A framework for approximate generalization in quantitative theories

Temur Kutsia, Cleo Pau

May 2022

RISC Report Series No. 22-04

ISSN: 2791-4267 (online)

Available at https://doi.org/10.35011/risc.22-04

A framework for approximate generalization in quantitative theories

Temur Kutsia and Cleo Pau

RISC, Johannes Kepler University Linz, Austria
{kutsia,ipau}@risc.jku.at

Abstract. Anti-unification aims at computing generalizations for given terms, retaining their common structure and abstracting differences by variables. We study quantitative anti-unification where the notion of the common structure is relaxed into "proximal" up to the given degree with respect to the given fuzzy proximity relation. Proximal symbols may have different names and arities. We develop a generic set of rules for computing minimal complete sets of approximate generalizations and study their properties. Depending on the characterizations of proximities between symbols and the desired forms of solutions, these rules give rise to different versions of concrete algorithms.

Keywords: Generalization · Anti-unification · Quantiative theories · Fuzzy proximity relations.

1 Introduction

Generalization problems play an important role in various areas of mathematics, computer science, and artificial intelligence. Anti-unification [12,14] is a logic-based method for computing generalizations. Being originally used for inductive and analogical reasoning, some recent applications include recursion scheme detection in functional programs [4], programming by examples in domain-specific languages [13], learning bug-fixing from software code repositories [15,3], automatic program repair [7], preventing bugs and misconfiguration in services [10], linguistic structure learning for chatbots [6], to name just a few.

In most of the existing theories where anti-unification is studied, the background knowledge is assumed to be precise. Therefore, those techniques are not suitable for reasoning with incomplete, imprecise information (which is very common in real-world communication), where the exact equality is replaced by its (quantitative) approximation. Fuzzy proximity and similarity relations are notable examples of such extensions. These kinds of quantitative theories have many useful applications, some most recent ones being related to artificial intelligence, program verification, probabilistic programming, or natural language processing. Many tasks arising in these areas require reasoning methods and computational tools that deal with quantitative information. For instance, approximate inductive reasoning, reasoning and programming by analogy, similarity detection in programming language statements or in natural language texts could benefit from solving approximate generalization constraints, which is a theoretically interesting and challenging task. Investigations in this direction have been started only recently. In [1], the authors proposed an anti-unification algorithm for fuzzy similarity (reflexive, symmetric, min-transitive) relations, where mismatches are allowed not only in symbol names, but also in their arities (fully fuzzy signatures). The algorithm from [8] is designed for fuzzy proximity (i.e., reflexive and symmetric) relations with mismatches only in symbol names.

In this paper, we study approximate anti-unification from a more general perspective. The considered relations are fuzzy proximity relations. Proximal symbols may have different names and arities. We consider four different variants of relating arguments between different proximal symbols: unrestricted relations / functions, and correspondence (i.e. left- and right-total) relations / functions. A generic set of rules for computing minimal complete sets of generalizations is introduced and its termination, soundness and completeness properties are proved. From these rules, we obtain concrete algorithms that deal with different kinds of argument relations. We also show how the existing approximate anti-unification algorithms and their generalizations fit into this framework.

Related works that concern unification in fully fuzzy signatures have been published in [1] (for similarity) and [11] (for proximity).

Organization: In Sect. 2 we introduce the notation and definitions. Sect. 3 is devoted to a technical notion of term set consistency and to an algorithm for computing elements of consistent sets of terms. It is used later in the main set of anti-unification rules, which are introduced and characterized in Sect. 4. The concrete algorithms obtained from those rules are also described in this section. In Sect. 5, we discuss complexity. In Sect. 6, and extended example is given. Sect. 7 offers a high-level picture of the studied problems and concludes.

2 Preliminaries

Proximity relations. Given a set S, a mapping $\mathcal{R}$ from $S \times S$ to the real interval $[0, 1]$ is called a binary *fuzzy relation* on S. By fixing a number λ, $0 \leqslant \lambda \leqslant 1$, we can define the crisp (i.e., two-valued) counterpart of $\mathcal{R}$, named the λ-*cut* of $\mathcal{R}$, as $\mathcal{R}_\lambda := \{(s_1, s_2) \mid \mathcal{R}(s_1, s_2) \geqslant \lambda\}$. A fuzzy relation $\mathcal{R}$ on a set S is called a *proximity relation* if it is reflexive ($\mathcal{R}(s, s) = 1$ for all $s \in S$) and symmetric ($\mathcal{R}(s_1, s_2) = \mathcal{R}(s_2, s_1)$ for all $s_1, s_2 \in S$).

A T-norm $\wedge$ is an associative, commutative, non-decreasing binary operation on $[0, 1]$ with 1 as the unit element. We take minimum in the role of T-norm.

Terms and substitutions. We consider a first-order alphabet consisting of a set of fixed arity function symbols $\mathcal{F}$ and a set of variables $\mathcal{V}$, which includes a special symbol $\_$ (the anonymous variable). The set of *named* (i.e., non-anonymous) variables $\mathcal{V} \backslash \{\_\}$ is denoted by $\mathcal{V}^{\mathrm{N}}$. When the set of variables is not explicitly specified, we mean $\mathcal{V}$. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ over $\mathcal{F}$ and $\mathcal{V}$ is defined in the standard way: $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ iff t is defined by the grammar $t := x \mid$

$f(t_1,\dots,t_n)$, where $x \in \mathcal{V}$ and $f \in \mathcal{F}$ is an n-ary symbol with $n \geqslant 0$. Terms over $\mathcal{T}(\mathcal{F},\mathcal{V}^{\mathrm{N}})$ are defined similarly except that all variables are taken from $\mathcal{V}^{\mathrm{N}}$.

We denote arbitrary function symbols by f, g, h, constants by a, b, c, variables by x, y, z, v, and terms by s, t, r. The *head* of a term is defined as $\mathsf{head}(x) := x$ and $\mathsf{head}(f(t_1,\dots,t_n)) := f$. For a term t, we denote with $\mathcal{V}(t)$ (resp. by $\mathcal{V}^{\mathrm{N}}(t)$) the set of all variables (resp. all named variables) appearing in t. A term is called *linear* if no named variable occurs in it more than once.

The deanonymization operation deanon replaces each occurrence of the anonymous variable in a term by a fresh variable. For instance, $\mathsf{deanon}(f(\_, x, g(\_))) = f(y', x, g(y'')))$, where y' and y'' are fresh. Hence, $\mathsf{deanon}(t) \in \mathcal{T}(\mathcal{F},\mathcal{V}^{\mathrm{N}})$ is unique up to variable renaming for all $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$. If t is linear, then $\mathsf{deanon}(t)$ is linear as well and vice versa.

The notions of *term depth*, *term size* and a *position in a term* are defined in the standard way, see, e.g. [2]. By $t|_p$ we denote the subterm of t at position p and by $t[s]_p$ a term that is obtained from t by replacing the subterm at position p by the term s.

A *substitution* is a mapping from $\mathcal{V}^{\mathrm{N}}$ to $\mathcal{T}(\mathcal{F},\mathcal{V}^{\mathrm{N}})$ (i.e., without anonymous variables), which is the identity almost everywhere. We use the Greek letters $\sigma, \vartheta, \varphi$ to denote substitutions, except for the identity substitution which is written as Id. We represent substitutions with the usual set notation. *Application* of a substitution σ to a term t, denoted by $t\sigma$, is defined as $\_\sigma := \_$, $x\sigma := \sigma(x)$, $f(t_1,\dots,t_n)\sigma := f(t_1\sigma,\dots,t_n\sigma)$. Substitution *composition* is defined as a composition of mappings. We write $\sigma\vartheta$ for the composition of σ with ϑ.

Argument relations and mappings. Given two sets $N = \{1,\dots,n\}$ and $M = \{1,\dots,m\}$, a binary *argument relation* over $N \times M$ is a (possibly empty) subset of $N \times M$. We denote argument relations by ρ. An argument relation $\rho \subseteq N \times M$ is (i) *left-total* if for all $i \in N$ there exists $j \in M$ such that $(i,j) \in \rho$; (ii) *right-total* if for all $j \in M$ there exists $i \in N$ such that $(i,j) \in \rho$. *Correspondence relations* are those that are both left- and right-total.

An *argument mapping* is an argument relation that is a partial injective function. In other words, an argument mapping π from $N = \{1,\dots,n\}$ to $M = \{1,\dots,m\}$ is a function $\pi : I_n \mapsto I_m$, where $I_n \subseteq N$, $I_m \subseteq M$ and $|I_n| = |I_m|$. Note that it can be also the empty mapping: $\pi : \emptyset \mapsto \emptyset$. The inverse of an argument mapping is again an argument mapping.

Given a proximity relation $\mathcal{R}$ over $\mathcal{F}$, we assume that for each pair of function symbols f and g with $\mathcal{R}(f,g) = \alpha > 0$, where f is n-ary and g is m-ary, there is also given an argument relation ρ over $\{1,\dots,n\} \times \{1,\dots,m\}$. We use the notation $f \sim^{\rho}_{\mathcal{R},\alpha} g$. These argument relations should satisfy the following conditions: ρ is the empty relation if f or g is a constant; ρ is the identity if $f = g$; $f \sim^{\rho}_{\mathcal{R},\alpha} g$ iff $g \sim^{\rho^{-1}}_{\mathcal{R},\alpha} f$, where ρ^{-1} is the inverse of ρ.

Example 1. Assume that we have four different versions of defining the notion of author (e.g., originated from four different knowledge bases) $author_1$(*first*-name, *middle-initial*, *last*-name), $author_2$(*first*-name, *last*-name), $author_3$(*last*-name, *first*-name, *middle-initial*), and $author_4$(*full*-name). One could define the argument relations/mappings between these function symbols e.g., as follows:

$$
\begin{aligned}
&author_1 \sim^{\{(1,1),(3,2)\}}_{\mathcal{R},0.7} author_2, \quad author_1 \sim^{\{(3,1),(1,2),(2,3)\}}_{\mathcal{R},0.9} author_3,\\
&author_1 \sim^{\{(1,1),(3,1)\}}_{\mathcal{R},0.5} author_4, \quad author_2 \sim^{\{(1,2),(2,1)\}}_{\mathcal{R},0.7} author_3,\\
&author_2 \sim^{\{(1,1),(2,1)\}}_{\mathcal{R},0.5} author_4, \quad author_3 \sim^{\{(1,1),(2,1)\}}_{\mathcal{R},0.5} author_4.
\end{aligned}
$$

Proximity relations over terms. Each proximity relation $\mathcal{R}$ in this paper is defined on $\mathcal{F} \cup \mathcal{V}$ such that $\mathcal{R}(f, x) = 0$ for all $f \in \mathcal{F}$ and $x \in \mathcal{V}$, and $\mathcal{R}(x, y) = 0$ for all $x \neq y$, $x, y \in \mathcal{V}$. We assume that $\mathcal{R}$ is *strict*: for all $w_1, w_2 \in \mathcal{F} \cup \mathcal{V}$, if $\mathcal{R}(w_1, w_2) = 1$, then $w_1 = w_2$. Yet another assumption is that for each $f \in \mathcal{F}$, its $(\mathcal{R}, \lambda)$-proximity class $\{g \mid \mathcal{R}(f, g) \geqslant \lambda\}$ is *finite* for any $\mathcal{R}$ and λ.

We extend such an $\mathcal{R}$ to terms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ as follows:

(a) $\mathcal{R}(t, s) := 0$ if $\mathcal{R}(\mathsf{head}(s), \mathsf{head}(t)) = 0$;
(b) $\mathcal{R}(t, s) := 1$ if $t = s$ and $t, s \in \mathcal{V}$;
(c) $\mathcal{R}(t, s) := \mathcal{R}(f, g) \wedge \mathcal{R}(t_{i_1}, s_{j_1}) \wedge \cdots \wedge \mathcal{R}(t_{i_k}, s_{j_k})$, if $t = f(t_1, \ldots, t_n)$, $s = g(s_1, \ldots, s_m)$, $f \sim^{\rho}_{\mathcal{R},\lambda} g$, and $\rho = \{(i_1, j_1), \ldots, (i_k, j_k)\}$.

If $\mathcal{R}(t, s) \geqslant \lambda$, we write $t \simeq_{\mathcal{R},\lambda} s$. When $\lambda = 1$, the relation $\simeq_{\mathcal{R},\lambda}$ does not depend on $\mathcal{R}$ due to strictness of the latter and is just the syntactic equality $=$.

The $(\mathcal{R}, \lambda)$-*proximity class* of a term t is $\mathbf{pc}_{\mathcal{R},\lambda}(t) := \{s \mid s \simeq_{\mathcal{R},\lambda} t\}$.

Generalizations. Given $\mathcal{R}$ and λ, a term r is an $(\mathcal{R}, \lambda)$-*generalization* of (alternatively, $(\mathcal{R}, \lambda)$-*more general than*) a term t, written as $r \precsim_{\mathcal{R},\lambda} t$, if there exists a substitution σ such that $\mathsf{deanon}(r)\sigma \simeq_{\mathcal{R},\lambda} \mathsf{deanon}(t)$. The strict part of $\precsim_{\mathcal{R},\lambda}$ is denoted by $\prec_{\mathcal{R},\lambda}$, i.e., $r \prec_{\mathcal{R},\lambda} t$ if $r \precsim_{\mathcal{R},\lambda} t$ and not $t \precsim_{\mathcal{R},\lambda} r$.

Example 2. Given a proximity relation $\mathcal{R}$, a cut value λ, constants $a \sim^{\varnothing}_{\mathcal{R},\alpha_1} b$ and $b \sim^{\varnothing}_{\mathcal{R},\alpha_2} c$, binary function symbols f and h, and a unary function symbol g such that $h \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},\alpha_3} f$ and $h \sim^{\{(1,1)\}}_{\mathcal{R},\alpha_4} g$ with $\alpha_i \geqslant \lambda$, $1 \leqslant i \leqslant 4$, we have

– $h(x, \_) \precsim_{\mathcal{R},\lambda} h(a, x)$, because $h(x, x')\{x \mapsto a, x' \mapsto x\} = h(a, x) \simeq_{\mathcal{R},\lambda} h(a, x)$.
– $h(x, \_) \precsim_{\mathcal{R},\lambda} h(\_, x)$, because $h(x, x')\{x \mapsto y', x' \mapsto x\} = h(y', x) \simeq_{\mathcal{R},\lambda} h(y', x)$.
– $h(x, x) \not\precsim_{\mathcal{R},\lambda} h(\_, x)$, because $h(x, x) \not\precsim_{\mathcal{R},\lambda} h(y', x)$.
– $h(x, \_) \precsim_{\mathcal{R},\lambda} f(a, c)$, because $h(x, x')\{x \mapsto b\} = h(b, x') \simeq_{\mathcal{R},\lambda} f(a, c)$.
– $h(x, \_) \precsim_{\mathcal{R},\lambda} g(c)$, because $h(x, x')\{x \mapsto c\} = h(c, x') \simeq_{\mathcal{R},\lambda} g(c)$.

The notion of *syntactic generalization* of a term is a special case of $(\mathcal{R}, \lambda)$-generalization for $\lambda = 1$. We write $r \precsim t$ to indicate that r is a syntactic generalization of t. Its strict part is denoted by $\prec$.

Since $\mathcal{R}$ is strict, $r \precsim t$ is equivalent to $\mathsf{deanon}(r)\sigma = \mathsf{deanon}(t)$ for some σ (note the syntactic equality here).

Theorem 1. *If $r \precsim t$ and $t \precsim_{\mathcal{R},\lambda} s$, then $r \precsim_{\mathcal{R},\lambda} s$.*

Proof. $r \precsim t$ implies $\mathsf{deanon}(r)\sigma = \mathsf{deanon}(t)$ for some σ, while from $t \precsim_{\mathcal{R},\lambda} s$ we have $\mathsf{deanon}(t)\vartheta \simeq_{\mathcal{R},\lambda} \mathsf{deanon}(s)$ for some ϑ. Then $\mathsf{deanon}(r)\sigma\vartheta \simeq_{\mathcal{R},\lambda} \mathsf{deanon}(s)$, which implies $r \precsim_{\mathcal{R},\lambda} s$. □

Note that $r \lesssim_{\mathcal{R},\lambda} t$ and $t \lesssim_{\mathcal{R},\lambda} s$, in general, do not imply $r \lesssim_{\mathcal{R},\lambda} s$ due to non-transitivity of $\simeq_{\mathcal{R},\lambda}$.

Definition 1 (Minimal complete set of $(\mathcal{R},\lambda)$-generalizations). *Given $\mathcal{R}$, λ, t_1, and t_2, a set of terms T is a* complete set of $(\mathcal{R},\lambda)$-generalizations *of t_1 and t_2 if*

(a) every $r \in T$ is an $(\mathcal{R},\lambda)$-generalization of t_1 and t_2,
(b) if r' is an $(\mathcal{R},\lambda)$-generalization of t_1 and t_2, then there exists $r \in T$ such that $r' \lesssim r$ (note that we use syntactic generalization here).

In addition, T is minimal, if it satisfies the following property:

(c) if $r, r' \in T$, $r \neq r'$, then neither $r \prec_{\mathcal{R},\lambda} r'$ nor $r' \prec_{\mathcal{R},\lambda} r$.

A minimal complete set of $(\mathcal{R},\lambda)$-generalizations *($(\mathcal{R},\lambda)$-mcsg) of two terms is unique modulo variable renaming. The elements of the $(\mathcal{R},\lambda)$-mcsg of t_1 and t_2 are called least general $(\mathcal{R},\lambda)$-generalizations ($(\mathcal{R},\lambda)$-lggs) of t_1 and t_2.*

This definition directly extends to generalizations of finitely many terms.

The problem of computing an $(\mathcal{R},\lambda)$-generalization of terms t and s is called the $(\mathcal{R},\lambda)$-*anti-unification problem* of t and s. In anti-unification, the goal is to compute their least general $(\mathcal{R},\lambda)$-generalization.

The precise formulation of the anti-unification problem would be the following: Given $\mathcal{R}$, λ, t_1, t_2, find an $(\mathcal{R},\lambda)$-lgg r of t_1 and t_2, substitutions σ_1, σ_2, and the approximation degrees α_1, α_2 such that $\mathcal{R}(r\sigma_1, t_1) = \alpha_1$ and $\mathcal{R}(r\sigma_2, t_2) = \alpha_2$. A minimal complete algorithm to solve this problem would compute exactly the elements of $(\mathcal{R},\lambda)$-mcsg of t_1 and t_2 together with their approximation degrees. However, as we see below, it is problematic to solve the problem in this form. Therefore, we will consider a slightly modified variant, taking into account anonymous variables in generalizations and relaxing bounds on their degrees.

We assume that the terms to be generalized are ground. It is not a restriction because we can treat variables as constants that are close only to themselves.

Recall that the proximity class of any alphabet symbol is finite. Also, the symbols are related to each other by finitely many argument relations. One may think that it leads to finite proximity classes of terms, but this is not the case. Consider, e.g., $\mathcal{R}$ and λ, where $h \simeq_{\mathcal{R},\lambda}^{\{(1,1)\}} f$ with binary h and unary f. Then the $(\mathcal{R},\lambda)$-proximity class of $f(a)$ is infinite: $\{f(a)\} \cup \{h(a,t) \mid t \in \mathcal{T}(\mathcal{F},\mathcal{V})\}$. Also, the $(\mathcal{R},\lambda)$-mcsg for $f(a)$ and $f(b)$ is infinite: $\{f(x)\} \cup \{h(x,t) \mid t \in \mathcal{T}(\mathcal{F},\varnothing)\}$.

Definition 2. *Given the terms $t_1,\dots,t_n$, $n \geqslant 1$, a position p in a term r is called* irrelevant for $(\mathcal{R},\lambda)$-generalizing *(resp.* for $(\mathcal{R},\lambda)$-proximity to*) $t_1,\dots,t_n$ if $r[s]_p \lesssim_{\mathcal{R},\lambda} t_i$ (resp. $r[s]_p \simeq_{\mathcal{R},\lambda} t_i$) for all $1 \leqslant i \leqslant n$ and for all terms s.*

We say that r is a relevant $(\mathcal{R},\lambda)$-generalization *(resp.* relevant $(\mathcal{R},\lambda)$-proximal term*) of $t_1,\dots,t_n$ if $r \lesssim_{\mathcal{R},\lambda} t_i$ (resp. $r \simeq_{\mathcal{R},\lambda} t_i$) for all $1 \leqslant i \leqslant n$ and $r|_p = \_$ for all positions p in r that is irrelevant for generalizing (resp. for proximity to) $t_1,\dots,t_n$. The* $(\mathcal{R},\lambda)$-relevant proximity class *of t is*

$$\mathbf{rpc}_{\mathcal{R},\lambda}(t) := \{s \mid s \text{ is a relevant } (\mathcal{R},\lambda)\text{-proximal term of } t\}.$$

In the example above, position 2 in $h(x,t)$ is irrelevant for generalizing $f(a)$ and $f(b)$, and $h(x,\_)$ is one of their relevant generalizations. Note that $f(x)$ is also a relevant generalization of $f(a)$ and $f(b)$, since it contains no irrelevant positions. More general generalizations like, e.g., x, are relevant as well. Similarly, position 2 in $h(a,t)$ is irrelevant for proximity to $f(a)$ and $\mathbf{rpc}_{\mathcal{R},\lambda}(f(a)) = \{f(a), h(a,\_)\}$. Generally, $\mathbf{rpc}_{\mathcal{R},\lambda}(t)$ is finite for any t due to the finiteness of proximity classes of symbols and argument relations mentioned above.

Definition 3 (Minimal complete set of relevant $(\mathcal{R},\lambda)$-generalizations). *Given $\mathcal{R}$, λ, t_1, and t_2, a set of terms T is a* complete set of relevant $(\mathcal{R},\lambda)$-generalizations *of t_1 and t_2 if*

(a) every element of T is a relevant $(\mathcal{R},\lambda)$-generalization of t_1 and t_2, and
(b) if r is a relevant $(\mathcal{R},\lambda)$-generalization of t_1 and t_2, then there exists $r' \in T$ such that $r \precsim r'$.

The minimality property is defined as in Definition 1.

This definition directly extends to relevant generalizations of finitely many terms. We use $(\mathcal{R},\lambda)$-mcsrg as an abbreviation for minimal complete set of relevant $(\mathcal{R},\lambda)$-generalization. Like relevant proximity classes, mcsrg's are also finite.

Lemma 1. *For given $\mathcal{R}$ and λ, if all argument relations are correspondence relations, then $(\mathcal{R},\lambda)$-mcsg's and $(\mathcal{R},\lambda)$-proximity classes for all terms are finite.*

Proof. Under correspondence relations no term contains an irrelevant position for generalization or for proximity. □

Hence, for correspondence relations the notions of mcsg and mcsrg coincide, as well as the notions of proximity class and relevant proximity class.

For a term r, we define its *linearized version* $\mathsf{lin}(r)$ as a term obtained from r by replacing each occurrence of a named variable in r by a fresh one. For instance, $\mathsf{lin}(f(x,\_,g(y,x,a),b)) = f(x',\_,g(y',x'',a),b)$, where x', x'', y' are fresh variables. Linearized versions of terms are unique modulo variable renaming.

Definition 4 (Generalization degree upper bound). *Given two terms r and t, a proximity relation $\mathcal{R}$, and a λ-cut, the* $(\mathcal{R},\lambda)$-generalization degree upper bound of r and t, *denoted by $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$, is defined as follows:*

Let $\alpha := \max\{\mathcal{R}(\mathsf{lin}(r)\sigma, t) \mid \sigma$ is a substitution$\}$. Then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$ is α if $\alpha \geqslant \lambda$, and 0 otherwise.

Intuitively, $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = \alpha$ means that no instance of r can get closer than α to t in $\mathcal{R}$. From the definition it follows that if $r \precsim_{\mathcal{R},\lambda} t$, then $0 < \lambda \leqslant \mathsf{gdub}_{\mathcal{R},\lambda}(r,t) \leqslant 1$ and if $r \not\precsim_{\mathcal{R},\lambda} t$, then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = 0$.

The upper bound computed by gdub is more relaxed than it would be if the linearization function were not used, but this is what we will be able to compute in our algorithms later.

Example 3. Let $\mathcal{R}(a,b) = 0.6$, $\mathcal{R}(b,c) = 0.7$, and $\lambda = 0.5$. Then $\mathsf{gdub}_{\mathcal{R},\lambda}(f(x,b), f(a,c)) = 0.7$ and $\mathsf{gdub}_{\mathcal{R},\lambda}(f(x,x), f(a,c)) = \mathsf{gdub}_{\mathcal{R},\lambda}(f(x,y), f(a,c)) = 1$.

It is not difficult to see that if $r\sigma \simeq_{\mathcal{R},\lambda} t$, then $\mathcal{R}(r\sigma, t) \leqslant \mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$. In Example 3, for $\sigma = \{x \mapsto b\}$ we have $\mathcal{R}(f(x,x)\sigma, f(a,c)) = \mathcal{R}(f(b,b), f(a,c)) = 0.6 < \mathsf{gdub}_{\mathcal{R},\lambda}(f(x,x), f(a,c)) = 1$.

We compute $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$ as follows: If r is a variable, then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = 1$. Otherwise, if $\mathsf{head}(r) \sim^{\rho}_{\mathcal{R},\beta} \mathsf{head}(t)$, then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = \beta \wedge \bigwedge_{(i,j)\in\rho} \mathsf{gdub}_{\mathcal{R},\lambda}(r|_i, t|_j)$. Otherwise, $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = 0$.

3 Term set consistency

The notion of term set consistency plays an important role in the computation of proximal generalizations. Intuitively, a set of terms is $(\mathcal{R},\lambda)$-consistent if all the terms in the set have a common $(\mathcal{R},\lambda)$-proximal term. In this section, we discuss this notion and the corresponding algorithms.

Definition 5 (Consistent set of terms). *A finite set of terms T is $(\mathcal{R},\lambda)$-*consistent *if there exists a term s such that $s \simeq_{\mathcal{R},\lambda} t$ for all $t \in T$.*

$(\mathcal{R},\lambda)$-consistency of a finite term set T is equivalent to $\bigcap_{t\in T} \mathbf{pc}_{\mathcal{R},\lambda}(t) \neq \emptyset$, but we cannot use this property to decide consistency, since proximity classes of terms can be infinite (when the argument relations are not restricted). For this reason, we introduce the operation $\sqcap$ on terms as follows: (i) $t \sqcap \_ = \_ \sqcap t = t$, (ii) $f(t_1,\dots,t_n) \sqcap f(s_1,\dots,s_n) = f(t_1 \sqcap s_1, \dots, t_n \sqcap s_n)$, $n \geqslant 0$. Obviously, $\sqcap$ is associative (A), commutative (C), idempotent (I), and has $\_$ as its unit element (U). It can be extended to sets of terms: $T_1 \sqcap T_2 := \{t_1 \sqcap t_2 \mid t_1 \in T_1, t_2 \in T_2\}$. It is easy to see that $\sqcap$ on sets also satisfies the ACIU properties with the set $\{\_\}$ playing the role of the unit element.

Lemma 2. *A finite set of terms T is $(\mathcal{R},\lambda)$-consistent iff $\bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \neq \emptyset$.*

Proof. ($\Rightarrow$) If $s \simeq_{\mathcal{R},\lambda} t$ for all $t \in T$, then $s_t \in \mathbf{rpc}_{\mathcal{R},\lambda}(t)$, where s_t is obtained from s by replacing all subterms that are irrelevant for its $(\mathcal{R},\lambda)$-proximity to t by $\_$. Assume $T = \{t_1,\dots,t_n\}$. Then $s_{t_1} \sqcap \cdots \sqcap s_{t_n} \in \bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$.

($\Leftarrow$) Obvious, since $s \simeq_{\mathcal{R},\lambda} t$ for $s \in \bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ and for all $t \in T$. □

Now we design an algorithm $\mathfrak{C}$ that computes $\bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ without actually computing $\mathbf{rpc}_{\mathcal{R},\lambda}(t)$ for each $t \in T$. A special version of the algorithm can be used to decide the $(\mathcal{R},\lambda)$-consistency of T.

The algorithm is rule-based. The rules work on states, that are pairs $\mathbf{I}; s$, where s is a term and $\mathbf{I}$ is a finite set of expressions of the form x in T, where T is a finite set of terms. $\mathcal{R}$ and λ are given. There are two rules ($\uplus$ stands for disjoint union):

Rem: **Removing the empty set**

$\{x \text{ in } \emptyset\} \uplus \mathbf{I}; s \Longrightarrow \mathbf{I}; s\{x \mapsto \_\}$.

Red: **Reduce a set to new sets**

$\{x \text{ in } \{t_1, \dots, t_m\}\} \uplus \mathbf{I}; s \Longrightarrow \{y_1 \text{ in } T_1, \dots, y_n \text{ in } T_n\} \cup \mathbf{I}; s\{x \mapsto h(y_1, \dots, y_n)\}$,

where $m \geqslant 1$, h is an n-ary function symbol such that $h \sim^{\rho_k}_{\mathcal{R},\gamma_k} \mathsf{head}(t_k)$ with $\gamma_k \geqslant \lambda$ for all $1 \leqslant k \leqslant m$, and $T_i := \{t_k|_j \mid (i,j) \in \rho_k, 1 \leqslant k \leqslant m\}$, $1 \leqslant i \leqslant n$, is the set of all those arguments of the terms $t_1, \dots, t_m$ that are supposed to be $(\mathcal{R}, \lambda)$-proximal to the i's argument of h.

To compute $\bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$, $\mathfrak{C}$ starts with $\{x \text{ in } T\}; x$ and applies the rules as long as possible. Red causes branching. A state of the form $\emptyset; s$ is called a success state. A failure state has the form $\mathbf{I}; s$, to which no rule applies and $\mathbf{I} \neq \emptyset$. In the full derivation tree, each leaf is a either success or a failure state.

Example 4. Assume a, b, c are constants, g, f, h are function symbols with the arities respectively 1, 2, and 3. Let λ be given and $\mathcal{R}$ be defined so that $\mathcal{R}(a,b) \geqslant \lambda$, $\mathcal{R}(b,c) \geqslant \lambda$, $h \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},\beta} f$, $h \sim^{\{(2,1)\}}_{\mathcal{R},\gamma} g$ with $\beta \geqslant \lambda$ and $\gamma \geqslant \lambda$. Then

$$\mathbf{rpc}_{\mathcal{R},\lambda}(f(a,c)) = \{f(a,c),\ f(b,c),\ f(a,b),\ f(b,b),\ h(b,\_,\_)\},$$
$$\mathbf{rpc}_{\mathcal{R},\lambda}(g(a)) = \{g(a),\ g(b), h(\_,a,\_), h(\_,b,\_)\},$$

and $\mathbf{rpc}_{\mathcal{R},\lambda}(f(a,c)) \sqcap \mathbf{rpc}_{\mathcal{R},\lambda}(g(a)) = \{h(b,a,\_), h(b,b,\_)\}$. We show how to compute this set with $\mathfrak{C}$: $\{x \text{ in } \{f(a,c), g(a)\}\};\ x \Longrightarrow_{\mathsf{Red}} \{y_1 \text{ in } \{a,c\}, y_2 : \{a\}, y_3 \text{ in } \emptyset\}$; $h(y_1, y_2, y_3) \Longrightarrow_{\mathsf{Rem}} \{y_1 \text{ in } \{a,c\}, y_2 : \{a\}\}; h(y_1, y_2, \_) \Longrightarrow_{\mathsf{Red}} \{y_2 \text{ in } \{a\}\}; h(b, y_2, \_)$. Here we have two ways to apply Red to the last state, leading to two elements of $\mathbf{rpc}_{\mathcal{R},\lambda}(f(a,c)) \sqcap \mathbf{rpc}_{\mathcal{R},\lambda}(g(a))$: $h(b,a,\_)$ and $h(b,b,\_)$.

Theorem 2. *Given a finite set of terms T, the algorithm $\mathfrak{C}$ always terminates starting from the state $\{x \text{ in } T\}; x$ (where x is a fresh variable). If S is the set of success states produced at the end, we have $\{s \mid \emptyset; s \in S\} = \bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$.*

Proof. Termination: Associate to each state $\{x_1 \text{ in } T_1, \dots x_n \text{ in } T_n\}; s$ the multiset $\{d_1, \dots, d_n\}$, where d_i is the maximum depth of terms occurring in T_i. $d_i = 0$ if $T_i = \emptyset$. Compare these multisets by the Dershowitz-Manna ordering [5]. Each rule strictly reduces them, which implies termination.

By the definitions of $\mathbf{rpc}_{\mathcal{R},\lambda}$ and $\sqcap$, $h(s_1, \dots, s_n) \in \bigsqcap_{t \in \{t_1, \dots, t_m\}} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ iff $h \sim^{\rho_k}_{\mathcal{R},\gamma_k} \mathsf{head}(t_k)$ with $\gamma_k \geqslant \lambda$ for all $1 \leqslant k \leqslant m$ and $s_i \in \bigsqcap_{t \in T_i} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$, where $T_i = \{t_k|_j \mid (i,j) \in \rho_k, 1 \leqslant k \leqslant m\}$, $1 \leqslant i \leqslant n$. Therefore, in the Rem rule, the instance of x (which is $h(y_1, \dots, y_n)$) is in $\bigsqcap_{t \in \{t_1, \dots, t_m\}} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ iff for each $1 \leqslant i \leqslant n$ we can find an instance of y_i in $\bigsqcap_{t \in T_i} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$. If T_i is empty, it means that the i's argument of h is irrelevant for terms in $\{t_1, \dots, t_m\}$ and can be replaced by $\_$. (Rem does it in a subsequent step.) Hence, in each success branch of the derivation tree, the algorithm $\mathfrak{C}$ computes one element of $\bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$. Branching at Red helps produce all elements of $\bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$. □

It is easy to see how to use $\mathfrak{C}$ to decide the $(\mathcal{R}, \lambda)$-consistency of T: it is enough to find one successful branch in the $\mathfrak{C}$-derivation tree for $\{x \text{ in } T\}; x$. If there is no such branch, then T is not $(\mathcal{R}, \lambda)$-consistent. In fact, during the derivation we can even ignore the second component of the states.

4 Solving generalization problems

Now we can reformulate the anti-unification problem that will be solved in the remaining part of the paper. $\mathcal{R}$ is a proximity relation and λ is a cut value.

Given: $\mathcal{R}$, λ, and the ground terms $t_1, \dots, t_n$, $n \geqslant 2$.

Find: a set S of tuples $(r, \sigma_1, \dots, \sigma_n, \alpha_1, \dots, \alpha_n)$ such that

- $\{r \mid (r, \dots) \in \mathsf{S}\}$ is an $(\mathcal{R}, \lambda)$-mcsrg of $t_1, \dots, t_n$,
- $r\sigma_i \simeq_{\mathcal{R},\lambda} t_i$ and $\alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$, $1 \leqslant i \leqslant n$, for each $(r, \sigma_1, \dots, \sigma_n, \alpha_1, \dots, \alpha_n) \in \mathsf{S}$.

(When $n = 1$, this is a problem of computing a relevant proximity class of a term.) Below we give a set of rules, from which one can obtain algorithms to solve the anti-unification problem for four versions of argument relations:

1. The most general (unrestricted) case; see algorithm $\mathfrak{A}_1$ below, the computed set of generalizations is an mcsrg;
2. Correspondence relations: using the same algorithm $\mathfrak{A}_1$, the computed set of generalizations is an mcsg;
3. Mappings: using a dedicated algorithm $\mathfrak{A}_2$, the computed set of generalizations is an mcsrg;
4. Correspondence mappings (bijections): using the same algorithm $\mathfrak{A}_2$, the computed set of generalizations is an mcsg.

Each of them has also the corresponding linear variant, computing minimal complete sets of (relevant) linear $(\mathcal{R}, \lambda)$-generalizations. They are denoted by adding the superscript lin to the corresponding algorithm name: $\mathfrak{A}_1^{\mathsf{lin}}$ and $\mathfrak{A}_2^{\mathsf{lin}}$.

For simplicity, we formulate the algorithms for the case $n = 2$. They can be extended for arbitrary n straightforwardly.

The main data structure in these algorithms is an anti-unification triple (AUT) $x : T_1 \triangleq T_2$, where T_1 and T_2 are finite *consistent* sets of ground terms. The idea is that x is a common generalization of all terms in $T_1 \cup T_2$. A configuration is a tuple $A; S; r; \alpha_1; \alpha_2$, where A is a set of AUTs to be solved, S is a set of solved AUTs (the store), r is the generalization computed so far, and the α's are the current approximations of generalization degree upper bounds of r for the input terms.

Before formulating the rules, we discuss one peculiarity of approximate generalizations:

Example 5. For a given $\mathcal{R}$ and λ, assume $\mathcal{R}(a, b) \geqslant \lambda$, $\mathcal{R}(b, c) \geqslant \lambda$, $h \sim_{\mathcal{R},\alpha}^{\{(1,1),(1,2)\}} f$ and $h \sim_{\mathcal{R},\beta}^{\{(1,1)\}} g$, where f is binary, g, h are unary, $\alpha \geqslant \lambda$ and $\beta \geqslant \lambda$. Then

- $h(b)$ is an $(\mathcal{R}, \lambda)$-generalization of $f(a, c)$ and $g(a)$.
- x is the only $(\mathcal{R}, \lambda)$-generalization of $f(a, d)$ and $g(a)$. One may be tempted to have h as the head of the generalization, e.g., $h(x)$, but x cannot be instantiated by any term that would be $(\mathcal{R}, \lambda)$-close to both a and d, since in the given $\mathcal{R}$, d is $(\mathcal{R}, \lambda)$-close only to itself. Hence, there would be no instance

of $h(x)$ that is $(\mathcal{R},\lambda)$-close to $f(a,d)$. Since there is no other alternative (except h) for the common neighbor of f and g, the generalization should be a fresh variable x.

This example shows that generalization algorithms should take into account not only the heads of the terms to be generalized, but also should look deeper, to make sure that the arguments grouped together by the given argument relation have a common neighbor. This justifies the requirement of consistency of a set of arguments, the notion introduced in the previous section and used in the decomposition rule below.

Anti-Unification for unrestricted argument relations. Algorithms $\mathfrak{A}_1^{\mathsf{lin}}$ and $\mathfrak{A}_1$ use the rules below to transform configurations into configurations. Given $\mathcal{R}$, λ, and the ground terms t_1 and t_2, we create the initial configuration $\{x : \{t_1\} \triangleq \{t_2\}\}; \emptyset; x; 1; 1$ and apply the rules as long as possible. Note that the rules preserve consistency of AUTs. The process generates a finite complete tree of derivations, whose terminal nodes have configurations with the first component empty. We will show how from these terminal configurations one collects the result as required in the anti-unification problem statement.

Tri: **Trivial**

$\{x : \emptyset \triangleq \emptyset\} \uplus A;\ S;\ r;\ \alpha_1; \alpha_2 \Longrightarrow A;\ S;\ r\{x \mapsto \_\};\ \alpha_1; \alpha_2.$

Dec: **Decomposition**

$\{x : T_1 \triangleq T_2\} \uplus A; S; r; \alpha_1; \alpha_2 \Longrightarrow$
$\qquad \{y_i : Q_{i1} \triangleq Q_{i2} \mid 1 \leqslant i \leqslant n\} \cup A; S; r\{x \mapsto h(y_1, \dots, y_n)\}; \alpha_1 \wedge \beta_1; \alpha_2 \wedge \beta_2,$

where $T_1 \cup T_2 \neq \emptyset$; h is n-ary with $n \geqslant 0$; $y_1, \dots, y_n$ are fresh; and for $j = 1, 2$, if $T_j = \{t_1^j, \dots, t_{m_j}^j\}$, then

- $h \sim^{\rho_k^j}_{\mathcal{R},\gamma_k^j} \mathsf{head}(t_k^j)$ with $\gamma_k^j \geqslant \lambda$ for all $1 \leqslant k \leqslant m_j$ and $\beta_j = \gamma_1^j \wedge \cdots \wedge \gamma_{m_j}^j$ (note that $\beta_j = 1$ if $m_j = 0$),
- for all $1 \leqslant i \leqslant n$, $Q_{ij} = \cup_{k=1}^{m_j} \{t_k^j|_q \mid (i,q) \in \rho_k^j\}$ and is $(\mathcal{R},\lambda)$-consistent.

Sol: **Solving**

$\{x : T_1 \triangleq T_2\} \uplus A;\ S;\ r;\ \alpha_1; \alpha_2 \Longrightarrow A;\ \{x : T_1 \triangleq T_2\} \cup S;\ r;\ \alpha_1; \alpha_2,$

if Tri and Dec rules are not applicable. (It means that at least one $T_i \neq \emptyset$ and either there is no h as it is required in the Dec rule, or at least one Q_{ij} from Dec is not $(\mathcal{R},\lambda)$-consistent.)

Let expand be an *expansion operation* defined for sets of AUTs as

$$\mathsf{expand}(S) := \{x : \bigsqcap_{t \in T_1} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \triangleq \bigsqcap_{t \in T_2} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \mid x : T_1 \triangleq T_2 \in S\}.$$

Exhaustive application of the three rules above leads to configurations of the form $\emptyset; S; r; \alpha_1; \alpha_2$, where r is a linear term. These configurations are further postprocessed, replacing S by $\mathsf{expand}(S)$. We will use the letter E for expanded stores. Hence, terminal configurations obtained after the exhaustive rule application and expansion have the form $\emptyset; E; r; \alpha_1; \alpha_2$, where r is a linear term.[1] This is what Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ stops with.

To an expanded store $E = \{y_1 : Q_{11} \triangleq Q_{12}, \dots, y_n : Q_{n1} \triangleq Q_{n2}\}$ we associate two sets of substitutions $\Sigma_L(E)$ and $\Sigma_R(E)$, defined as follows: $\sigma \in \Sigma_L(E)$ (resp. $\sigma \in \Sigma_R(E)$) iff $\mathsf{dom}(\sigma) = \{y_1, \dots, y_n\}$ and $y_i\sigma \in Q_{i1}$ (resp. $y_i\sigma \in Q_{i2}$) for each $1 \leqslant i \leqslant n$. We call them the sets of *witness substitutions*.

Configurations containing expanded stores are called *expanded configurations*. From each expanded configuration $C = \emptyset; E; r; \alpha_1; \alpha_2$, we construct the set $\mathsf{S}(C) := \{(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \mid \sigma_1 \in \Sigma_L(E), \sigma_2 \in \Sigma_R(E)\}$.

Given an anti-unification problem $\mathcal{R}$, λ, t_1 and t_2, the *answer computed by Algorithm* $\mathfrak{A}_1^{\mathsf{lin}}$ is the set $\mathsf{S} := \cup_{i=1}^m \mathsf{S}(C_i)$, where $C_1, \dots, C_m$ are all of the final expanded configurations reached by $\mathfrak{A}_1^{\mathsf{lin}}$ for $\mathcal{R}$, λ, t_1, and t_2.[2]

Example 6. Assume a, b, c and d are constants with $b \sim^{\emptyset}_{\mathcal{R},0.5} c$, $c \sim^{\emptyset}_{\mathcal{R},0.6} d$, and f, g and h are respectively binary, ternary and quaternary function symbols with $h \sim^{\{(1,1),(3,2),(4,2)\}}_{\mathcal{R},0.7} f$ and $h \sim^{\{(1,1),(3,3)\}}_{\mathcal{R},0.8} g$. For the proximity relation $\mathcal{R}$ given in this way and $\lambda = 0.5$, Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ performs the following steps to anti-unify $f(a, b)$ and $g(a, c, d)$:

$$\begin{aligned}
&\{x : \{f(a,b)\} \triangleq \{g(a,c,d)\}\}; \emptyset; x; 1; 1 \Longrightarrow_{\mathsf{Dec}}\\
&\{x_1 : \{a\} \triangleq \{a\},\ x_2 : \emptyset \triangleq \emptyset,\ x_3 : \{b\} \triangleq \{d\},\\
&\qquad x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(x_1, x_2, x_3, x_4); 0.7; 0.8 \Longrightarrow_{\mathsf{Dec}}\\
&\{x_2 : \emptyset \triangleq \emptyset,\ x_3 : \{b\} \triangleq \{d\},\ x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(a, x_2, x_3, x_4); 0.7; 0.8 \Longrightarrow_{\mathsf{Tri}}\\
&\{x_3 : \{b\} \triangleq \{d\},\ x_4 : \{b\} \triangleq \emptyset\};\ \emptyset; h(a, \_, x_3, x_4); 0.7; 0.8 \Longrightarrow_{\mathsf{Dec}}\\
&\{x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(a, \_, c, x_4); 0.5; 0.6.
\end{aligned}$$

Here Dec applies in two different ways, with the substitutions $\{x_4 \mapsto b\}$ and $\{x_4 \mapsto c\}$, leading to two final configurations: $\emptyset; \emptyset; h(a, \_, c, b); 0.5; 0.6$ and $\emptyset; \emptyset; h(a, \_, c, c); 0.5; 0.6$. The witness substitutions are the identity substitutions. Then we have $\mathcal{R}(h(a, \_, c, b), f(a, b)) = 0.5$, $\mathcal{R}(h(a, \_, c, b), g(a, c, d)) = 0.6$, $\mathcal{R}(h(a, \_, c, c), f(a, b)) = 0.5$, and $\mathcal{R}(h(a, \_, c, c), g(a, c, d)) = 0.6$.

If we had $h \sim^{\{(1,1),(1,2),(4,2)\}}_{\mathcal{R},0.7} f$, then the algorithm would perform only the Sol step, because in the attempt to apply Dec to the initial configuration, the set $Q_{11} = \{a, b\}$ is inconsistent: $\mathbf{rpc}_{\mathcal{R},\lambda}(a) = \{a\}$, $\mathbf{rpc}_{\mathcal{R},\lambda}(b) = \{b, c\}$, and, hence, $\mathbf{rpc}_{\mathcal{R},\lambda}(a) \sqcap \mathbf{rpc}_{\mathcal{R},\lambda}(b) = \emptyset$.

Algorithm $\mathfrak{A}_1$ is obtained by further transforming the expanded configurations produced by $\mathfrak{A}_1^{\mathsf{lin}}$. This transformation is performed by applying the **Merge** rule below as long as possible. Intuitively, its purpose is to make the linear generalization obtained by $\mathfrak{A}_1^{\mathsf{lin}}$ less general by merging some variables.

[1] Note that no side of the AUTs in E in those configurations is empty due to the condition at the **Decomposition** rule requiring the Q_{ij}'s to be $(\mathcal{R}, \lambda)$-consistent.

[2] If we are interested only in linear generalizations *without witness substitutions*, there is no need in computing expanded configurations in $\mathfrak{A}_1^{\mathsf{lin}}$.

Mer: **Merge**

$$\varnothing; \{x_1 : R_{11} \triangleq R_{12},\, x_2 : R_{21} \triangleq R_{22}\} \uplus E;\, r;\, \alpha_1;\, \alpha_2 \Longrightarrow$$
$$\varnothing;\, \{y : Q_1 \triangleq Q_2\} \cup E;\, r\sigma;\, \alpha_1;\, \alpha_2,$$

where $Q_i = (R_{1i} \sqcap R_{2i}) \neq \varnothing$, $i = 1, 2$, y is fresh, and $\sigma = \{x_1 \mapsto y,\, x_2 \mapsto y\}$.

The answer computed by $\mathfrak{A}_1$ is defined similarly to the answer computed by $\mathfrak{A}_1^{\mathsf{lin}}$.

Example 7. Assume a, b are constants, f_1, f_2, g_1, and g_2 are unary function symbols, p is a binary function symbol, and h_1 and h_2 are ternary function symbols. Let λ be a cut value and $\mathcal{R}$ be defined as $f_i \sim^{\{(1,1)\}}_{\mathcal{R},\alpha_i} h_i$ and $g_i \sim^{\{(1,2)\}}_{\mathcal{R},\beta_i} h_i$ with $\alpha_i \geqslant \lambda$, $\beta_i \geqslant \lambda$, $i = 1, 2$. To generalize $p(f_1(a), g_1(b))$ and $p(f_2(a), g_2(b))$, we use $\mathfrak{A}_1$. The derivation starts as

$$\{x : \{p(f_1(a), g_1(b))\} \triangleq \{p(f_2(a), g_2(b))\}\};\, \varnothing;\, x;\, 1;\, 1 \Longrightarrow_{\mathsf{Dec}}$$
$$\{y_1 : \{f_1(a)\} \triangleq \{f_2(a)\},\, y_2 : \{g_1(b)\} \triangleq \{g_2(b)\}\};\, \varnothing;\, p(y_1, y_2);\, 1;\, 1 \Longrightarrow^2_{\mathsf{Sol}}$$
$$\varnothing;\, \{y_1 : \{f_1(a)\} \triangleq \{f_2(a)\},\, y_2 : \{g_1(b)\} \triangleq \{g_2(b)\}\};\, p(y_1, y_2);\, 1;\, 1.$$

At this stage, we expand the store, obtaining

$$\varnothing;\, \{y_1 : \{f_1(a), h_1(a, \_, \_)\} \triangleq \{f_2(a), h_2(a, \_, \_)\},$$
$$y_2 : \{g_1(b), h_1(\_, b, \_)\} \triangleq \{g_2(b), h_2(\_, b, \_)\}\};\, p(y_1, y_2);\, 1;\, 1.$$

If we had the standard intersection $\cap$ in the Mer rule, we would not be able to merge y_1 and y_2, because the obtained sets in the corresponding AUTs are disjoint. However, Mer uses $\sqcap$: we have $\{f_i(a), h_i(a, \_, \_)\} \sqcap \{g_i(b), h_i(\_, b, \_)\} = \{h_i(a, b, \_)\}$, $i = 1, 2$ and, therefore, can make the step

$$\varnothing;\, \{y_1 : \{f_1(a), h_1(a, \_, \_)\} \triangleq \{f_2(a), h_2(a, \_, \_)\},$$
$$y_2 : \{g_1(b), h_1(\_, b, \_)\} \triangleq \{g_2(b), h_2(\_, b, \_)\}\};\, p(y_1, y_2);\, 1;\, 1 \Longrightarrow_{\mathsf{Mer}}$$
$$\varnothing;\, \{z : \{h_1(a, b, \_)\} \triangleq \{h_2(a, b, \_)\}\};\, p(z, z);\, 1;\, 1.$$

Indeed, if we take the witness substitutions $\sigma_i = \{z \mapsto h_i(a, b, \_)\}$, $i = 1, 2$, and apply them to the obtained generalization, we get

$$p(z, z)\sigma_1 = p(h_1(a, b, \_), h_1(a, b, \_)) \simeq_{\mathcal{R},\lambda} p(f_1(a), g_1(b)),$$
$$p(z, z)\sigma_2 = p(h_2(a, b, \_), h_2(a, b, \_)) \simeq_{\mathcal{R},\lambda} p(f_2(a), g_2(b)).$$

Theorem 3. *Given $\mathcal{R}$, λ, and the ground terms t_1 and t_2, Algorithm $\mathfrak{A}_1$ terminates for $\{x : \{t_1\} \triangleq \{t_2\}\}; \varnothing; x; 1; 1$ and computes an answer set S such that*

1. *the set $\{r \mid (r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}\}$ is an $(\mathcal{R}, \lambda)$-mcsrg of t_1 and t_2,*
2. *for each $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$ we have $\mathcal{R}(r\sigma_i, t_i) \leqslant \alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$, $i = 1, 2$.*

Proof. Termination: Define the depth of an AUT $x : \{t_1, \ldots, t_m\} \triangleq \{s_1, \ldots, s_n\}$ as the depth of the term $f(g(t_1, \ldots, t_m), h(s_1, \ldots, s_n))$. The rules Tri, Dec, and Sol strictly reduce the multiset of depths of AUTs in the first component of the configurations. Mer strictly reduces the number of distinct variables in generalizations. Hence, these rules cannot be applied infinitely often and $\mathfrak{A}_1$ terminates.

In order to prove 1), we need to verify three properties:

- Soundness: If $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$, then r is a relevant $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2.
- Completeness: If r' is a relevant $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2, then there exists $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$ such that $r' \lesssim r$.
- Minimality: If r and r' belong to two tuples from S such that $r \neq r'$, then neither $r \prec_{\mathcal{R},\lambda} r'$ nor $r' \prec_{\mathcal{R},\lambda} r$.

Soundness: We show that each rule transforms an $(\mathcal{R}, \lambda)$-generalization into an $(\mathcal{R}, \lambda)$-generalization. Since we start from a most general $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2 (a fresh variable x), at the end of the algorithm we will get an $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2. We also show that in this process all irrelevant positions are abstracted by anonymous variables, to guarantee that each computed generalization is relevant.

Dec: The computed h is $(\mathcal{R}, \lambda)$-close to the head of each term in $T_1 \cup T_2$. Q_{ij}'s correspond to argument relations between h and those heads, and each Q_{ij} is $(\mathcal{R}, \lambda)$-consistent, i.e., there exists a term that is $(\mathcal{R}, \lambda)$-close to each term in Q_{ij}. It implies that $x\sigma = h(y_1, \dots, y_n)$ $(\mathcal{R}, \lambda)$-generalizes all the terms from $T_1 \cup T_2$. Note that at this stage, $h(y_1, \dots, y_n)$ might not yet be a relevant $(\mathcal{R}, \lambda)$-generalization of T_1 and T_2: if there exists an irrelevant position $1 \leqslant i \leqslant n$ for the $(\mathcal{R}, \lambda)$-generalization of T_1 and T_2, then in the new configuration we will have an AUT $y_i : \varnothing \triangleq \varnothing$.

Tri: When Dec generates $y : \varnothing \triangleq \varnothing$, the Tri rule replaces y by $\_$ in the computed generalization, making it relevant.

Sol does not change generalizations.

Mer merges AUTs whose terms have *nonempty* intersection of **rpc**'s. Hence, we can reuse the same variable in the corresponding positions in generalizations, i.e., Mer transforms a generalization computed so far into a less general one.

Completeness: We prove a slightly more general statement. Given two finite consistent sets of ground terms T_1 and T_2, if r' is a relevant $(\mathcal{R}, \lambda)$-generalization for all $t_1 \in T_1$ and $t_2 \in T_2$, then starting from $\{x : T_1 \triangleq T_2\}; \varnothing; x; 1; 1$, Algorithm $\mathfrak{A}_1$ computes a $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2)$ such that $r' \lesssim r$.

We may assume w.l.o.g. that r' is a relevant $(\mathcal{R}, \lambda)$-lgg. Due to the transitivity of $\lesssim$, completeness for such an r' will imply it for all terms more general than r'.

We proceed by structural induction on r'. If r' is a (named or anonymous) variable, the statement holds. Assume $r' = h(r'_1, \dots, r'_n)$, $T_1 = \{u_1, \dots, u_m\}$, and $T_2 = \{w_1, \dots, w_l\}$. Then h is such that $h \sim^{\rho_i}_{\mathcal{R},\beta_i} \mathsf{head}(u_i)$ for all $1 \leqslant i \leqslant m$ and $h \sim^{\mu_j}_{\mathcal{R},\gamma_j} \mathsf{head}(w_j)$ for all $1 \leqslant j \leqslant l$. Moreover, each r'_k is a relevant $(\mathcal{R}, \lambda)$-generalization of $Q_{k1} = \cup_{i=1}^{m} \{u_i|_q \mid (k, q) \in \rho_i\}$ and $Q_{k2} = \cup_{j=1}^{l} \{w_j|_q \mid (k, q) \in \mu_j\}$ and, hence, Q_{k1} and Q_{k2} are $(\mathcal{R}, \lambda)$-consistent. Therefore, we can perform a step by Dec, choosing $h(y_1, \dots, y_k)$ as the generalization term and $y_i : Q_{i1} \triangleq Q_{i2}$

as the new AUTs. By the induction hypothesis, for each $1 \leqslant i \leqslant n$ we can compute a relevant $(\mathcal{R}, \lambda)$-generalization r_i for Q_{i_1} and Q_{i2} such that $r'_i \lesssim r_i$.

If r' is linear, then the combination of the current Dec step with the derivations that lead to those r_i's computes a tuple $(r, \ldots) \in \mathsf{S}$, where $r = h(r_1, \ldots, r_n)$ and, hence, $r' \lesssim r$.

If r' is non-linear, assume w.l.o.g. that all occurrences of a shared variable z appear as the direct arguments of h: $z = r'_{k_1} = \cdots = r'_{k_p}$ for $1 \leqslant k_1 < \cdots < k_p \leqslant n$. Since r' is an lgg, $Q_{k_i 1}$ and $Q_{k_i 2}$ cannot be generalized by a non-variable term, thus, Tri and Dec are not applicable. Therefore, the AUTs $y_i : Q_{k_i 1} \triangleq Q_{k_i 2}$ would be transformed by Sol. Since all pairs $Q_{k_i 1}$ and $Q_{k_i 2}$, $1 \leqslant i \leqslant p$, are generalized by the same variable, we have $\sqcap_{t \in Q_j} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \neq \emptyset$, where $Q_j = \cup_{i=1}^{p} Q_{k_i j}$, $j = 1, 2$. Additionally, $r'_{k_1}, \ldots, r'_{k_p}$ are all occurrences of z in r'. Hence, the condition of Mer is satisfied and we can extend our derivation with $p - 1$-fold application of this rule, obtaining $r = h(r_1, \ldots, r_n)$ with $z = r_{k_1} = \cdots = r_{k_p}$, implying $r' \lesssim r$.

Minimality: Alternative generalizations are obtained by branching in Dec or Mer. If the current generalization r is transformed by Dec into two generalizations r_1 and r_2 on two branches, then $r_1 = h_1(y_1, \ldots, y_m)$ and $r_2 = h_2(z_1, \ldots, z_n)$ for some h's, and fresh y's and z's. It may happen that $r_1 \lesssim_{\mathcal{R},\lambda} r_2$ or vice versa (if h_1 and h_2 are $(\mathcal{R}, \lambda)$-close to each other), but neither $r_1 \prec_{\mathcal{R},\lambda} r_2$ nor $r_2 \prec_{\mathcal{R},\lambda} r_1$ holds. Hence, the set of generalizations computed before applying Mer is minimal. Mer groups AUTs together maximally, and different groupings are not comparable. Therefore, variables in generalizations are merged so that distinct generalizations are not $\prec_{\mathcal{R},\lambda}$-comparable. Hence, 1) is proven.

As for 2), for $i = 1, 2$, from the construction in Dec follows $\mathcal{R}(r\sigma_i, t_i) \leqslant \alpha_i$. Mer does not change α_i, thus, $\alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$ also holds, since the way how α_i is computed corresponds exactly to the computation of $\mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$: $r \lesssim_{\mathcal{R},\lambda} t_i$ and only the decomposition changes the degree during the computation. □

Corollary 1. *Given $\mathcal{R}$, λ, and the ground terms t_1 and t_2, Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ terminates for $\{x : \{t_1\} \triangleq \{t_2\}\}; \emptyset; x; 1; 1$ and computes an answer set S such that*

1. *the set $\{r \mid (r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}\}$ is a minimal complete set of relevant* linear *$(\mathcal{R}, \lambda)$-generalizations of t_1 and t_2,*
2. *for each $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$ we have $\mathcal{R}(r\sigma_i, t_i) \leqslant \alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$, $i = 1, 2$.*

Anti-Unification with correspondence argument relations. Correspondence relations make sure that for a pair of proximal symbols, no argument is irrelevant for proximity. Left- and right-totality of those relations guarantee that each argument of a term is close to at least one argument of its proximal term and the inverse relation remains a correspondence relation. Consequently, in the Dec rule of $\mathfrak{A}_1$, the sets Q_{ij} never get empty. Therefore, the Tri rule becomes obsolete and no anonymous variable appears in generalizations. As a result, the $(\mathcal{R}, \lambda)$-mcsrg and the $(\mathcal{R}, \lambda)$-mcsg coincide, and the algorithm computes a solution from which we get an $(\mathcal{R}, \lambda)$-mcsg for the given anti-unification problem. The linear version $\mathfrak{A}_1^{\mathsf{lin}}$ works analogously.

Anti-Unification with argument mappings. When the argument relations are mappings, we are able to design a more constructive method for computing generalizations and their degree bounds. (Recall that our mappings are partial injective functions, which guarantees that their inverses are also mappings.) The configurations stay the same as in before, but the AUTs in A will contain only empty or singleton sets of terms. In the store, we may still get (after the expansion) AUTs with term sets containing more than one element. Only the Dec rule differs from its previous counterpart.

Dec: **Decomposition**

$$\{x : T_1 \triangleq T_2\} \uplus A; S; r; \alpha_1; \alpha_2 \Longrightarrow$$
$$\{y_i : Q_{i1} \triangleq Q_{i2} \mid 1 \leqslant i \leqslant n\} \cup A; S; r\{x \mapsto h(y_1, \dots, y_n)\}; \alpha_1 \wedge \beta_1; \alpha_2 \wedge \beta_2,$$

where $T_1 \cup T_2 \neq \varnothing$; h is n-ary with $n \geqslant 0$; $y_1, \dots, y_n$ are fresh; for $j = 1, 2$ and for all $1 \leqslant i \leqslant n$, if $T_j = \{t_j\}$ then $h \sim^{\pi_j}_{\mathcal{R},\beta_j} \mathsf{head}(t_j)$ and $Q_{ij} = \{t_j|_{\pi_j(i)}\}$, and if $T_j = \varnothing$ then $\beta_j = 1$ and $Q_{ij} = \varnothing$.

This Dec rule is equivalent to the special case of Dec for argument relations where $m_j \leqslant 1$. The new Q_{ij}'s contain at most one element (due to mappings) and, thus, are always $(\mathcal{R}, \lambda)$-consistent. Various choices of h in Dec and alternatives in grouping AUTs in Mer cause branching in the same way as in $\mathfrak{A}_1$. It is easy to see that the counterparts of Theorem 3 hold for $\mathfrak{A}_2$ and $\mathfrak{A}_2^{\mathsf{lin}}$ as well.

A special case of this fragment of anti-unification is anti-unification for similarity relations in full fuzzy signatures from [1]. Similarity relations are min-transitive proximity relations. The position mappings in [1] can be modeled by our argument mappings, requiring them to be total for symbols of the smaller arity and to satisfy the similarity-specific consistency restrictions from [1].

Anti-Unification with correspondence argument mappings. Correspondence argument mappings are bijections between arguments of function symbols of the same arity. For such mappings, if $h \simeq^{\pi}_{\mathcal{R},\lambda} f$ and h is n-ary, then f is also n-ary and π is a permutation of $(1, \dots, n)$. Hence, $\mathfrak{A}_2$ combines in this case the properties of $\mathfrak{A}_1$ for correspondence relations (Section 4) and of $\mathfrak{A}_2$ for argument mappings (Section 4): all generalizations are relevant, computed answer gives an mcsg of the input terms, and the algorithm works with term sets of cardinality at most 1.

5 Remarks about the complexity

The proximity relation $\mathcal{R}$ can be naturally represented as an undirected graph, where the vertices are function symbols and an edge between them indicates that they are proximal. Graphs induced by proximity relations are usually sparse. Therefore we can represent them by (sorted) adjacency lists. In the adjacency lists, we can also accommodate the argument relations and proximity degrees.

In the rest of this section we use the following notation:

- n: the size of the input (number of symbols) of the corresponding algorithms,

- Δ: the maximum degree of $\mathcal{R}$ considered as a graph,
- $\mathfrak{a}$: the maximum arity of function symbols that occur in $\mathcal{R}$.
- $m^{\bullet n}$: a function defined on natural numbers m and n such that $1^{\bullet n} = n$ and $m^{\bullet n} = m^n$ for $m \neq 1$.

We assume that the given anti-unification problem is represented as a completely shared directed acyclic graph (dag). Each node of the dag has a pointer to the adjacency list (with respect to $\mathcal{R}$) of the symbol in the node.

Theorem 4. *Time complexities of $\mathfrak{C}$ and the linear versions of the generalization algorithms are as follows:*

- *$\mathfrak{C}$ for argument relations and $\mathfrak{A}_1^{\mathsf{lin}}$: $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet \mathfrak{a}^{\bullet n}})$,*
- *$\mathfrak{C}$ for argument mappings and $\mathfrak{A}_2^{\mathsf{lin}}$: $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet n})$.*

Proof (Sketch). In $\mathfrak{C}$, in the case of argument relations, an application of the Red rule to a state $\mathbf{I}; s$ replaces one element of $\mathbf{I}$ of size m by at most $\mathfrak{a}$ new elements, each of them of size $m - 1$. Hence, one branch in the search tree for $\mathfrak{C}$, starting from a singleton set $\mathbf{I}$ of size n, will have the length at most $l = \sum_{i=0}^{n-1} \mathfrak{a}^i$. At each node on it there are at most $\mathit{\Delta}$ choices of applying Red with different h's, which gives the total size of the search tree to be at most $\sum_{i=0}^{l-1} \mathit{\Delta}^i$, i.e., the number of steps performed by $\mathfrak{C}$ in the worst case is $O(\mathit{\Delta}^{\bullet \mathfrak{a}^{\bullet n}})$. Those different h's are obtained by intersecting the proximity classes of the heads of terms $\{t_1, \ldots, t_m\}$ in the Red rule. In our graph representation of the proximity relation, proximity classes of symbols are exactly the adjacency lists of those symbols which we assume are sorted. Their maximal length is $\mathit{\Delta}$. Hence, the work to be done at each node of the search tree of $\mathfrak{C}$ is to find the intersection of at most n sorted lists, each containing at most $\mathit{\Delta}$ elements. It needs $O(n \cdot \mathit{\Delta})$ time. It gives the time complexity $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet \mathfrak{a}^{\bullet n}})$ of $\mathfrak{C}$ for the relation case.

In the mapping case, an application of the Red rule to a state $\mathbf{I}; s$ replaces one element of $\mathbf{I}$ of size m by at most $\mathfrak{a}$ new elements of the *total* size $m - 1$. Therefore, the maximal length of a branch is n, the branching factor is $\mathit{\Delta}$, and the amount of work at each node, like above, is $O(n \cdot \mathit{\Delta})$. Hence, the number of steps in the worst case is $O(\mathit{\Delta}^{\bullet n})$ and the time complexity of $\mathfrak{C}$ is $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet n})$.

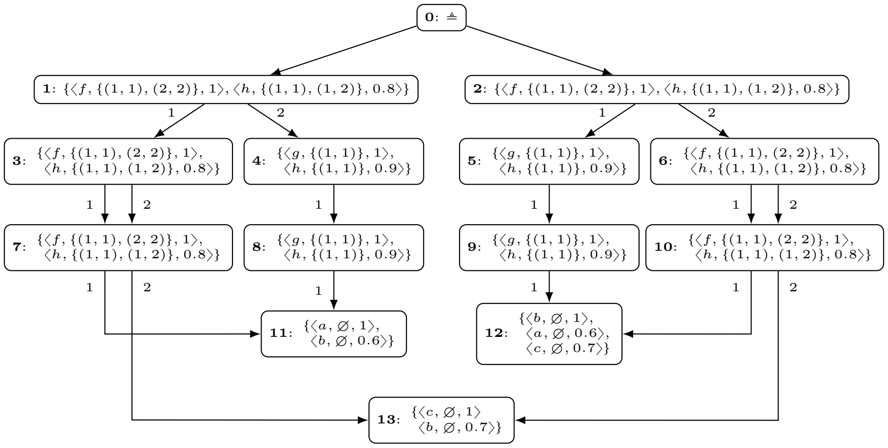
The fact that consistency check is incorporated in the Dec rule in $\mathfrak{A}_1^{\mathsf{lin}}$ can be used to guide the application of this rule, using the values memoized by the previous applications of Red. The very first time, the appropriate h in Dec is chosen arbitrarily. In any subsequent application of this rule, h is chosen according to the result of the Red rule that has already been applied to the arguments of the current AUT for their consistency check, as required by the condition of Dec. In this way, the applications of Dec and Sol will correspond to the applications of Red. There is a natural correspondence between the applications of Rem and Tri rules. Therefore, $\mathfrak{A}_1^{\mathsf{lin}}$ will have the search tree analogous to that of $\mathfrak{C}$. Hence the complexity of $\mathfrak{A}_1^{\mathsf{lin}}$ is $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet \mathfrak{a}^{\bullet n}})$. $\mathfrak{A}_2^{\mathsf{lin}}$ does not call the consistency check, but does the same work as $\mathfrak{C}$ and, hence, has the same complexity $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet n})$. $\square$

6 An extended example

In this section we illustrate the details of computing $(\mathcal{R},\lambda)$-lggs by $\mathfrak{A}_1^{\mathsf{lin}}$.

Assume a, b, c, d are constants, f and h are binary function symbols, and g is a unary function symbol. Let $\mathcal{R}$ be defined as $a \sim_{\mathcal{R},0.6}^{\emptyset} b$, $b \sim_{\mathcal{R},0.7}^{\emptyset} c$, $f \sim_{\mathcal{R},0.8}^{\{(1,1),(2,1)\}} h$, and $h \sim_{\mathcal{R},0.9}^{\{(1,1)\}} g$.

Take $\lambda = 0.5$ and consider the anti-unification problem between $f(f(f(a,c), f(a,c)), g(g(a)))$ and $f(g(g(b)), f(f(b,c), f(b,c)))$. The dag representation of the problem looks as follows (the bold face numbers are the node IDs):



0: $\triangleq$
1: $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$
2: $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$
1
2
1
2
3: $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$
4: $\{\langle g, \{(1,1)\}, 1\rangle, \langle h, \{(1,1)\}, 0.9\rangle\}$
5: $\{\langle g, \{(1,1)\}, 1\rangle, \langle h, \{(1,1)\}, 0.9\rangle\}$
6: $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$
1
2
1
1
1
2
7: $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$
8: $\{\langle g, \{(1,1)\}, 1\rangle, \langle h, \{(1,1)\}, 0.9\rangle\}$
9: $\{\langle g, \{(1,1)\}, 1\rangle, \langle h, \{(1,1)\}, 0.9\rangle\}$
10: $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$
1
2
1
1
1
2
11: $\{\langle a, \emptyset, 1\rangle, \langle b, \emptyset, 0.6\rangle\}$
12: $\{\langle b, \emptyset, 1\rangle, \langle a, \emptyset, 0.6\rangle, \langle c, \emptyset, 0.7\rangle\}$
13: $\{\langle c, \emptyset, 1\rangle \langle b, \emptyset, 0.7\rangle\}$

Fig. 1. Dag representation of the anti-unification problem between $f(f(f(a,c), f(a,c)), g(g(a)))$ and $f(g(g(b)), f(f(b,c), f(b,c)))$.

Each subgraph of this graph is a compact representation of a set of terms that form the proximity class of the corresponding subterm in the problem. For instance, the subgraph at node **3** is a compact representation of the proximity class of the subterm $f(f(a,c), f(a,c))$. The label $\{\langle f, \{(1,1),(2,2)\}, 1\rangle, \langle h, \{(1,1),(1,2)\}, 0.8\rangle\}$ of **3** is the adjacency list of f in $\mathcal{R}$ (containing the argument relation and the proximity degree for each symbol proximal to f).

Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ starts with the configuration

$$\begin{aligned}&\{x : \{f(f(f(a,c), f(a,c)), g(g(a)))\} \triangleq \{f(g(g(b)), f(f(b,c), f(b,c)))\}\};\\ &\qquad \emptyset;\ x;\ 1;\ 1\end{aligned}$$

The attempt to apply the **Decomposition** rule involves checking whether the labels at nodes **1** (i.e., the adjacency list of f) and **2** (the adjacency list of the same f) have a common symbol. There are actually two: f (with the argument relation $\{(1,1),(2,2)\}$) and h (with the argument relation $\{(1,1),(1,2)\}$).

The next step is the consistency check. For the case of f, we should check whether the set of terms at nodes **3** (corresponds to Q_{11} in the Dec rule), **5** (Q_{12}), **4** (Q_{21}), and **6** (Q_{22}) are consistent. All these checks are successful. In the process, we can do even more: perform the consistency check concurrently for **3** and **5**, and for **4** and **6** (as these pairs come from the same AUTs), and use the same new function symbol in each pair when applying the Red rule. (For instance, we can use h for **3** and **5** as it appears in both nodes.). Repeatedly apply this concurrent check to the children of the involved nodes in the process of showing consistency. Memoize common function symbols as they will be useful in the subsequent applications of Dec. After applying this process as long as possible, in the cash we will have:[3]

3 and **5** are consistent and have the common symbol h,

4 and **6** are consistent and have the common symbol h,

7 and **9** are consistent and have the common symbol h,

8 and **10** are consistent and have the common symbol h,

11 $\cup$ **13** and **12** are consistent and have the common symbol b,

11 and **12** $\cup$ **13** are consistent and have the common symbol b.

It is important to notice that if the consistency check failed for at least one node, e.g., for **9**, then the condition of Dec would fail and this rule would not be applicable for **1** and **2** using f. Then we should try h. If the same thing happens for h as well, then Dec is not applicable to **1** and **2** at all and we have to use Sol. Another important thing is to see what would happen if a pair of consistent nodes did not have a common symbol: for instance, if **5** and **6** are consistent but do not have a common symbol. In this case, we would cash this info and would not continue to check consistency of the successors of these nodes, i.e., we would not check whether **8** and **10**, and **11** and **12** $\cup$ **13** are consistent.

Coming back to the derivation, we have a new configuration

$$\{y_1 : \{f(f(a,c), f(a,c))\} \triangleq \{g(g(b))\}, y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};$$
$$\varnothing;\ f(y_1,y_2);\ 1;\ 1.$$

We select the first AUT and apply Dec. It should check whether nodes **3** and **5** have a common symbol. But we already did it in the consistency check and cashed the value. It is h. (If the cashed result told us there is no such common symbol, we would use Sol instead of Dec.) Subsequently, since in **3** the h comes with $\rho = \{(1,1),(1,2)\}$, we need to check whether the set of the first and second successors of **3** is consistent (the set Q_{11} in Dec). As one can see from the shared representation of the graph, this set is just **7**. We already know that it is consistent, because we checked its consistency when we showed that **3** is consistent. Similarly, **9** is consistent (the set Q_{12} in Dec). As for Q_{21} and Q_{22},

[3] By "**3** is consistent" we actually mean "the set of terms at node **3** is consistent", etc. Consistency of **11** $\cup$ **13** means that the union of term sets at **11** and at **13** is consistent.

they both are empty because the second argument of h does not appear in the $\mathsf{\rho}$'s at **3** and at **5**. Therefore, the new configuration is

$$\begin{aligned}&\{z_1 : \{f(a,c)\} \triangleq \{g(b)\},\ z_2 : \emptyset \triangleq \emptyset,\ y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};\\ &\qquad \emptyset;\ f(h(z_1, z_2), y_2);\ 0.8;\ 0.9.\end{aligned}$$

By the Tri rule, we can remove z_2:

$$\begin{aligned}&\{z_1 : \{f(a,c)\} \triangleq \{g(b)\},\ y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};\\ &\qquad \emptyset;\ f(h(z_1, \_), y_2);\ 0.8;\ 0.9.\end{aligned}$$

Now we apply Dec to the first AUT. It should check whether the nodes that correspond to the terms in this AUT (i.e., **7** and **9**) have a common symbol. But again, we can retrieve it from the cash. It is h. Based on the $\mathsf{\rho}$'s of h in these nodes, we need to check whether the set of the first and second successors of **7**, i.e., $\mathbf{11} \cup \mathbf{13}$, is consistent (the set Q_{11} in Dec), and the successor of **9**, i.e., **12**, is consistent (the set Q_{12} in Dec). We again reuse the cashed info that we got when we checked the consistency of **3**. Hence, the new configuration is

$$\begin{aligned}&\{u_1 : \{a,c\} \triangleq \{b\},\ u_2 : \emptyset \triangleq \emptyset,\ y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};\\ &\qquad \emptyset;\ f(h(h(u_1, u_2), \_), y_2);\ 0.8;\ 0.9.\end{aligned}$$

By the Tri rule, we can remove u_2:

$$\begin{aligned}&\{u_1 : \{a,c\} \triangleq \{b\},\ y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};\\ &\qquad \emptyset;\ f(h(h(u_1, \_), \_), y_2);\ 0.8;\ 0.9.\end{aligned}$$

Applying Dec to the first AUT, we check what is the common symbol between the nodes that correspond the terms there: $\mathbf{11} \cup \mathbf{13}$ and **12**. The cashed result tells us that it is b. No further consistency checks are needed because of the empty $\mathsf{\rho}$ it has. We get

$$\{y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};\ \emptyset;\ f(h(h(b, \_), \_), y_2);\ 0.6;\ 0.9$$

and continue in the similar manner:

$$\begin{aligned}&\{y_2 : \{g(g(a))\} \triangleq \{f(f(b,c), f(b,c))\}\};\ \emptyset;\ f(h(h(b, \_), \_), y_2);\ 0.6;\ 0.9 \Longrightarrow_{\mathsf{Dec}}\\ &\{v_1 : \{g(a)\} \triangleq \{f(b,c)\},\ v_2 : \emptyset \triangleq \emptyset\};\ \emptyset;\\ &\qquad f(h(h(b, \_), \_), h(v_1, v_2));\ 0.6;\ 0.8 \Longrightarrow_{\mathsf{Tri}}\\ &\{v_1 : \{g(a)\} \triangleq \{f(b,c)\}\};\ \emptyset;\ f(h(h(b, \_), \_), h(v_1, \_));\ 0.6;\ 0.8 \Longrightarrow_{\mathsf{Dec}}\\ &\{w_1 : \{a\} \triangleq \{b,c\},\ w_2 : \emptyset \triangleq \emptyset\};\ \emptyset;\\ &\qquad f(h(h(b, \_), \_), h(h(w_1, w_2), \_));\ 0.6;\ 0.8 \Longrightarrow_{\mathsf{Tri}}\\ &\{w_1 : \{a\} \triangleq \{b,c\}\};\ \emptyset;\ f(h(h(b, \_), \_), h(h(w_1, \_), \_));\ 0.6;\ 0.7 \Longrightarrow_{\mathsf{Dec}}\\ &\emptyset;\ \emptyset;\ f(h(h(b, \_), \_), h(h(b, \_), \_));\ 0.6;\ 0.7.\end{aligned}$$

This is the first terminal configuration. Remember that in the first decomposition step, we had an alternative in choosing h instead of f. Exploring it, we start with the step:

$$\begin{aligned}&\{x : \{f(f(f(a,c), f(a,c)), g(g(a)))\} \triangleq \{f(g(g(b)), f(f(b,c), f(b,c)))\}\};\\ &\qquad \emptyset;\ x;\ 1;\ 1 \Longrightarrow_{\mathsf{Dec}}\\ &\{y_1 : \{f(f(a,c), f(a,c)), g(g(a))\} \triangleq \{g(g(b)), f(f(b,c), f(b,c))\},\\ &\qquad y_2 : \emptyset \triangleq \emptyset\}; \emptyset;\ h(y_1, y_2);\ 0.8;\ 0.8.\end{aligned}$$

(To perform this step, we had to make sure that both $\mathbf{3} \cup \mathbf{4}$ and $\mathbf{5} \cup \mathbf{6}$ are $(\mathcal{R}, \lambda)$-consistent.) Continuing further, we reach the next terminal configuration:

$$\begin{aligned}&\{y_1 : \{f(f(a,c), f(a,c)), g(g(a))\} \triangleq \{g(g(b)), f(f(b,c), f(b,c))\},\\ &\qquad y_2 : \emptyset \triangleq \emptyset\}; \emptyset;\ h(y_1, y_2);\ 0.8;\ 0.8 \Longrightarrow^{*}\\ &\emptyset;\ \emptyset;\ h(h(h(b, \_), \_), \_);\ 0.6;\ 0.7.\end{aligned}$$

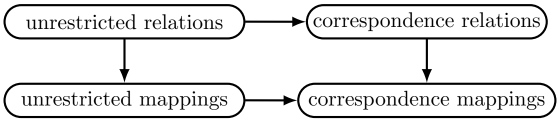
Hence, we got two answers computed by $\mathfrak{A}_1^{\mathsf{lin}}$:

$$f(h(h(b, \_), \_), h(h(b, \_))); 0.6; 0.7, \qquad\qquad h(h(h(b, \_), \_), \_), 0.6, 0.7.$$

$\mathfrak{A}_1$ would give the same answers, since the store is empty: no merging is needed.

7 Discussion and conclusion

The diagram below illustrates the connections between different anti-unification problems based on argument relations:



The arrows indicate the direction from more general problems to more specific ones. For the unrestricted cases (left column) we compute mcsrg's. For correspondence relations and correspondence mappings (right column), mcsg's are computed. (In fact, for them, the notions of mcsrg and mcsg coincide). The algorithms for relations (upper row) are more involved than those for mappings (lower row): Those for relations deal with AUTs containing arbitrary sets of terms, while for mappings, those sets have cardinality at most one, thus simplifying the conditions in the rules. Moreover, the two cases in the lower row generalize the existing anti-unification problems:

- the unrestricted mappings case generalizes the problem from [1] by extending similarity to proximity and relaxing the smaller-side-totality restriction;
- the correspondence mappings case generalizes the problem from [8] by allowing permutations between arguments of proximal function symbols.

All our algorithms can be easily turned into anti-unification algorithms for crisp tolerance relations[4] by taking lambda-cuts and ignoring the computation of the approximation degrees. Besides, they are modular and can be used to compute only linear generalizations by just skipping the merging rule. We provided complexity estimations for the algorithms that compute linear generalizations (that often are of practical interest).

In this paper, we did not consider cases when the same pair of symbols is related to each other by more than one argument relation. Our results can be extended to them, that would open a way towards approximate anti-unification modulo background theories specified by shallow collapse-free axioms. Another interesting direction of future work would be extending our results to quantitative algebras [9] that also deal with quantitative extensions of equality.

Acknowledgment. Supported by the Austrian Science Fund, project P 35530.

References

1. Aït-Kaci, H., Pasi, G.: Fuzzy lattice operations on first-order terms over signatures with similar constructors: A constraint-based approach. Fuzzy Sets Syst. **391**, 1–46 (2020). https://doi.org/10.1016/j.fss.2019.03.019
2. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1998)
3. Bader, J., Scott, A., Pradel, M., Chandra, S.: Getafix: learning to fix bugs automatically. Proc. ACM Program. Lang. **3**(OOPSLA), 159:1–159:27 (2019). https://doi.org/10.1145/3360585
4. Barwell, A.D., Brown, C., Hammond, K.: Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. Future Gener. Comput. Syst. **79**, 669–686 (2018). https://doi.org/10.1016/j.future.2017.07.024
5. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22**(8), 465–476 (1979). https://doi.org/10.1145/359138.359142
6. Galitsky, B.: Developing Enterprise Chatbots - Learning Linguistic Structures. Springer (2019). https://doi.org/10.1007/978-3-030-04299-8
7. Kirbas, S., Windels, E., McBello, O., Kells, K., Pagano, M.W., Szalanski, R., Nowack, V., Winter, E.R., Counsell, S., Bowes, D., Hall, T., Haraldsson, S., Woodward, J.R.: On the introduction of automatic program repair in Bloomberg. IEEE Softw. **38**(4), 43–51 (2021). https://doi.org/10.1109/MS.2021.3071086
8. Kutsia, T., Pau, C.: Matching and generalization modulo proximity and tolerance relations. In: Özgün, A., Zinova, Y. (eds.) Language, Logic, and Computation - 13th International Tbilisi Symposium, TbiLLC 2019, Batumi, Georgia, September 16-20, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13206, pp. 323–342. Springer (2019). https://doi.org/10.1007/978-3-030-98479-3_16
9. Mardare, R., Panangaden, P., Plotkin, G.D.: Quantitative algebraic reasoning. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proc. of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16. pp. 700–709. ACM (2016). https://doi.org/10.1145/2933575.2934518

[4] Tolerance: reflexive, symmetric, not necessarily transitive relation. According to Poincaré, a fundamental notion for mathematics applied to the physical world.

10. Mehta, S., Bhagwan, R., Kumar, R., Bansal, C., Maddila, C.S., Ashok, B., Asthana, S., Bird, C., Kumar, A.: Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In: Bhagwan, R., Porter, G. (eds.) 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020. pp. 435–448. USENIX Association (2020), `https://www.usenix.org/conference/nsdi20/presentation/mehta`
11. Pau, C., Kutsia, T.: Proximity-based unification and matching for fully fuzzy signatures. In: 30th IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2021, Luxembourg, July 11-14, 2021. pp. 1–6. IEEE (2021). https://doi.org/10.1109/FUZZ45933.2021.9494438
12. Plotkin, G.D.: A note on inductive generalization. Machine Intel. **5**(1), 153–163 (1970)
13. Raza, M., Gulwani, S., Milic-Frayling, N.: Programming by example using least general generalizations. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada. pp. 283–290. AAAI Press (2014)
14. Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. Machine Intel. **5**(1), 135–151 (1970)
15. Rolim, R., Soares, G., Gheyi, R., D'Antoni, L.: Learning quick fixes from code repositories. CoRR **abs/1803.03806** (2018), `http://arxiv.org/abs/1803.03806`