\bar{x}]$ is empty and, thus, provides a second base case ($\mathbf{BC}_2$).

Now let us assume that the $t_1 \not\approx_I f(t_1, t_2)$ holds when the longest derivation in the tree grammar starting from $x \to \hat{t}[\bar{x}]$ is of length at most k. We refer to this induction hypothesis as ($\mathbf{IH}_3$). We show that the statement holds if the longest derivation has length at most $k+1$. Now once again we have to consider the number of idempotent symbols, but this time the number of non-terminals occurring in $\hat{t}[\bar{x}]$ is non-zero.

Once again, let us consider the case when $\hat{t}[\bar{x}]$ does not contain idempotent function symbols. This implies that its construction is syntactic and we can go directly to the non-terminals $\bar{x}$ and look at the rules containing them on the left hand side. Let $x_i \to \hat{r}_i[\bar{z}]$, where $x_i \in \bar{x}$, be such a rule. For each rule of this form we can add a new rule to the grammar $x \to \hat{t}(x_1, \ldots, \hat{r}[\bar{z}], \ldots, x_w)$ where $\hat{r}[\bar{z}]$ replaces the proper non-terminal. Once this is done for all rules $x_i \to \hat{r}_i[\bar{z}]$, we remove $x \to \hat{t}[\bar{x}]$ from the grammar. This reduces the path length but does not change the language of the grammar. Thus, by ($\mathbf{IH}_3$), the statement holds in this case and provides us with ($\mathbf{BC}_3$).

We now assume $t_1 \not\approx_I f(t_1, t_2)$ when the number of idempotent symbols occurring in $\hat{t}[\bar{x}]$ is at most m ($\mathbf{IH}_4$) and show that it holds when the number of idempotent symbols occurring in $\hat{t}[\bar{x}]$ is at most $m+1$. This is exactly the same as the proof of ($\mathbf{BC}_2$). This ends the proof of Case 1.

Case 2 For some $0 < m < n+1$ there exists p such that $r|_p = t_1$ and $r \in L_m(G(s,t))$. Let us assume that the grammar rules used to construct t_1 are of the form $x \to g(x,x)$ where g is an idempotent function symbol, or are of the form $x \to \hat{t}[\bar{x}]$ where $\bar{x}$ is an empty set of non-terminals. The case when t_1 is constructed from a single rule of the form $\hat{t}[\bar{x}]$ is equivalent to the construction without duplication rules, which is considered in Case 1 above. Thus, we assume that at least two rules of the form $x \to \hat{t}[\bar{x}]$ are used for the construction of t_1. Furthermore, we assume that none of the $\hat{t}[\bar{x}]$ terms contain idempotent function symbols ($\mathbf{BC}_4$).

($\mathbf{BC}_4$) By our assumptions so far, if p' is a position of t_1 such that $t_1|_{p'} = h(s_1, \ldots, s_k)$ for $k \geq 1$, where at least one s_i is a variable, then h must be a non-idempotent function symbol. If h were idempotent, then $s_i \preceq s_{((i+1) \mod 2)}$ and would have been removed during minimization of $G_b(s,t)$. Note, this implies that for any subterm t' of t_1, if an immediate subterm of t' is a variable, then $head(t')$ cannot be idempotent. Thus, the argument from ($\mathbf{BC}_2$) concerning non-idempotent symbols can be applied.

Let us assume that $t_1 \not\approx_I f(t_1, t_2)$ holds when the rules of the form $x \to \hat{t}[\bar{x}]$ used to construct t_1 contain at most q idempotent function symbols ($\mathbf{IH}_5$). We show that the statement also holds when $x \to \hat{t}[\bar{x}]$ contains at most $q+1$ idempotent function symbols ($\mathbf{IS}_5$). We prove ($\mathbf{IS}_5$) using the same techniques as used to prove ($\mathbf{IS}_2$), except we need to add an additional distinction which differentiates between idempotent function symbols which occur as part of the rules $x \to \hat{t}[\bar{x}]$ and those which occur as part of $R \setminus R_b$, that is idempotent function symbols introduced by a duplication rule. Note that $q+1$ concerns the former idempotent symbols addressed by ($\mathbf{BC}_4$) and ($\mathbf{IH}_5$). This requires the same argument used for ($\mathbf{BC}_2$).

Now we have to repeat the argument used in Case 1 concerning the longest derivation in the tree grammar ignoring the duplication rules of $R \setminus R_b$. We are referring to the argument proving ($\mathbf{BC}_3$), and which also completes the proof of case 2.

Thus we have shown $t_1 \not\prec_I f(t_1, t_2)$.

□

Now that we have shown that the particular type of infinite chain discussed above cannot occur, we can address the minimality of each language $L_n(G(s,t))$, that is Theorem 8. These properties establish the following relationship between languages: any generalization which is in $L_{n+1}(G(s,t))$ but not in $L_n(G(s,t))$ must be incomparable to all generalizations of $L_n(G(s,t))$. Then the minimality result can easily be extended to the language $L(G(s,t))$ by taking the infinite union of the indexed languages (Theorem 9). It shows that idempotent generalization is not nullary, implying that it is infinitary.

Theorem 8 *Let $s \triangleq t$ be an AUT over a signature Σ which may contain idempotent function symbols and $G(s,t)$ be the grammar computed by* I-Gen*. Then for all $n \geq 0$, $L_n(G(s,t))$ is minimal.*

Proof.

We consider the minimality of $L_0(G(s,t))$ as a base case ($\mathbf{BC}_1$). Let us assume as an induction hypothesis that the theorem holds for $L_n(G(s,t))$ and show that it holds for $L_{n+1}(G(s,t))$. We will refer to this induction hypothesis as ($\mathbf{IH}_1$) and the induction step as ($\mathbf{IS}_1$).

Since $L_n(G(s,t))$ is minimal by ($\mathbf{IH}_1$), we can split the proof of minimality of $L_{n+1}(G(s,t))$ into three cases. These cases, proving ($\mathbf{IS}_1$), are denoted by the following three propositions:

P1 For all $s_1 \in L_{n+1}(G(s,t)) \setminus L_n(G(s,t))$ and $s_2 \in L_n(G(s,t))$, it is not the case that $s_2 \preceq_I s_1$.

P2 For all $s_1 \in L_{n+1}(G(s,t)) \setminus L_n(G(s,t))$ and $s_2 \in L_n(G(s,t))$, it is not the case that $s_1 \preceq_I s_2$.

P3 For all $s_1, s_2 \in L_{n+1}(G(s,t)) \setminus L_n(G(s,t))$, if $s_1 \neq s_2$, then $s_1 \not\preceq_I s_2$.

Proving P1 In order to prove **P1** we perform a second induction over the minimum $k \geq 0$ such that $s_2 \in L_k(G(s,t))$. The following assertions play a fundamental role in the majority of the cases outlined below:

(a) There exist positions p in s_1 and q in s_2 such that $s_1|_p = f(t_1, t_2)$ and $s_2|_q\sigma = f(t_1, t_2)$, where $t_1, t_2 \in L_n(G(s,t))$, and at least one of t_1 and t_2 is in the language $L_n(G(s,t)) \setminus L_{n-1}(G(s,t))$.

The existence of such positions is implied by the construction of s_1 from a term $t \in L_n(G(s,t)) \setminus L_{n-1}(G(s,t))$ by replacing the subterm at position p in t by $f(t_1, t_2)$.

(b) There exist positions l_1 in s and l_2 in t such that $f(t_1, t_2)$, t_1, and t_2 are generalizations of $s|_{l_1} \triangleq t|_{l_2}$. This is a consequence of grammar construction and the definition of our duplication rules.

Case: ($\mathbf{BC}_2$) We assume that $k = 0$ and, thus, $s_2 \in L_0(G(s,t))$. Even under this restriction we need to perform a third induction on the structure of $s_2|_q$.

Case: ($\mathbf{BC}_3$) We assume $s_2|_q = x$ where x is a variable. A consequence of this assumption is $s_2\{x \mapsto f(t_1, t_2)\} = s_1$. By assertion (a), $s_1 = t[f(t_1, t_2)]_p$ and $t[t_1]_p, t[t_2]_p \in L_n(G(s,t))$. This implies that either $s_2\{x \mapsto t_1\} = t[t_1]_p$ or $s_2\{x \mapsto t_2\} = t[t_2]_p$, contradicting the minimality of $L_n(G(s,t))$ assumed by ($\mathbf{IH}_1$). Therefore, $s_2|_q$ can be neither a variable nor a term of depth 1.

Case: ($\mathbf{IH}_3$) We assume that $s_2|_q$ cannot be a term of depth n and show that it cannot be a term of depth $n+1$ ($\mathbf{IS}_3$). By construction, $s_2|_q \neq g(w_1, \ldots, w_s)$ for a free function symbol g. Therefore, we only consider the following cases in order to prove ($\mathbf{IS}_3$):

Case 1 ($\mathbf{IS}_3$) Let $s_2|_q = f(r_1, r_2)$. This implies that $r_1\sigma = f(t_1, t_2)$ and $r_2\sigma = f(t_1, t_2)$, or, alternatively, $r_1\sigma = t_1$ and $r_2\sigma = t_2$. In the former case $s_2|_q$ can be replaced by either r_1 or r_2, but it would contradict ($\mathbf{IH}_3$). In the latter case, r_1 and r_2 must also be solutions to $s|_{l_1} \triangleq t|_{l_2}$, but this would imply that $L_n(G(s|_{l_1}, t|_{l_2}))$ is not minimal, which contradicts ($\mathbf{IH}_1$).

Case 2 ($\mathbf{IS}_3$) Let $s_2|_q = g(r_1, r_2)$ for $g \neq f$. This implies that $r_1\sigma = f(t_1, t_2)$ and $r_2\sigma = f(t_1, t_2)$. In this case $s_2|_q$ can be replaced by either r_1 or r_2 and, by ($\mathbf{IH}_3$), the statement holds.

Case: ($\mathbf{IH}_2$) We assume for $s_2 \in L_k(G(s,t)) \setminus L_{k-1}(G(s,t))$ with $0 < k < n$ that no position in s_2 can result in $f(t_1, t_2)$ after applying σ to s_2. We show that the same holds for $k = n$ ($\mathbf{IS}_2$). It must be the case that $s_2|_q = f(r_1, r_2)$ for an idempotent function symbol f and $r_1, r_2 \in L_{n-1}(G(s,t))$, such that at least one of r_1, r_2 is in $L_n(G(s,t)) \setminus L_{n-1}(G(s,t))$. If the latter condition is not fulfilled, the case is covered by ($\mathbf{IH}_2$)). Thus, as before, we get $r_1\sigma = f(t_1, t_2)$ and $r_2\sigma = f(t_1, t_2)$, or, alternatively, $r_1\sigma = t_1$ and $r_2\sigma = t_2$. The former is an instance of ($\mathbf{IH}_3$) Which can be handled in a similar fashion as ($\mathbf{IS}_3$) and the latter contradicts ($\mathbf{IH}_1$). Thus, we have proven ($\mathbf{IS}_2$).

The only case we have not considered is $t_1 \prec_I f(t_1, t_2)$ or $t_2 \prec_I f(t_1, t_2)$. However, this is not possible by Lemma 3.

Proving P2 Let us consider a position p in s_1 such that $s_1|_p = f(t_1, t_2)$, where f is idempotent and either $dra(t_1) = n$ and/or $dra(t_2) = n$. Without loss of generality, we assume $dra(t_1) = n$. Furthermore, let σ be the substitution such that $s_1\sigma = s_2$. Let us now consider positions l_1 in s and l_2 in t such that $f(t_1, t_2)$, t_1, and t_2 are generalizations of $s|_{l_1} \triangleq t|_{l_2}$. We now consider the sub-grammar of $G(s,t)$, namely $G(s|_{l_1}, t|_{l_2})$. If $t_1\sigma$ and/or $t_2\sigma$ are generalizations of $s|_{l_1} \triangleq t|_{l_2}$, then we contradict the minimality of $L_n(G(s|_{l_1}, t|_{l_2}))$. Otherwise, if neither is a generalization of $s|_{l_1} \triangleq t|_{l_2}$, then $f(t_1\sigma, t_2\sigma)$ cannot be a generalization of the AUP implied by the subgrammar by Lemma 2.

Proving P3 Let $s_1 = f(r_1, r_2)$ and assume by contradiction that $s_1 \preceq_I s_2$. Let σ be a substitution such that $f(r_1\sigma, r_2\sigma) \approx_I s_2$. We proceed by case distinction on s_2.

Case 1 s_2 has the form $f(t_1, t_2)$. Then we have two subcases:

- Subcase 1. $r_i\sigma \approx_I t_i$, $i = 1, 2$. But it contradicts the minimality of $L_n(G(s,t))$, because $r_1, r_2, t_1, t_2 \in L_n(G(s,t))$.
- Subcase 2. $r_1\sigma = r_2\sigma \approx_I f(t_1, t_2) = s_2$. But since $r_1, r_2 \in L_n(G(s,t))$, $s_2 \in L_{n+1}(G(s,t)) \setminus L_n(G(s,t))$, we get a contradiction with the already proved **P1**.

Case 2 s_2 has the form $g(t_1, t_2)$, $g \neq f$. The only possibility is $r_1\sigma = r_2\sigma \approx_I g(t_1, t_2)$, and the argument is like in Subcase 2 above.

It proves **P3**.

□

Theorem 9 *$L(G(s,t))$ is a minimal complete set of I-generalizations of s and t.*

Proof.

From Theorem 8 and the fact that $L_n(G(s,t)) \subseteq L_{n+1}(G(s,t))$ for all $n \geq 0$, we get that $\bigcup_{n=0}^{\infty}(L_n(G(s,t))$ is minimal. But the latter set is exactly $L(G(s,t))$. (If we assume that $L(G(s,t))$ is I-normalized, then $L(G(s,t)) = nf_I(\bigcup_{n=0}^{\infty}(L_n(G(s,t))))$, but $nf_I(\bigcup_{n=0}^{\infty}(L_n(G(s,t))))$ is minimal iff $\bigcup_{n=0}^{\infty}(L_n(G(s,t)))$ is minimal). Completeness of $L(G(s,t))$ was shown in Theorem 6.

□

Corollary 1 *Idempotent generalization is infinitary.*

Proof.

Theorem 9 says that the minimal complete set of idempotent generalizations of two terms always exists. The example in Section 3 shows that it is infinitary.

□

6 Conclusion

The main contributions of this paper are twofold: first, we show that anti-unification with a single idempotent function symbol is at least infinitary. It was known from [33] that the same holds for anti-unification with two idempotent symbols. Therefore, our result has a practical implication: if it were not true (i.e., if anti-unification with a single idempotent symbol were unitary or finitary), one could not hope to get a general combination method for anti-unification in disjoint theories, because finitary algorithms cannot be combined into an infinitary procedure. Now, when this potential obstacle is removed (at least for the idempotent case), it makes sense to look into the details of the combination of anti-unification algorithms, and it is a part of our future work.

The second contribution is the algorithm for solving idempotent anti-unification problems. Although the solution sets of those problems might be infinite, it turns out that they form regular tree languages, and our algorithm computes their finite representation in the form of regular tree grammars. It should be remarked that the languages generated by those grammars are not only complete, but also minimal sets of generalizations. The algorithm does not depend on the number of different idempotent symbols in the theory. This result implies that idempotent anti-unification is infinitary for any number of idempotent function symbols.

References

[1] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014.

[2] Franz Baader. The theory of idempotent semigroups is of unification type zero. *J. Autom. Reasoning*, 2(3):283–286, 1986.

[3] Franz Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1991.

[4] Franz Baader and Wolfram Büttner. Unification in commutative idempotent monoids. *Theor. Comput. Sci.*, 56:345–353, 1988.

[5] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. Symb. Comput.*, 21(2):211–243, 1996.

[6] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.

[7] 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`.

[8] Alexander Baumgartner and Temur Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017.

[9] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. A variant of higher-order anti-unification. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[10] 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.

[11] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reasoning*, 58(2):293–310, 2017.

[12] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Term-graph anti-unification. In Kirchner [23], pages 9:1–9:17.

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

[14] Jochen Burghardt. E-generalization using grammars. *Artif. Intell.*, 165(1):1–35, 2005.

[15] Jochen Burghardt and Birgit Heinz. Implementing anti-unification modulo equational theory. *CoRR*, abs/1404.0953, 2014.

[16] David M. Cerna and Temur Kutsia. Higher-order equational pattern anti-unification. In Kirchner [23], pages 12:1–12:17.

[17] Sebastian Eberhard and Stefan Hetzl. Inductive theorem proving based on tree grammars. *Ann. Pure Appl. Logic*, 166(6):665–700, 2015.

[18] Birgit Heinz. *Anti-Unifikation modulo Gleichungstheorie und deren Anwendung zur Lemmagenerierung*. PhD thesis, TU Berlin, 1995.

[19] Alexander Herold. Narrowing techniques applied to idempotent unification. In Katharina Morik, editor, *GWAI-87, 11th German Workshop on Artificial Intelligence, Geseke, Germany, September 28 - October 2, 1987, Proceedings*, volume 152 of *Informatik-Fachberichte*, pages 231–240. Springer, 1987.

[20] Stefan Hetzl, Alexander Leitsch, Giselle Reis, Janos Tapolczai, and Daniel Weller. Introducing quantified cuts in logic with equality. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2014.

[21] Stefan Hetzl, Alexander Leitsch, and Daniel Weller. Towards algorithmic cut-introduction. In Nikolaj Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2012.

[22] Jean-Marie Hullot. Canonical forms and unification. In Wolfgang Bibel and Robert A. Kowalski, editors, *5th Conference on Automated Deduction, Les Arcs, France, July 8-11, 1980, Proceedings*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer, 1980.

[23] Hélène Kirchner, editor. *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[24] Boris Konev and Temur Kutsia. Anti-unification of concepts in description logic EL. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 227–236. AAAI Press, 2016.

[25] 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.

[26] Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014.

[27] Alexander Leitsch, Nicolas Peltier, and Daniel Weller. CERES for first-order schemata. *J. Log. Comput.*, 27(7):1897–1954, 2017.

[28] Mike Livesey and Jörg Siekmann. Unification of sets and multisets. Internal report MEMO SEKI-76-II, Universität Karlsruhe, 1976.

[29] Mike Livesey, Jörg Siekmann, Peter Szabó, and Eva Unvericht. Unification problems for combinations of associativity, commutativity, distributivity and idempotence axioms. In William Joyner, editor, *Proceedings of the 4th Workshop on Automated Deduction, CADE-4*, University of Texas at Austin, USA, 1979.

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

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

[32] Gordon D. Plotkin. Building in equational theories. *Machine Intell.*, 7:7390, 1972.

[33] Loïc Pottier. Generalisation de termes en theorie equationnelle. Cas associatif-commutatif. Rapports de Recherche 1056, INRIA Sophia Antipolis, 1989.

[34] Peter Raulefs and Jörg Siekmann. Unification of idempotent functions. MEMO SEKI-78-II-KL, Universität Karlsruhe, 1978.

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

[36] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[37] Manfred Schmidt-Schauß. Unification properties of idempotent semigroups. SEKI Report SR-86-07, Universität Kaiserslautern, 1986.

[38] Manfred Schmidt-Schauß. Unification under associativity and idempotence is of type nullary. *J. Autom. Reasoning*, 2(3):277–281, 1986.

[39] Jörg Siekmann. String-unification. Internal Report Memo CSM-7, Essex University, 1975.

[40] Jörg Siekmann. *Unification and Matching Problems.* Memo csa-4-78, Essex University, 1978.

[41] Jörg H. Siekmann. Unification theory. *J. Symb. Comput.*, 7(3/4):207–274, 1989.

[42] Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.

[43] Akihiro Yamamoto, Kimihito Ito, Akira Ishino, and Hiroki Arimura. Modelling semi-structured documents with hedges for deduction and induction. In Céline Rouveirol and Michèle Sebag, editors, *ILP*, volume 2157 of *Lecture Notes in Computer Science*, pages 240–247. Springer, 2001.