Solving Proximity Constraints[⋆]

Temur Kutsia and Cleo Pau

Johannes Kepler University Linz
Research Institute for Symbolic Computation
Linz, Austria

Abstract. Proximity relations are binary fuzzy relations that satisfy reflexivity and symmetry properties, but are not transitive. They induce the notion of distance between function symbols, which is further extended to terms. Given two terms, we aim at bringing them "sufficiently close" to each other, by finding an appropriate substitution. We impose no extra restrictions on proximity relations, allowing a term in unification to be close to two terms that themselves are not close to each other. Our unification algorithm works in two phases: first reducing the equation solving problem to constraints over sets of function symbols, and then solving the obtained constraints. Termination, soundness and completeness of both algorithms are shown. The unification problem has finite minimal complete set of unifiers.

1 Introduction

Proximity relations are reflexive and symmetric fuzzy binary relations. Introduced in [2], they generalize similarity relations (a fuzzy version of equivalence), by dropping transitivity. Proximity relations help to represent fuzzy information in situations, where similarity is not adequate, providing more flexibility in expressing vague knowledge. For unification, working modulo proximity or similarity means to treat different function symbols as if they were the same when the given relation asserts they are "close enough" to each other.

Unification for similarity relations was studied in [3, 4, 11] in the context of fuzzy logic programming. In [1], the authors extended the algorithm from [11] to full fuzzy signature (permitting arity mismatches between function symbols) and studied also anti-unification, a dual technique to unification.

A constraint logic programming schema with proximity relations has been introduced in [10]. The similarity-based unification algorithm from [11] was generalized for proximity relations in [5], where the authors introduced the notion of proximity-based unification under a certain restriction imposed on the proximity relation. The restriction requires that the same symbol can not be close to two symbols at the same time, when those symbols are not close to each other. One of them should be chosen as the proximal candidate to the given symbol. Essentially, proximal neighborhoods of two function symbols are treated as being

⋆ Supported by Austrian Science Fund (FWF) under project 28789-N32 and by the strategic program "Innovatives OÖ 2020" by the Upper Austrian Government.

either identical or disjoint. Looking at the proximity relation as an undirected graph, this restriction implies that maximal cliques in the graph are disjoint. From the unification point of view, it means that $p(x, x)$ can not be unified with $p(a, c)$ when a and c are not close to each other, even if there exists a b which is close both to a and c. Anti-unification for such kind of proximity relations, and its subalgorithm for computing all maximal partitions of a graph into maximal cliques have been considered in [7, 8].

In this paper we consider the general case of unification for a proximity relation without any restrictions, and develop an algorithm which computes a compact representation of the set of solutions. Considering neighborhoods of function symbols as finite sets, we work with term representation where in place of function symbols we permit neighborhoods or names. The latter are some kind of variables, which stand for unknown neighborhoods. The algorithm is split into two phases. In the first one, which is a generalization of syntactic unification for proximity relations, we produce a substitution together with a constraint over neighborhoods and their names. A crucial step in the algorithm is variable elimination, which is done not with a term to which a variable should be unified, but with a copy of that term with fresh names and variables. This step also introduces new neighborhood constraints to ensure that the copy of the term remains close to its original.

In the second phase, the constraint is solved by a constraint solving algorithm. Combining each solution from the second phase with the substitution computed in the first phase, we obtain a compact representation of the minimal complete set of unifiers of the original problem. We prove termination, soundness and completeness of both algorithms. To the best of our knowledge, this is the first detailed study of full-scale proximity-based unification.

In the rest of the paper, the necessary notions and terminology are introduced in Sect. 2, and both algorithms are developed and studied in Sect. 3. Some proofs, which are only sketched here, can be found in the technical report [9].

2 Preliminaries

Proximity relations. We define basic notions about proximity relations following [5]. A binary *fuzzy relation* on a set S is a mapping from $S \times S$ to the real interval $[0, 1]$. If $\mathcal{R}$ is a fuzzy relation on S and λ is a number $0 \leq \lambda \leq 1$, then the λ-*cut* of $\mathcal{R}$ on S, denoted $\mathcal{R}_\lambda$, is an ordinary (crisp) relation on S defined as $\mathcal{R}_\lambda := \{(s_1, s_2) \mid \mathcal{R}(s_1, s_2) \geq \lambda\}$. In the role of T-norm $\wedge$ we take the minimum.

A fuzzy relation $\mathcal{R}$ on a set S is called a *proximity relation* on S iff it is reflexive and symmetric. The λ-*proximity class of* $s \in S$ (a λ-*neighborhood* of s) is a set $\mathbf{pc}(s, \mathcal{R}, \lambda) = \{s' \mid \mathcal{R}(s, s') \geq \lambda\}$.

Terms and substitutions. Given a set of variables $\mathcal{V}$ and a set of fixed arity function symbols $\mathcal{F}$, *terms* over $\mathcal{F}$ and $\mathcal{V}$ are defined as usual, by the grammar $t := x \mid f(t_1, \ldots, t_n)$, where $x \in \mathcal{V}$ and $f \in \mathcal{F}$ is n-ary. The set of terms over $\mathcal{V}$ and $\mathcal{F}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We denote variables by x, y, z, arbitrary function symbols by f, g, h, constants by a, b, c, and terms by s, t, r.

Substitutions are mappings from variables to terms, where all but finitely many variables are mapped to themselves. The identity substitution is denoted by *Id*. We use the usual set notation for substitutions, writing, e.g., a substitution σ as $\{x \mapsto \sigma(x) \mid x \neq \sigma(x)\}$. Substitution application and composition are defined in the standard way. We use the postfix notation for substitution application, e.g., $t\sigma$, and juxtaposition for composition, e.g., $\sigma\vartheta$ means the composition of σ and ϑ (the order matters).

Extended terms, extended substitutions, name-neighborhood mappings. Assume that $\mathcal{N}$ is a countable *set of names*, which are symbols together with associated arity (like function symbols). We use the letters $\mathrm{N}, \mathrm{M}, \mathrm{K}$ for them. It is assumed that $\mathcal{N} \cap \mathcal{F} = \emptyset$ and $\mathcal{N} \cap \mathcal{V} = \emptyset$.

Neighborhood is either a name, or a finite subset of $\mathcal{F}$, where all elements have the same arity. Since in the construction of extended terms below neighborhoods will behave like function symbols, we will use the letters F and G to denote them. $\mathit{arity}(\mathsf{F})$ is defined as the arity of elements of F. The set of all neighborhoods is denoted by Nb.

An *extended term* (or, shortly, an *X-term*) t over $\mathcal{F}$, $\mathcal{N}$, and $\mathcal{V}$ is defined by the grammar:

$$\mathsf{t} := x \mid \mathsf{F}(\mathsf{t}_1, \dots, \mathsf{t}_n), \text{ where } \mathit{arity}(\mathsf{F}) = n.$$

The set of X-terms is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{N}, \mathcal{V})$. X-terms, in which every neighborhood set is a singleton, are called *singleton X-terms* or, shortly, *SX-terms*. Slightly abusing the notation, we assume that a term (i.e., an element of $\mathcal{T}(\mathcal{F}, \mathcal{V})$) is a special case of an SX-term (as an SX-term without names), identifying a function symbol f with the singleton neighborhood $\{f\}$. We will use this assumption in the rest of the paper.

The notion of *head* is defined as $\mathit{head}(x) = x$ and $\mathit{head}(\mathsf{F}(\mathsf{t}_1, \dots, \mathsf{t}_n)) = \mathsf{F}$.

The set of variables (resp. names) occurring in an X-term t is denoted by $\mathcal{V}(\mathsf{t})$ (resp. $\mathcal{N}(\mathsf{t})$). Approximate extended equations (X-equations) are pairs of X-terms.

The notion of substitution (and the associated relations and operations) are extended to X-terms straightforwardly. We use the term "*X-substitution*" and denote them by upright Greek letters $\upmu$, $\upnu$, and $\upxi$. When we want to emphasize that we are talking about substitutions for terms, we use the letters σ, ϑ, and φ.

The restriction of an X-substitution $\upmu$ to a set of variables V is denoted by $\upmu|_V := \{x \mapsto x\upmu \mid x \in \mathit{dom}(\upmu) \cap V\}$.

A *name-neighborhood mapping* $\Phi : \mathcal{N} \longrightarrow \mathsf{Nb} \setminus \mathcal{N}$ is a finite mapping from names to non-name neighborhoods (i.e., finite sets of function symbols of the same arity) such that if $\mathrm{N} \in \mathit{dom}(\Phi)$ (where *dom* is the domain of mapping), then $\mathit{arity}(\mathrm{N}) = \mathit{arity}(\Phi(\mathrm{N}))$. They are also represented as finite sets, writing Φ as $\{\mathrm{N} \mapsto \Phi(\mathrm{N}) \mid \mathrm{N} \in \mathit{dom}(\Phi)\}$.

A name-neighborhood mapping Φ can *apply* to an X-term t, resulting in an X-term $\Phi(\mathsf{t})$, which is obtained by replacing each name N in t by the neighborhood $\Phi(\mathrm{N})$. The *application of* Φ *to a set of X-equations* P, denoted by $\Phi(P)$, is a set of equations obtained from P by applying Φ to both sides of each equation in P.

Proximity relations over X-terms. The proximity relation $\mathcal{R}$ is defined on the set $\mathsf{Nb} \cup \mathcal{V}$ (where neighborhoods are assumed to be nonempty) in such a way that it satisfies the following conditions (in addition to reflexivity and symmetry):

(a) $\mathcal{R}(\mathsf{F},\mathsf{G}) = 0$ if $\mathit{arity}(\mathsf{F}) \neq \mathit{arity}(\mathsf{G})$;
(b) $\mathcal{R}(\mathsf{F},\mathsf{G}) = \min\{\mathcal{R}(f,g) \mid f \in \mathsf{F}, g \in \mathsf{G}\}$, if $\mathsf{F} = \{f_1,\ldots,f_n\}$, $\mathsf{G} = \{g_1,\ldots,g_m\}$, and $\mathit{arity}(\mathsf{F}) = \mathit{arity}(\mathsf{G})$;
(c) $\mathcal{R}(\mathrm{N},\mathsf{F}) = 0$, if $\mathsf{F} \notin \mathcal{N}$.
(d) $\mathcal{R}(\mathrm{N},\mathrm{M}) = 0$, if $\mathrm{N} \neq \mathrm{M}$;
(e) $\mathcal{R}(x,y) = 0$, if $x \neq y$ for all $x,y \in \mathcal{V}$.
(f) $\mathcal{R}(\mathsf{F},\mathsf{G})$ is undefined, if $\mathsf{F} = \emptyset$ or $\mathsf{G} = \emptyset$.

We write $\mathsf{F} \approx_{\mathcal{R},\lambda} \mathsf{G}$ if $\mathcal{R}(\mathsf{F},\mathsf{G}) \geq \lambda$. Note that for $\mathsf{F} = \{f_1,\ldots,f_n\}$ and $\mathsf{G} = \{g_1,\ldots,g_m\}$, $\mathsf{F} \approx_{\mathcal{R},\lambda} \mathsf{G}$ is equivalent to $\mathcal{R}(f,g) \geq \lambda$ for all $f \in \mathsf{F}$ and $g \in \mathsf{G}$. It is easy to see that the obtained relation is again a proximity relation. Furthermore, it can be extended to X-terms (which do not contain the empty neighborhood):

1. $\mathcal{R}(\mathsf{s},\mathsf{t}) := 0$ if $\mathcal{R}(\mathit{head}(\mathsf{s}),\mathit{head}(\mathsf{t})) = 0$.
2. $\mathcal{R}(\mathsf{s},\mathsf{t}) := 1$ if $\mathsf{s} = \mathsf{t}$ and $\mathsf{s},\mathsf{t} \in \mathcal{V}$.
3. $\mathcal{R}(\mathsf{s},\mathsf{t}) := \mathcal{R}(\mathsf{F},\mathsf{G}) \wedge \mathcal{R}(\mathsf{s}_1,\mathsf{t}_1) \wedge \cdots \wedge \mathcal{R}(\mathsf{s}_n,\mathsf{t}_n)$, if $\mathsf{s} = \mathsf{F}(\mathsf{s}_1,\ldots,\mathsf{s}_n)$, $\mathsf{t} = \mathsf{G}(\mathsf{t}_1,\ldots,\mathsf{t}_n)$.
4. $\mathcal{R}(\mathsf{s},\mathsf{t})$ is not defined, if s or t contains the empty neighborhood $\emptyset$.

Two X-terms s and t are $(\mathcal{R},\lambda)$-*close* to each other, written $\mathsf{s} \simeq_{\mathcal{R},\lambda} \mathsf{t}$, if $\mathcal{R}(\mathsf{s},\mathsf{t}) \geq \lambda$. We say that s is $(\mathcal{R},\lambda)$-*more general than* t and write $\mathsf{s} \precsim_{\mathcal{R},\lambda} \mathsf{t}$, if there exists a substitution σ such that $\mathsf{s}\sigma \simeq_{\mathcal{R},\lambda} \mathsf{t}$.

Neighborhood equations, unification problems. We introduce the notions of problems we would like to solve.

Definition 1 (Neighborhood equations). *Given $\mathcal{R}$ and λ, an $(\mathcal{R},\lambda)$-*neighborhood equation *is a pair of neighborhoods, written as $\mathsf{F} \approx^?_{\mathcal{R},\lambda} \mathsf{G}$. The question mark indicates that it has to be solved.*

A name-neighborhood mapping Φ is a solution *of an $(\mathcal{R},\lambda)$-neighborhood equation $\mathsf{F} \approx^?_{\mathcal{R},\lambda} \mathsf{G}$ if $\Phi(\mathsf{F}) \simeq_{\mathcal{R},\lambda} \Phi(\mathsf{G})$. The notation implies that $\mathcal{R}(\Phi(\mathsf{F}),\Phi(\mathsf{G}))$ is defined, i.e., neither $\Phi(\mathsf{F})$ nor $\Phi(\mathsf{G})$ contains the empty neighborhood.*

An $(\mathcal{R},\lambda)$-neighborhood constraint is a finite set of $(\mathcal{R},\lambda)$-neighborhood equations. A name-neighborhood mapping Φ is a solution of an $(\mathcal{R},\lambda)$-neighborhood constraint C if it is a solution of every $(\mathcal{R},\lambda)$-neighborhood equation in C.

We shortly write "an $(\mathcal{R},\lambda)$-solution to C" instead of "a solution to an $(\mathcal{R},\lambda)$-neighborhood constraint C".

To each X-term t we associate a set of SX-terms $\mathsf{Singl}(\mathsf{t})$ defined as follows:

$$\mathsf{Singl}(x) := \{x\},$$
$$\mathsf{Singl}(\mathrm{N}(\mathsf{t}_1,\ldots,\mathsf{t}_n)) := \{\mathrm{N}(\mathsf{s}_1,\ldots,\mathsf{s}_n) \mid \mathsf{s}_i \in \mathsf{Singl}(\mathsf{t}_i),\ 1 \leq i \leq n\}.$$
$$\mathsf{Singl}(\mathsf{F}(\mathsf{t}_1,\ldots,\mathsf{t}_n)) := \{f(\mathsf{s}_1,\ldots,\mathsf{s}_n) \mid f \in \mathsf{F},\ \mathsf{s}_i \in \mathsf{Singl}(\mathsf{t}_i),\ 1 \leq i \leq n\}.$$

The notation extends to substitutions as well:

$$\mathsf{Singl}(\upmu) := \{\vartheta \mid x\vartheta \in \mathsf{Singl}(x\upmu) \text{ for all } x \in \mathcal{V}\}.$$

Definition 2 (Approximate X-unification). *Given $\mathcal{R}$ and λ, a finite set P of $(\mathcal{R},\lambda)$-equations between X-terms is called an $(\mathcal{R},\lambda)$-X-unification problem. A mapping-substitution pair (Φ,μ) is called an $(\mathcal{R},\lambda)$-*solution *of an $(\mathcal{R},\lambda)$-X-equation $\mathsf{t} \simeq^{?}_{\mathcal{R},\lambda} \mathsf{s}$, if $\Phi(\mathsf{t}\mu) \simeq_{\mathcal{R},\lambda} \Phi(\mathsf{s}\mu)$. An $(\mathcal{R},\lambda)$-solution of P is a pair (Φ,μ) which solves each equation in P.*

If (Φ,μ) is an $(\mathcal{R},\lambda)$-solution of P, then the X-substitution $\Phi(\mu)$ is called an $(\mathcal{R},\lambda)$-X-unifier *of P.*

SX-unification problems, SX-solutions and SX-unifiers are defined analogously. For unification between terms, we do not use any prefix, talking about unification problems, solutions, and unifiers.

Instead of writing "a $\cdots$-unifier of an $(\mathcal{R},\lambda)$-unification problem P", we often shortly say "an $(\mathcal{R},\lambda)$-$\cdots$-unifier of P".

The notion of more generality for substitutions is defined with the help of syntactic equality: μ is *more general* than ν, written $\mu \preceq \nu$, if there exists ξ, such that $\mu\xi = \nu$. In this case, ν is called an *instance* of μ.

Remark 1. Note that we did not use proximity in the definition of this notion. The reason is that in our definition, $\preceq$ is a quasi-order and preserves good properties of unifiers. In particular, if μ is an $(\mathcal{R},\lambda)$-X-unifier of P, then so is any ν for which $\mu \preceq \nu$ holds.

If we defined this notion as "$\mu \precsim_{\mathcal{R},\lambda} \nu$ if there exists a substitution ξ, such that $x\mu\nu \simeq_{\mathcal{R},\lambda} x\xi$, for all x", it would not be a quasi-order, because it is not transitive. Therefore, it might happen that μ is an $(\mathcal{R},\lambda)$-X-unifier of P, but ν with $\mu \precsim_{\mathcal{R},\lambda} \nu$ is not. A simple example is $P = \{x \simeq^{?}_{\mathcal{R},\lambda} a\}$ and $\mathcal{R}_\lambda = \{(a,b),(b,c)\}$. Then $\mu = \{x \mapsto b\}$ is an $(\mathcal{R},\lambda)$-unifier of P, but $\nu = \{x \mapsto c\}$ is not. However, $\mu \precsim_{\mathcal{R},\lambda} \nu$.

Two substitutions μ and ν are called *equigeneral* iff $\mu \preceq \nu$ and $\nu \preceq \mu$. In this case we write $\mu \simeq \nu$. It is an equivalence relation.

Unification between terms. Our unification problem will be formulated between terms, and we would like to have a characterization of the set of its unifiers. (Extended terms and substitutions will play a role in the formulating of algorithms, proving their properties, and representing the mentioned unifier set compactly.)

Definition 3 (Complete set of unifiers). *Given a proximity relation $\mathcal{R}$, a cut value λ, and an $(\mathcal{R},\lambda)$-proximity unification problem P, the set of substitutions Σ is a* complete set of $(\mathcal{R},\lambda)$-unifiers *of P if the following conditions hold:*

Soundness: *Every substitution $\sigma \in \Sigma$ is an $(\mathcal{R},\lambda)$-unifier of P.*
Completeness: *For any $(\mathcal{R},\lambda)$-unifier ϑ of P, there exists $\sigma \in \Sigma$ such that $\sigma \preceq \vartheta$.*

Σ is *a minimal complete set of unifiers* of P if it is its complete set of unifiers and, in addition, the following condition holds:

Minimality: No two elements in Σ are comparable with respect to $\preceq$: For all $\sigma,\vartheta \in \Sigma$, if $\sigma \preceq \vartheta$, then $\sigma = \vartheta$.

Under this definition, $\{x \simeq_{\mathcal{R},0.5} b\}$ for $\mathcal{R}(a,b) = 0.6$, $\mathcal{R}(b,c) = 0.5$ has a minimal complete set of unifiers $\{\{x \mapsto a\}, \{x \mapsto b\}, \{x \mapsto c\}\}$. Note that the substitutions $\{x \mapsto a\}$ and $\{x \mapsto b\}$ are $\preceq$-incomparable, but $\precsim_{\mathcal{R},\lambda}$-comparable. The same is true for $\{x \mapsto a\}$ and $\{x \mapsto c\}$.

Given an approximate unification problem P, our goal is to obtain a compact representation of its minimal complete set of $(\mathcal{R}, \lambda)$-unifiers. The representation will be constructed as a set of X-unifiers $\mathcal{U}^X_{\mathcal{R},\lambda}(P) = \{\Phi_1(\mu), \dots, \Phi_n(\mu)\}$. The algorithms below construct this representation.

3 Solving unification problems

We start with a high-level view of the process of solving an approximate unification problem $s \simeq^?_{\mathcal{R},\lambda} t$ between terms s and t (we omit $\mathcal{R}$ and λ below):

– First, we treat the input equation as an SX-equation and apply rules of the pre-unification algorithm. Pre-unification works on SX-equations. It either fails (in this case the input terms are not unifiable) or results in a neighborhood constraint C and a substitution μ over $\mathcal{T}(\emptyset, \mathcal{N}, \mathcal{V})$.
– Next, we solve C by the neighborhood constraint solving algorithm. If the process fails, then the input terms are not unifiable. Otherwise, we get a finite set of name-neighborhood mappings $\mathcal{M} = \{\Phi_1, \dots, \Phi_n\}$. Note that Φ's do not necessarily map names to singleton sets here.
– For each $\Phi_i \in \mathcal{M}$, the pair (Φ_i, μ) solves the original unification problem, i.e., the X-substitution $\Phi_i(\mu)$ is an X-unifier of it.
– From the obtained set $\{\Phi_1(\mu), \dots, \Phi_n(\mu)\}$ of computed $(\mathcal{R}, \lambda)$-X-unifiers of s and t we can construct a minimal complete set of unifiers $mcsu_{\mathcal{R},\lambda}(s,t)$ of s and t as the set $mcsu_{\mathcal{R},\lambda}(s,t) = \mathsf{Singl}(\Phi(\mu_1)) \cup \dots \cup \mathsf{Singl}(\Phi(\mu_1))$.

Hence, the algorithm consists of two phases: pre-unification and constraint solving. They are described in separate subsections below.

3.1 Pre-unification rules

We start with the definition of a technical notion needed later:

Definition 4. *We say that a set of SX-equations $\{x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \uplus P$ contains an* occurrence cycle *for the variable x if $\mathsf{t} \notin \mathcal{V}$ and there exist SX-term-pairs $(x_0, \mathsf{t}_0), (x_1, \mathsf{t}_1), \dots, (x_n, \mathsf{t}_n)$ such that $x_0 = x$, $\mathsf{t}_0 = \mathsf{t}$, for each $0 \le i \le n$ P contains an equation $x_i \simeq^?_{\mathcal{R},\lambda} \mathsf{t}_i$ or $\mathsf{t}_i \simeq^?_{\mathcal{R},\lambda} x_i$, and $x_{i+1} \in \mathcal{V}(\mathsf{t}_i)$ where $x_{n+1} = x_0$.*

Lemma 1. *If a set of SX-equations P contains an occurrence cycle for some variable, then it has no $(\mathcal{R}, \lambda)$-solution for P for any $\mathcal{R}$ and λ.*

Proof. The requirement that neighborhoods of different arity are not $(\mathcal{R}, \lambda)$-close to each other guarantees that an SX-term can not be $(\mathcal{R}, \lambda)$-close to its proper subterm. Therefore, equations containing an occurrence cycle can not have an $(\mathcal{R}, \lambda)$-solution. □

In the rules below we will use the *renaming function* $\rho : \mathcal{T}(\mathcal{F},\mathcal{N},\mathcal{V}) \to \mathcal{T}(\mathcal{N},\mathcal{V})$. Applied to a term, ρ gives its fresh copy, obtained by replacing each occurrence of a symbol from $\mathcal{F} \cup \mathcal{N}$ by a new name and each variable occurrence by a fresh variable. For instance, if the term is $f(\mathrm{N}(a,x,x,f(a)))$, where $f, a \in \mathcal{F}$ and $\mathrm{N} \in \mathcal{N}$, then $\rho(f(\mathrm{N}(a,x,x,f(a))) = \mathrm{N}_1(\mathrm{N}_2(\mathrm{N}_3, x_1, x_2, \mathrm{N}_4(\mathrm{N}_5)))$, where $\mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_3, \mathrm{N}_4, \mathrm{N}_5 \in \mathcal{N}$ are new names and x_1, x_2 are new variables.

Given $\mathcal{R}$ and λ, an *equational* $(\mathcal{R},\lambda)$*-configuration* is a triple $P; C; \mu$, where

- P is a finite set of $(\mathcal{R},\lambda)$-SX-equations. It is initialized with the unification equation between the original terms;
- C is a $(\mathcal{R},\lambda)$-neighborhood constraint;
- μ is an X-substitution over $\mathcal{T}(\emptyset,\mathcal{N},\mathcal{V})$, initialized by Id. It serves as an accumulator, keeping the pre-unifier computed so far.

The pre-unification algorithm takes given terms s and t, creates the initial configuration $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id$ and applies the rules given below exhaustively.

The rules are very similar to the syntactic unification algorithm with the difference that here the function symbol clash does not happen unless their arities differ, and variables are not replaced by other variables until the very end. (The notation $\overline{exp_n}$ in the rules below abbreviates the sequence $exp_1, \dots, exp_n$.)

(Tri) Trivial: $\{x \simeq^?_{\mathcal{R},\lambda} x\} \uplus P;\ C;\ \mu \Longrightarrow P;\ C;\ \mu.$

(Dec) Decomposition:

$$\{\mathsf{F}(\overline{\mathsf{s}_n}) \simeq^?_{\mathcal{R},\lambda} \mathsf{G}(\overline{\mathsf{t}_n})\} \uplus P; C; \mu \Longrightarrow \{\overline{\mathsf{s}_n \simeq^?_{\mathcal{R},\lambda} \mathsf{t}_n}\} \cup P; \{\mathsf{F} \approx^?_{\mathcal{R},\lambda} \mathsf{G}\} \cup C; \mu,$$

where each of F and G is a name or a function symbol treated as a singleton neighborhood.

(VE) Variable Elimination:

$$\{x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \uplus P;\ C;\ \mu \Longrightarrow \{\mathsf{t}' \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \cup P\{x \mapsto \mathsf{t}'\};\ C;\ \mu\{x \mapsto \mathsf{t}'\},$$

where $\mathsf{t} \notin \mathcal{V}$, there is no occurrence cycle for x in $\{x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \uplus P$, and $\mathsf{t}' = \rho(\mathsf{t})$.

(Ori) Orient: $\{\mathsf{t} \simeq^?_{\mathcal{R},\lambda} x\} \uplus P;\ C;\ \mu \Longrightarrow \{x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \cup P;\ C;\ \mu,$ if $\mathsf{t} \notin \mathcal{V}$.

(Cla) Clash: $\mathsf{F}(\overline{\mathsf{s}_n}) \simeq^?_{\mathcal{R},\lambda} \mathsf{G}(\overline{\mathsf{t}_m})\} \uplus P;\ C;\ \mu \Longrightarrow \bot,$ where $n \neq m$.

(Occ) Occur Check: $\{x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \uplus P;\ C;\ \mu \Longrightarrow \bot,$

if there is an occurrence cycle for x in $\{x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}\} \uplus P$.

(VO) Variables Only:

$$\{x \simeq^?_{\mathcal{R},\lambda} y,\ \overline{x_n \simeq^?_{\mathcal{R},\lambda} y_n}\}; C; \mu \Longrightarrow \{\overline{x_n \simeq^?_{\mathcal{R},\lambda} y_n}\}\{x \mapsto y\}; C; \mu\{x \mapsto y\}.$$

Informally, in the (VE) rule, we imitate the structure of t in t' by ρ, replace x by t', and then try to bring t' close to t by solving the equation $\mathsf{t}' \simeq^?_{\mathcal{R},\lambda} \mathsf{t}$.

Theorem 1 (Termination of pre-unification). *The pre-unification algorithm terminates either with $\bot$ or with a configuration of the form $\emptyset; C; \mu$.*

Proof. The rules (Tri) and (Dec) strictly decrease the size of P. (Ori) does not changes the size, but strictly decreases the number of equations of the form $\mathsf{t} \simeq^?_{\mathcal{R},\lambda} x$, where $\mathsf{t} \notin \mathcal{V}$. (VO) stands separately, because once it starts applying, no other rule is applicable and (VO) itself is terminating. So are the failure rules.

To see what is decreased by (VE), we need some definitions. First, with each variable occurring in the initial unification problem we associate the set of its copies, which is initialized with the singleton set consisting of the variables themselves. For instance, if the problem contains variables x, y, z, and u, we will have four copy sets: $\{x\}$, $\{y\}$, $\{z\}$, and $\{u\}$. Rules may add new copies to these sets, or remove some copies from them. However, the copy sets themselves are fixed. None of them will be removed, and no new copy sets will be created.

In the process of rule applications, we will maintain a directed acyclic graph, whose vertices are labeled by copy sets, and there is an edge from a vertex V_1 to a vertex V_2 if we have encountered an equation of the form $x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}$ such that $x \in V_1$ and t contains a variable $y \in V_2$.

One can notice that the graph is a variable dependency graph. If it contains a cycle, the algorithm stops with failure by the (Occ) rule. From the beginning, the vertices (i.e. the copy sets) are isolated. In the process of rule applications, assume that we reach a configuration that is transformed by the (VE) rule, applied to an equation $x \simeq^?_{\mathcal{R},\lambda} \mathsf{t}$, where t contains variables y, z', and z'' (the latter two are copies of z). The rule creates a fresh copy of t, which contains copies of variables: $\rho(y)$, $\rho(z')$, and $\rho(z'')$. They are added to the corresponding copy sets (graph vertices): $\rho(y)$ to the copy set of y, and $\rho(z')$ and $\rho(z'')$ to the copy set of z. Let us call those vertices V_y and V_z. Besides, if there was no edge connecting the vertex V_x (containing the copy set of x) to the vertices V_y and V_z, the edges are created. Finally, x is removed from V_x.

Hence, after each application of the (VE) rule, the copy set decreases in one vertex V (in the example above it is V_x), and stays unchanged in all vertices that are not reachable from V. We say in this case that the graph measure decreases. This ordering can be seen as a generalization of lexicographic ordering to graphs. It is well-founded. (VE) strictly decreases it, and the other rules have no effect.

Hence, if we take the lexicographic combination of three measures: copy set dags, the size of the set of equations, and the number of equations with non-variable term in the left and variable in the right, each rule except (VO) strictly decrease it. After finitely many steps, either failure will occur, or one reaches the variable-only equations, which are solved in finitely many steps by (VO). Then it stops with the configuration $\emptyset; C; \mu$. $\square$

We say that a mapping-substitution pair (Φ, ν) is a *solution of an equational* $(\mathcal{R}, \lambda)$*-configuration* $P; C; \mu$ if the following conditions hold:

- (Φ, ν) is an $(\mathcal{R}, \lambda)$-solution of P;
- Φ is an $(\mathcal{R}, \lambda)$-solution of C;
- For each $x \in dom(\mu)$, we have $x\nu = x\mu\nu$ (syntactic equality).

Lemma 2. *1. If $P; C; \mu \Longrightarrow \bot$ by* (Cla) *or* (Occ) *rules, then $P; C; \mu$ does not have a solution.*

2. Let $P_1; C_1; \mu_1 \Longrightarrow P_2; C_2; \mu_2$ be a step performed by a pre-unification rule (except (Cla) *and* (Occ)*). Then every solution of $P_2; C_2; \mu_2$ is a solution of $P_1; C_1; \mu_1$.*

Proof. See [9]. □

Theorem 2 (Soundness of pre-unification). *Let s and t be two terms, such that the pre-unification algorithm gives $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id \Longrightarrow^* \emptyset; C; \mu$. Let Φ be an $(\mathcal{R},\lambda)$-solution of C. Then the X-substitution $\Phi(\mu)$ contains no names and is an $(\mathcal{R},\lambda)$-X-unifier of $\{s \simeq^?_{\mathcal{R},\lambda} t\}$.*

Proof. Note that (Φ, μ) is an $(\mathcal{R},\lambda)$-solution of $\emptyset; C; \mu$, since Φ solves C. From $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id \Longrightarrow^* \emptyset; C; \mu$, by induction on the length of the derivation, using Lemma 2, we get that (Φ, μ) is a solution of $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id$.

Since s and t do not contain names, Φ has no effect on them: $\Phi(s\mu) = s\Phi(\mu)$ and $\Phi(t\mu) = t\Phi(\mu)$. From these equalities and $\Phi(s\mu) \simeq_{\mathcal{R},\lambda} \Phi(t\mu)$ we get $s\Phi(\mu) \simeq_{\mathcal{R},\lambda} t\Phi(\mu)$, which implies that $\Phi(\mu)$ is an $(\mathcal{R},\lambda)$-X-solution of $\{s \simeq^?_{\mathcal{R},\lambda} t\}$. By construction of pre-unification derivations, all the names in μ are in the domain of Φ. Hence, $\Phi(\mu)$ contains no names. □

Corollary 1. *Let s and t be two terms, such that the pre-unification algorithm gives $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id \Longrightarrow^* \emptyset; C; \mu$. Let Φ be an $(\mathcal{R},\lambda)$-solution of C. Then every substitution in $\mathsf{Singl}(\Phi(\mu))$ is an $(\mathcal{R},\lambda)$-unifier of s and t.*

Proof. Direct consequence of Theorem 2 and the definition of Singl. Note that $\mathsf{Singl}(\Phi(\mu))$ in this case is a set of substitutions, not a set of SX-substitutions, because $\Phi(\mu)$ does not contain names. □

For any solution of an approximate unification problem, we can not always compute a solution which is more general than the given one. For instance, for the problem $x \simeq^?_{\mathcal{R},\lambda} y$ we compute $\{x \mapsto y\}$, but the problem might have a solution, e.g., $\{x \mapsto a, y \mapsto b\}$ (when a and b are close to each other). Strictly speaking, $\{x \mapsto y\}$ is not more general than $\{x \mapsto a, y \mapsto b\}$, because there is no substitution σ such that $\{x \mapsto y\}\sigma = \{x \mapsto a, y \mapsto b\}$, but we have the relation $\{x \mapsto a, y \mapsto b\} \in \mathsf{Singl}(\{x \mapsto y\}\{y \mapsto \{a,b\}\}) = \mathsf{Singl}(\{x \mapsto \{a,b\}, y \mapsto \{a,b\}\})$.

The Completeness Theorem below proves a more general statement. It states that for any solution of an approximate unification problem, we can always compute a solution which is more general than a solution close to the given one.

Theorem 3 (Completeness of pre-unification). *Let a substitution ϑ be an $(\mathcal{R},\lambda)$-unifier of two terms s and t. Then any maximal derivation that starts at $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id$ must end with an equational configuration $\emptyset; C; \mu$, such that for some $(\mathcal{R},\lambda)$-solution Φ of C, which maps names to singleton neighborhoods, and some substitution ν such that $\vartheta \simeq_{\mathcal{R},\lambda} \sigma$ for some $\sigma \in \mathsf{Singl}(\nu)$, we have $\Phi(\mu|_{\mathcal{V}(s)\cup\mathcal{V}(t)}) \preceq \nu$.*

Proof. The full proof can be found in [9]. Here we sketch the idea.

Since $\{s \simeq^?_{\mathcal{R},\lambda} t\}$ is solvable, the derivation can not end with $\perp$ by soundness of pre-unification. Since for every equation there is a rule, and the algorithm terminates, the final configuration should have a form $\emptyset; C; \mu$.

Let V be $\mathcal{V}(s) \cup \mathcal{V}(t)$. In the construction of Φ, we need the proposition:

Proposition: Assume $P_1; C_1; \mu_1 \Longrightarrow P_2; C_2; \mu_2$ is a single step rule application in the above mentioned derivation. Let Φ_1 map names occurring in μ_1 to singleton neighborhoods such that equations in $\Phi_1(P_1)$ and in $\Phi_1(C_1)$ remain solvable. Assume that there exists a substitution ν_1 such that $\vartheta \simeq_{\mathcal{R},\lambda} \sigma_1$ for some $\sigma_1 \in \mathsf{Singl}(\nu_1)$ and $\Phi_1(\mu_1|_V) \preceq \nu_1$. Then there exist a substitution ν_2 such that $\vartheta \simeq_{\mathcal{R},\lambda} \sigma_2$ for some $\sigma_2 \in \mathsf{Singl}(\nu_2)$, and a name-neighborhood mapping Φ_2 which maps names occurring in μ_2 to singleton neighborhoods such that equations in $\Phi_2(P_2)$ and in $\Phi_2(C_2)$ remain solvable, and $\Phi_2(\mu_2|_V) \preceq \nu_2$.

In the constructed derivation, for the initial configuration $P_0; C_0; \mu_0 = \{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id$ we take $\Phi_0 = \emptyset$. Then $\Phi_0(P_0) = P_0 = \{s \simeq^?_{\mathcal{R},\lambda} t\}$ and $\Phi_0(C_0) = \emptyset$ are solvable, and $\Phi_0(\mu_0|_V) = Id$, $\mathsf{Singl}(Id) = \{Id\}$, and $Id \preceq \vartheta$. By applying the proposition iteratively, we get that for the final configuration $\emptyset; C; \mu$, there exists Φ which maps names to singleton neighborhoods such that $\Phi(C)$ is solvable. By the way how Φ is constructed, we have $dom(\Phi) = \mathcal{N}(\mu)$, but $\mathcal{N}(\mu) = \mathcal{N}(C)$. Hence, $\mathcal{N}(\Phi(C)) = \emptyset$ and its solvability means that it is already solved (trivially solvable). It implies that Φ is a solution of C. Besides, again by an iterative application of the proposition, we show the existence of ν such that $\vartheta \simeq_{\mathcal{R},\lambda} \sigma$ for some $\sigma \in \mathsf{Singl}(\nu)$ and $\Phi(\mu|_V) \preceq \nu$. □

From Theorem 2 and Theorem 3, by definition of Singl we get

Theorem 4. *Given $\mathcal{R}$, λ and two terms s and t, let the pre-unification algorithm produce a derivation $\{s \simeq^?_{\mathcal{R},\lambda} t\}; \emptyset; Id \Longrightarrow^+ \emptyset; C; \mu$. Let $\{\Phi_1, \ldots, \Phi_n\}$ be a complete set of solutions of C, restricted to $\mathcal{N}(C)$. Then the set $\mathsf{Singl}(\Phi_1(\mu)) \cup \cdots \cup \mathsf{Singl}(\Phi_1(\mu))$ is a minimal complete set of unifiers of s and t.*

We should be careful in interpreting Theorem 4. If $\sigma \in mcsu_{\mathcal{R},\lambda}(s,t)$ and ξ is an X-substitution, then $\mathsf{Singl}(\sigma\xi)$ contains at least one $(\mathcal{R},\lambda)$-unifier of s and t, but it may contain non-unifiers as well. If we restrict ξ to be an SX-substitution, then all elements of $\mathsf{Singl}(\sigma\xi)$ are $(\mathcal{R},\lambda)$-unifiers of s and t, but SX-substitution instances are too weak to capture all unifiers. See [9, Example 1].

We introduce a neighborhood constraint solving algorithm in the next section. Before that we illustrate the pre-unification rules with a couple of examples:

Example 1. Let $s = p(x, y, x)$ and $t = q(f(a), g(d), y)$. Then the pre-unification algorithm gives $\emptyset; C, \mu$, where $C = \{p \approx^?_{\mathcal{R},\lambda} q,\ \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} f,\ \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} a,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} g,\ \mathrm{N}_4 \approx^?_{\mathcal{R},\lambda} d,\ \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} \mathrm{N}_3,\ \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} \mathrm{N}_4\}$ and $\mu = \{x \mapsto \mathrm{N}_1(\mathrm{N}_2), y \mapsto \mathrm{N}_3(\mathrm{N}_4)\}$.

Assume that for the given λ-cut, the proximity relation consists of pairs $\mathcal{R}_\lambda = \{(a,b), (b,c), (c,d), (a,b'), (b',c'), (c',d), (f,g), (p,q)\}$. The obtained constraint can be solved, e.g., by the name-neighborhood mappings $\Phi = [\mathrm{N}_1 \mapsto \{f,g\}, \mathrm{N}_2 \mapsto \{b\}, \mathrm{N}_3 \mapsto \{f,g\}, \mathrm{N}_4 \mapsto \{c\}]$ and $\Phi' = [\mathrm{N}_1 \mapsto \{f,g\}, \mathrm{N}_2 \mapsto$

$\{b'\}, \mathrm{N}_3 \mapsto \{f, g\}$, $\mathrm{N}_4 \mapsto \{c'\}]$. From them and μ we can get the sets $\Phi(\mu)$ and $\Phi'(\mu)$ of $(\mathcal{R}, \lambda)$-unifiers of s and t.

If we did not have the VO rule and allowed the use of VE rule instead, we might have ended up with the unification problem $\{y \simeq^?_{\mathcal{R},\lambda} f(a), y \simeq^?_{\mathcal{R},\lambda} g(d)\}$, which does not have a solution, because the neighborhoods of a and d do not have a common element. Hence, we would have lost a solution.

Example 2. Let $s = p(x, x)$ and $t = q(f(y, y), f(a, c))$. The pre-unification algorithm stops with $\emptyset; P; \mu$, where $P = \{p \approx^?_{\mathcal{R},\lambda} q,\ \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} f,\ \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} a,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} c,\ \mathrm{M} \approx^?_{\mathcal{R},\lambda} \mathrm{N}_2,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} \mathrm{M}\}$ and $\mu = \{x \mapsto \mathrm{N}_1(\mathrm{N}_2, \mathrm{N}_3),\ y_1 \mapsto \mathrm{N}_2,\ y_2 \mapsto \mathrm{N}_3,\ y \mapsto \mathrm{M}\}$. Let $\mathcal{R}_\lambda = \{(a, a_1), (a_1, b), (b, c_1), (c_1, c), (p, q)\}$. Then C is solved by $\Phi = [\mathrm{N}_1 \mapsto \{f\},\ \mathrm{N}_2 \mapsto \{a_1\},\ \mathrm{M} \mapsto \{b\},\ \mathrm{N}_3 \mapsto \{c_1\}]$ and $\Phi(\mu|_{\mathcal{V}(s)\cup\mathcal{V}(t)})$ contains only one element, an $(\mathcal{R}, \lambda)$-unifier $\sigma = \{x \mapsto f(a_1, c_1),\ y \mapsto b\}$ of s and t. Indeed, $s\sigma = p(f(a_1, c_1), f(a_1, c_1)) \simeq_{\mathcal{R},\lambda} q(f(b, b), f(a, c)) = t\sigma$.

This example illustrates the necessity of introducing a fresh variable for *each occurrence* of a variable by the renaming function in the VE rule. If we used the same new variable, say y', for both occurrences of y in $f(y, y)$ (instead of using y_1 and y_2 as above), we would get the configuration $\emptyset; \{p \approx^?_{\mathcal{R},\lambda} q,\ \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} f,\ \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} a,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} c,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} \mathrm{N}_2\}; \{x \mapsto \mathrm{N}_1(\mathrm{N}_2, \mathrm{N}_2),\ y' \mapsto \mathrm{N}_2,\ y \mapsto \mathrm{N}_2\}$. But for the given $\mathcal{R}_\lambda$, the constraint $\{p \approx^?_{\mathcal{R},\lambda} q,\ \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} f,\ \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} a,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} c,\ \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} \mathrm{N}_2\}$ does not have a solution (because the neighborhoods of a and c are not close to each other). Hence, we would lose a unifier.

3.2 Rules for neighborhood constraints

Let Φ be a *name-neighborhood mapping.* The *combination* of two mappings Φ and Ψ, denoted by $\Phi \odot \Psi$, is defined as

$$\begin{aligned}\Phi \odot \Psi := {} & \{\mathrm{N} \mapsto \Phi(\mathrm{N}) \mid \mathrm{N} \in \mathit{dom}(\Phi) \setminus \mathit{dom}(\Psi)\} \cup \\ & \{\mathrm{N} \mapsto \Psi(\mathrm{N}) \mid \mathrm{N} \in \mathit{dom}(\Psi) \setminus \mathit{dom}(\Phi)\} \cup \\ & \{\mathrm{N} \mapsto \Phi(\mathrm{N}) \cap \Psi(\mathrm{N}) \mid \mathrm{N} \in \mathit{dom}(\Psi) \cap \mathit{dom}(\Phi)\}.\end{aligned}$$

We call Φ and Ψ *compatible*, if $(\Phi \odot \Psi)(\mathrm{N}) \neq \emptyset$ for all $\mathrm{N} \in \mathit{dom}(\Phi \odot \Psi)$. Otherwise they are *incompatible.*

A *constraint configuration* is a pair $C; \Phi$, where C is a set of $(\mathcal{R}, \lambda)$-neighborhood constraints to be solved, and Φ is a name-neighborhood mapping (as a set of rules), representing the $(\mathcal{R}, \lambda)$-solution computed so far. We say that Ψ is an *$(\mathcal{R}, \lambda)$-solution of a constraint configuration* $C; \Phi$ if Ψ is an $(\mathcal{R}, \lambda)$-solution to C, and Ψ and Φ are compatible.

The constraint simplification algorithm $\mathcal{CS}$ transforms constraint configurations, exhaustively applying the following rules ($\perp$ indicates failure):

(FFS) Function Symbols: $\{f \approx^?_{\mathcal{R},\lambda} g\} \uplus C;\ \Phi; \Longrightarrow C;\ \Phi,$ if $\mathcal{R}(f, g) \geq \lambda$.

(NFS) Name vs Function Symbol:

$$\{\mathrm{N} \approx^?_{\mathcal{R},\lambda} g\} \uplus C;\ \Phi \Longrightarrow C;\ \Phi \odot \{\mathrm{N} \mapsto \mathbf{pc}(g, \mathcal{R}, \lambda)\}.$$

(FSN) Function Symbol vs Name: $\{g \approx^?_{\mathcal{R},\lambda} \mathrm{N}\} \uplus C; \Phi \Longrightarrow \{\mathrm{N} \approx^?_{\mathcal{R},\lambda} g\} \cup C; \Phi$.

(NN1) Name vs Name 1:

$\{\mathrm{N} \approx^?_{\mathcal{R},\lambda} \mathrm{M}\} \uplus C;\ \Phi \Longrightarrow C;\ \Phi \odot \{\mathrm{N} \mapsto \{f\}, \mathrm{M} \mapsto \mathbf{pc}(f, \mathcal{R}, \lambda)\}$,
where $\mathrm{N} \in dom(\Phi)$, $f \in \Phi(\mathrm{N})$.

(NN2) Name vs Name 2: $\{\mathrm{M} \approx^?_{\mathcal{R},\lambda} \mathrm{N}\} \uplus C;\ \Phi \Longrightarrow \{\mathrm{N} \approx^?_{\mathcal{R},\lambda} \mathrm{M}\} \cup C;\ \Phi$,
where $\mathrm{M} \notin dom(\Phi)$, $\mathrm{N} \in dom(\Phi)$.

(Fail1) Failure 1: $\{f \approx^?_{\mathcal{R},\lambda} g\} \uplus C;\ \Phi \Longrightarrow \bot$, if $\mathcal{R}(f, g) < \lambda$.

(Fail2) Failure 2: $C;\ \Phi \Longrightarrow \bot$, if there exists $\mathrm{N} \in dom(\Phi)$ with $\Phi(\mathrm{N}) = \emptyset$.

The NN1 rule causes branching, generating n branches where n is the number of elements in $\Phi(\mathrm{N})$. (Remember that by definition, the proximity class of each symbol is finite.) When the derivation does not fail, the terminal configuration has the form $\{\mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} \mathrm{M}_1, \ldots, \mathrm{N}_n \approx^?_{\mathcal{R},\lambda} \mathrm{M}_n\}$; Φ, where for each $1 \le i \le n$, $\mathrm{N}_i, \mathrm{M}_i \notin dom(\Phi)$. Such a constraint is trivially solvable.

Theorem 5. *The constraint simplification algorithm $\mathcal{CS}$ is terminating.*

Proof. With each configuration $C; \Phi$ we associate a complexity measure, which is a triple of natural numbers (n_1, n_2, n_3): n_1 is the number of symbols occurrences in C, n_2 is the number of equations of the form $g \approx^?_{\mathcal{R},\lambda} \mathrm{N}$ in C, and n_3 is the number of equations of the form $\mathrm{M} \approx^?_{\mathcal{R},\lambda} \mathrm{N}$ in C, where $\mathrm{M} \notin dom(\Phi)$ and $\mathrm{N} \in dom(\Phi)$. Measures are compared by the lexicographic extension of the ordering $>$ on natural numbers. It is a well-founded ordering. The rules (FFS), (NFS), (NN1) decrease n_1. The rule (FSN) does not change n_1 and decreases n_2. The rule (NN2) does not change n_1 and n_2 and decreases n_3. The failing rules cause termination immediately. Hence, each rule reduces the measure or terminates. It implies the termination of the algorithm. □

In the statements below, we assume $\mathcal{R}$ and λ to be given and the problems are to be solved with respect to them.

Lemma 3. *1. If $C; \Phi \Longrightarrow \bot$ by* (Fail1) *or* (Fail2) *rules, then (C, Φ) does not have an $(\mathcal{R}, \lambda)$-solution.*
2. Let $C_1; \Phi_1 \Longrightarrow C_2; \Phi_2$ be a step performed by a constraint solving (nonfailing) rule. Then any $(\mathcal{R}, \lambda)$-solution of $C_1; \Phi_2$ is also an $(\mathcal{R}, \lambda)$-solution of $C_1; \Phi_1$.

Proof. 1. For (Fail1), the lemma follows from the definition of $(\mathcal{R}, \lambda)$-solution. For (Fail2), no Ψ is compatible with Φ which maps a name to the empty set.
2. The lemma is straightforward for (FFS), (FSN), and (NN2). To show it for (NFS), we take Ψ, which solves $C;\ \Phi \odot \{\mathrm{N} \mapsto \mathbf{pc}(g, \mathcal{R}, \lambda)\}$. By definition of $\odot$, we get $\Psi(\mathrm{N}) \subseteq \mathbf{pc}(g, \mathcal{R}, \lambda)$. But then Ψ is a solution to $\{\mathrm{N} \approx^?_{\mathcal{R},\lambda} g\} \uplus C;\ \Phi$. To show the lemma holds for (NN1), we take a solution Ψ of $C;\ \Phi \odot \{\mathrm{N} \mapsto \{f\}, \mathrm{M} \mapsto \mathbf{pc}(f, \mathcal{R}, \lambda)\}$. It implies that $\Psi(\mathrm{N}) = \{f\}$ and $\Psi(\mathrm{M}) \subseteq \mathbf{pc}(f, \mathcal{R}, \lambda)$. But then we immediately get that Ψ solves $\{\mathrm{N} \approx^?_{\mathcal{R},\lambda} \mathrm{M}\} \uplus C;\ \Phi$. □

Theorem 6 (Soundness of $\mathcal{CS}$). *Let C be an $(\mathcal{R},\lambda)$-neighborhood constraint such that $\mathcal{CS}$ produces a maximal derivation $C;\emptyset \Longrightarrow^* C';\Phi$. Then Φ is an $(\mathcal{R},\lambda)$-solution of $C \setminus C'$, and C' is a set of constraints between names which is trivially $(\mathcal{R},\lambda)$-satisfiable.*

Proof. If a neighborhood equation is not between names, there is a rule in $\mathcal{CS}$ which applies to it. Hence, a maximal derivation can not stop with a C' that contains such an equation. As for neighborhood equations between names, only two rules deal with them: (NN1) and (NN2). But they apply only if at least one of the involved names belongs to the domain of the corresponding mapping. Hence, it can happen that an equation of the form $\mathrm{N} \approx^?_{\mathcal{R},\lambda} \mathrm{M}$ is never transformed by $\mathcal{CS}$. When the algorithm stops, such equations remain in C' and are trivially solvable. We can remove C' from each configuration in $C;\emptyset \Longrightarrow^* C';\Phi$ without affecting any step, getting a derivation $C \setminus C';\emptyset \Longrightarrow^* \emptyset;\Phi$. Obviously, Φ is an $(\mathcal{R},\lambda)$-solution of $\emptyset;\Phi$. By induction on the length of the derivation and Lemma 3, we get that Φ is an $(\mathcal{R},\lambda)$-solution of $C \setminus C';\emptyset$ and, hence, it solves $C \setminus C'$. $\square$

Remark 2. When a neighborhood constraint C is produced by the pre-unification algorithm, then every maximal $\mathcal{CS}$-derivation starting from $C;\emptyset$ ends either in $\bot$ or in the pair of the form $\emptyset;\Phi$. This is due to the fact that the VE rule (which introduces names in pre-unification problems) and the subsequent decomposition steps always produce chains of neighborhood equations of the form $\mathrm{N}_1 \approx_{\mathcal{R},\lambda} \mathrm{N}_2, \mathrm{N}_2 \approx_{\mathcal{R},\lambda} \mathrm{N}_3, \dots, \mathrm{N}_n \approx_{\mathcal{R},\lambda} f$, $n \geq 1$, for the introduced N's and for some f.

Theorem 7 (Completeness of $\mathcal{CS}$). *Let C be an $(\mathcal{R},\lambda)$-neighborhood constraint produced by the pre-unification algorithm, and Φ be one of its solutions. Let $dom(\Phi) = \{\mathrm{N}_1,\dots,\mathrm{N}_n\}$. Then for each n-tuple $c_1 \in \Phi(\mathrm{N}_1),\dots,c_n \in \Phi(\mathrm{N}_n)$ there exists a $\mathcal{CS}$-derivation $C;\emptyset \Longrightarrow^* \emptyset;\Psi$ with $c_i \in \Psi(\mathrm{N}_i)$ for each $1 \leq i \leq n$.*

Proof. We fix $c_1,\dots,c_n$ such that $c_1 \in \Phi(\mathrm{N}_1),\dots,c_n \in \Phi(\mathrm{N}_n)$.

First, note that $dom(\Phi)$ coincides with $\mathcal{N}(C)$. It is implied by the assumption that C is produced by pre-unification, and Remark 2 above.

The desired derivation is constructed recursively, where the important step is to identify a single inference. To see how such a single step is made, we consider a configuration $C_i;\Phi_i$ in this derivation ($i \geq 0$, $C_0 = C$, $\Phi_0 = \emptyset$). We have $dom(\Phi_i) \subseteq dom(\Phi)$. During construction, we will maintain the following two invariants for each $i \geq 0$ (easy to check that they hold for $i = 0$):

(I1) Φ is an $(\mathcal{R},\lambda)$-solution of (C_i,Φ_i), and
(I2) for all $1 \leq j \leq n$, if $\mathrm{N}_j \in dom(\Phi_i)$, then $c_j \in \Phi_i(\mathrm{N}_j)$.

We consider the following cases:

- C_i contains an equation of the form $f \approx^?_{\mathcal{R},\lambda} g$. By **(I1)**, $\mathcal{R}(f,g) \geq \lambda$. Then we make the (FFS) step with $f \approx^?_{\mathcal{R},\lambda} g$, obtaining $C_{i+1} = C_i \setminus \{f \approx^?_{\mathcal{R},\lambda} g\}$ and $\Phi_{i+1} = \Phi_i$. Obviously, **(I1)** and **(I2)** hold also for the new configuration.
- Otherwise, assume C_i contains an equation $\mathrm{N}_k \approx^?_{\mathcal{R},\lambda} g$, where $1 \leq k \leq n$. Since Φ solves $C;\Phi_i$, we have $\Phi_i(\mathrm{N}) \neq \emptyset$ for all $\mathrm{N} \in dom(\Phi_i)$ and there is only

one choice to make the step: the (NFS) rule. It gives $C_{i+1} = C_i \setminus \{\mathrm{N}_k \approx^?_{\mathcal{R},\lambda} g\}$ and $\Phi_{i+1} = \Phi_i \odot \{\mathrm{N}_k \mapsto \mathbf{pc}(g, \mathcal{R}, \lambda)\}$. Since Φ solves, in particular, $\mathrm{N}_k \approx^?_{\mathcal{R},\lambda} g$, we have $\Phi(\mathrm{N}_k) \subseteq \mathbf{pc}(g, \mathcal{R}, \lambda)$ and, hence, $c_k \in \mathbf{pc}(g, \mathcal{R}, \lambda)$. First, assume $\mathrm{N}_k \notin \mathit{dom}(\Phi_i)$. Then $\Phi_{i+1}(\mathrm{N}_k) = \mathbf{pc}(g, \mathcal{R}, \lambda)$ and both **(I1)** and **(I2)** hold for $i+1$. Now, assume $\mathrm{N}_k \in \mathit{dom}(\Phi_i)$. Then $\Phi_{i+1}(\mathrm{N}_k) = \Phi_i(\mathrm{N}_k) \cap \mathbf{pc}(g, \mathcal{R}, \lambda)$. Besides, **(I2)** implies $c_k \in \Phi_i(\mathrm{N}_k)$. Hence, $c_k \in \Phi_{i+1}(\mathrm{N}_k)$, which implies that **(I2)** holds for $i + 1$. From $c_k \in \Phi_{i+1}(\mathrm{N}_k)$ and $c_k \in \Phi(\mathrm{N}_k)$ we get that Φ is compatible with Φ_{i+1}. Moreover, Φ solves C_i, therefore, it solves C_{i+1}. Hence, Φ solves C_{i+1}; Φ_{i+1} and **(I1)** holds for $i + 1$ as well.

– Otherwise, assume C_i contains an equation of the form $\mathrm{N}_k \approx^?_{\mathcal{R},\lambda} N_j$, where $1 \leq k, j \leq n$ and $\mathrm{N}_k \in \mathit{dom}(\Phi_i)$. By **(I1)**, we have $\mathrm{N}_k, \mathrm{N}_j \in \mathit{dom}(\Phi)$, $\Phi(\mathrm{N}_k) \cap \Phi(\mathrm{N}_j) \neq \emptyset$, and $\Phi(\mathrm{N}_k) \cap \Phi_i(\mathrm{N}_k) \neq \emptyset$. By **(I2)**, we have $c_k \in \Phi_i(\mathrm{N}_k)$. But since $c_k \in \Phi(\mathrm{N}_k)$, we have $c_k \in \Phi(\mathrm{N}_k) \cap \Phi_i(\mathrm{N}_k)$. We make the step with (NN1) rule, choosing the mapping $\mathrm{N}_k \mapsto \{c_k\}$. It gives $C_{i+1} = C_i \setminus \{\mathrm{N}_k \approx^?_{\mathcal{R},\lambda} \mathrm{N}_j\}$ and $\Phi_{i+1} = \Phi_i \odot \{\mathrm{N}_k \mapsto \{c_k\}, \mathrm{N}_j \mapsto \mathbf{pc}(c_k, \mathcal{R}, \lambda)\}$.
To see that **(I1)** holds for $i+1$, the only nontrivial thing is to check that Φ and Φ_{i+1} are compatible. For this, $\Phi_{i+1}(\mathrm{N}_k) \cap \Phi(\mathrm{N}_k) \neq \emptyset$ and $\Phi_{i+1}(\mathrm{N}_j) \cap \Phi(\mathrm{N}_j) \neq \emptyset$ should be shown.
Proving $\Phi_{i+1}(\mathrm{N}_k) \cap \Phi(\mathrm{N}_k) \neq \emptyset$: By construction, $\Phi_{i+1}(\mathrm{N}_k) = \{c_k\}$. By assumption, $c_k \in \Phi(\mathrm{N}_k)$. Hence, $\Phi_{i+1}(\mathrm{N}_k) \cap \Phi(\mathrm{N}_k) \neq \emptyset$.
Proving $\Phi_{i+1}(\mathrm{N}_j) \cap \Phi(\mathrm{N}_j) \neq \emptyset$: Since Φ solves $\mathrm{N}_k \approx^?_{\mathcal{R},\lambda} N_j$ and $c_k \in \Phi(\mathrm{N}_k)$, we have $\Phi(\mathrm{N}_j) \subseteq \mathbf{pc}(c_k, \mathcal{R}, \lambda)$. If $\mathrm{N}_j \notin \mathit{dom}(\Phi_i)$, then $\Phi_{i+1}(\mathrm{N}_j) = \mathbf{pc}(c_k, \mathcal{R}, \lambda)$ and $\Phi_{i+1}(\mathrm{N}_j) \cap \Phi(\mathrm{N}_j) \neq \emptyset$. If $\mathrm{N}_j \in \mathit{dom}(\Phi_i)$, then by **(I2)**, $c_j \in \Phi_i(\mathrm{N}_j)$. On the other hand, $c_j \in \Phi(\mathrm{N}_j)$ and, therefore, $c_j \in \mathbf{pc}(c_k, \mathcal{R}, \lambda)$. Since $\Phi_{i+1}(\mathrm{N}_j) = \Phi_i(\mathrm{N}_j) \cap \mathbf{pc}(c_k, \mathcal{R}, \lambda)$, we get $c_j \in \Phi_{i+1}(\mathrm{N}_j)$ and, hence, $\Phi_{i+1}(\mathrm{N}_j) \cap \Phi(\mathrm{N}_j) \neq \emptyset$.
To see that **(I2)** holds is easier. In fact, we have already shown above that $\Phi_{i+1}(\mathrm{N}_k) = \{c_k\}$. As for N_j, we have $\mathrm{N}_j \in \mathit{dom}(\Phi_{i+1})$ iff $\mathrm{N}_j \in \mathit{dom}(\Phi_i)$. In the latter case, we have seen in the proof of **(I1)** that $c_j \in \Phi_{i+1}(\mathrm{N}_j)$.

– The other cases will be dealt by the rules (FSN) and (NN2). The invariants for them trivially hold, since these rules do not change the problem.

By **(I1)**, the configurations in our derivation are solvable. Therefore, the failing rules do not apply. Hence, the derivation ends with $\emptyset; \Psi$ for some Ψ. By construction, $\mathit{dom}(\Psi) = \{\mathrm{N}_1, \ldots, \mathrm{N}_n\}$. By **(I2)**, $c_i \in \Psi(\mathrm{N}_i)$ for each $1 \leq i \leq n$. □

Example 3. The pre-unification derivation in Example 1 gives the neighborhood constraint $C = \{p \approx^?_{\mathcal{R},\lambda} q, \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} f, \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} a, \mathrm{N}_3 \approx^?_{\mathcal{R},\lambda} g, \mathrm{N}_4 \approx^?_{\mathcal{R},\lambda} d, \mathrm{N}_1 \approx^?_{\mathcal{R},\lambda} \mathrm{N}_3, \mathrm{N}_2 \approx^?_{\mathcal{R},\lambda} \mathrm{N}_4\}$. For $\mathcal{R}_\lambda = \{(a, b), (b, c), (c, d), (a, b'), (b', c'), (c', d), (f, g), (p, q)\}$, the algorithm $\mathcal{CS}$ gives four solutions:

$$\Phi_1 = \{\mathrm{N}_1 \mapsto \{f\}, \mathrm{N}_2 \mapsto \{b\}, \mathrm{N}_3 \mapsto \{f, g\}, \mathrm{N}_4 \mapsto \{c\}\}$$
$$\Phi_2 = \{\mathrm{N}_1 \mapsto \{f\}, \mathrm{N}_2 \mapsto \{b'\}, \mathrm{N}_3 \mapsto \{f, g\}, \mathrm{N}_4 \mapsto \{c'\}\}.$$
$$\Phi_3 = \{\mathrm{N}_1 \mapsto \{g\}, \mathrm{N}_2 \mapsto \{b\}, \mathrm{N}_3 \mapsto \{f, g\}, \mathrm{N}_4 \mapsto \{c\}\}$$
$$\Phi_4 = \{\mathrm{N}_1 \mapsto \{g\}, \mathrm{N}_2 \mapsto \{b'\}, \mathrm{N}_3 \mapsto \{f, g\}, \mathrm{N}_4 \mapsto \{c'\}\}.$$

Referring to the mappings Φ and Φ' and the substitution μ in Example 1, it is easy to observe that $\Phi(\mu) \cup \Phi'(\mu) = \Phi_1(\mu) \cup \Phi_2(\mu) \cup \Phi_3(\mu) \cup \Phi_4(\mu)$.

4 Final remarks

We described an algorithm that solves unification problems over unrestricted proximity relations. It is terminating, sound, complete, and computes a compact representation of a minimal complete set of unifiers. A next step is to incorporate the computation of unification degree into the procedure and use it to characterize the "best" unifiers. Another future work involves a combination of unranked unification [6] and proximity relations to permit proximal function symbols with possibly different arities, similarly to the analogous extension of similarity-based unification described in [1].

References

1. H. Aït-Kaci and G. Pasi. Fuzzy unification and generalization of first-order terms over similar signatures. In F. Fioravanti and J. P. Gallagher, editors, *Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, Namur, Belgium, October 10-12, 2017, Revised Selected Papers*, volume 10855 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2017.
2. D. Dubois and H. Prade. *Fuzzy sets and systems: theory and applications*, volume 144 of *Mathematics in science and engineering.* Acad. Press, 1980.
3. F. Formato, G. Gerla, and M. I. Sessa. Extension of logic programming by similarity. In M. C. Meo and M. V. Ferro, editors, *1999 Joint Conference on Declarative Programming, AGP'99, L'Aquila, Italy, September 6-9, 1999*, pages 397–410, 1999.
4. F. Formato, G. Gerla, and M. I. Sessa. Similarity-based unification. *Fundam. Inform.*, 41(4):393–414, 2000.
5. P. Julián-Iranzo and C. Rubio-Manzano. Proximity-based unification theory. *Fuzzy Sets and Systems*, 262:21–43, 2015.
6. T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computation, Joint International Conferences, AISC 2002 and Calculemus 2002, Marseille, France, July 1-5, 2002, Proceedings*, volume 2385 of *Lecture Notes in Computer Science*, pages 290–304. Springer, 2002.
7. T. Kutsia and C. Pau. Proximity-based generalization. In M. Ayala Rincón and P. Balbiani, editors, *Proceedings of the 32nd International Workshop on Unification, UNIF 2018*, 2018.
8. T. Kutsia and C. Pau. Computing all maximal clique partitions in a graph. RISC Report 19-04, RISC, Johannes Kepler University Linz, 2019.
9. T. Kutsia and C. Pau. Solving proximity constraints. RISC Report 19-06, RISC, Johannes Kepler University Linz, 2019.
10. M. Rodríguez-Artalejo and C. A. Romero-Díaz. A declarative semantics for CLP with qualification and proximity. *TPLP*, 10(4-6):627–642, 2010.
11. M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theor. Comput. Sci.*, 275(1-2):389–426, 2002.