Constraint solving over multiple similarity relations

Besik Dundua

FBT, International Black Sea University, Tbilisi, Georgia
VIAM, Ivane Javakhishvili Tbilisi State University, Georgia
bdundua@gmail.com

Temur Kutsia

Johannes Kepler University, Research Institute for Symbolic Computation, Linz, Austria
http://www.risc.jku.at/people/tkutsia/
kutsia@risc.jku.at

Mircea Marin

West University of Timişoara, Romania
http://staff.fmi.uvt.ro/~mircea.marin/
mircea.marin@e-uvt.ro

Cleopatra Pau

Johannes Kepler University, Research Institute for Symbolic Computation, Linz, Austria
ipau@risc.jku.at

Abstract

Similarity relations are reflexive, symmetric, and transitive fuzzy relations. They help to make approximate inferences, replacing the notion of equality. Similarity-based unification has been quite intensively investigated, as a core computational method for approximate reasoning and declarative programming. In this paper we consider solving constraints over several similarity relations, instead of a single one. Multiple similarities pose challenges to constraint solving, since we can not rely on the transitivity property anymore. Existing methods for unification with fuzzy proximity relations (reflexive, symmetric, non-transitive relations) do not provide a solution that would adequately reflect particularities of dealing with multiple similarities. To address this problem, we develop a constraint solving algorithm for multiple similarity relations, prove its termination, soundness, and completeness properties, and discuss applications.

2012 ACM Subject Classification Theory of computation → Logic; Computing methodologies → Symbolic and algebraic manipulation; Theory of computation → Semantics and reasoning

Keywords and phrases Fuzzy relations, similarity, constraint solving.

Digital Object Identifier 10.4230/LIPIcs.FSCD.2020.35

Acknowledgements This research has been partially supported by the Austrian Science Fund (FWF) under the project 28789-N32 and the Shota Rustaveli National Science Foundation of Georgia under the grant YS-18-1480.

1 Introduction

Reasoning with incomplete, imperfect information is very common in human communication. Its modeling is a highly nontrivial task, and remains an important issue in applications of artificial intelligence. There are various notions associated to such information (e.g., uncertainty, imprecision, vagueness, fuzziness) and different methodologies have been proposed to deal with them (e.g., approaches based on default logic, probability, fuzzy sets, etc.)

For many problems in this area, exact equality is replaced by its approximation. Several approaches use similarity relations to express the approximation, modeling the corresponding imprecise information. Similarity relations are fuzzy binary relations, specifying to which degree two objects are similar to each other. They satisfy reflexivity, symmetry, and fuzzy min-transitivity properties, and can be characterized by "level-wise" equivalence. Once the level $\lambda > 0$ of similarity is fixed, the set of all object-pairs, which have the similarity degree at least λ, form a classical equivalence relation.

Reasoning with similarity relations requires approximate inference techniques. Solving similarity-based constraints is the central computational mechanism for such inferences. Several approaches to unification modulo similarity have been proposed, see, e.g., [1, 2, 5–9, 14, 16, 17, 20, 21]. Recently, unification was studied also for proximity relations, which generalize similarity in the sense they are reflexive and symmetric but non-transitive fuzzy relations [10, 12, 15]. The techniques studied in these papers usually assume a single fuzzy (similarity or proximity) relation. However, in many practical situations, one needs to deal with several similarities between the objects from the same set, see, e.g. [18, 19], where examples about building online fashion compatibility representation and understanding visual similarities are considered in the context of learning image embeddings.

Multiple similarities pose challenges to constraint solving, since we can not rely on the transitivity property anymore. Note that proximity relations are not transitive either, but their unification methods have some limitations in dealing with multiple similarities simultaneously.

We address this problem, proposing an algorithm for solving constraints over multiple similarity relations. A simple example below illustrates the problem together with the results of different approaches, and motivates the development of a dedicated technique for it.

▶ **Example 1.** Let *white-circle*, *white-ellipse*, *gray-circle* and *gray-ellipse* be four symbols and $\mathcal{R}_1$ and $\mathcal{R}_2$ be two similarity relations, where $\mathcal{R}_1$ stands for "similar color, same shape" and $\mathcal{R}_2$ denotes "same color, similar shape". They are defined as

- $\mathcal{R}_1(\textit{white-circle}, \textit{gray-circle}) = \mathcal{R}_1(\textit{white-ellipse}, \textit{gray-ellipse}) = 0.5$, and
- $\mathcal{R}_2(\textit{white-circle}, \textit{white-ellipse}) = \mathcal{R}_2(\textit{gray-circle}, \textit{gray-ellipse}) = 0.7$.

Assume we want to find an object X such that from the color point of view, it is at least 0.4-similar to *white-circle* and from the shape point of view, it is at least 0.5-similar to *gray-ellipse*. The corresponding constraint is $X \simeq_{\mathcal{R}_1,0.4}$ *white-circle* and $X \simeq_{\mathcal{R}_2,0.5}$ *gray-ellipse*. The expected answer is $X =$ *gray-circle*. But it is problematic to compute it by the existing fuzzy unification techniques. The direct approach, trying to solve each equation separately by the weak unification algorithm from [17] leads to no solution in this case, because *white-circle* and *gray-ellipse* are not similar to each other by any of the given relations. An alternative way could be to consider the constraint over the relation $\mathcal{R}_1 \cup \mathcal{R}_2$, which is a proximity, not a similarity, since transitivity is not satisfied. However, the proximity unification algorithm from [12] gives no solution. We can try to use the algorithm for solving proximity constraints from [15], but it would give two answers instead of one: $X =$ *gray-circle* and $X =$ *white-ellipse*. On the other hand, the algorithm proposed in this paper computes the right solution $X =$ *gray-circle*. Its similarity degrees are 0.5 for the relation $\mathcal{R}_1$ and 0.7 for $\mathcal{R}_2$. ◀

It should be mentioned that the multi-adjoint framework [14, 16] is flexible enough to accommodate multiple similarities. It is a logic programming-based approach, where one needs to extend programs by fuzzy similarity axioms for each alphabet symbol and use classical unification. The authors show how to encode Sessa's algorithm [17] in this framework.

Our approach is different. We develop the solving algorithm directly, without being dependent on the implementation or application preferences. It can be incorporated in a modular way in the constraint logic programming schema, can be used for constrained rewriting, querying, or similar purposes. It combines three parts: solving syntactic equations, solving similarity problems for one relation, and solving mixed problems. Except variables for terms, we permit also variables for function symbols, since they are necessary in the process of finding an "intermediate object" between terms in different similarity relations.

The paper is organized as follows: In Section 2 we introduce the basic notions, define constraints and their solutions. Section 3 is the main section of the paper: it describes all three parts of our algorithm and presents its termination, soundness, and completeness results. In Section 4 we show how to include the computation of approximation degrees in the algorithm. Concluding discussion can be found in Section 5.

2 Preliminaries

Similarity relations

We define basic notions about similarity relations following [9,17]. 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 $0 < \lambda \leq 1$ (called *cut value*), then the λ-*cut* of $\mathcal{R}$ on S, denoted $\mathcal{R}_\lambda$, is an ordinary (crisp) relation on S defined as $\mathcal{R}_\lambda := \{(s_1, s_2) \mid \mathcal{R}(s_1, s_2) \geq \lambda\}$.

A fuzzy relation $\mathcal{R}$ on a set S is called a *proximity relation*, if it is reflexive and symmetric:

Reflexivity: $\mathcal{R}(s, s) = 1$ for all $s \in S$;
Symmetry: $\mathcal{R}(s_1, s_2) = \mathcal{R}(s_2, s_1)$ for all $s_1, s_2 \in S$.

Let $\wedge$ be a T-norm: an associative, commutative, non-decreasing binary operation on $[0,1]$ with 1 as the unit element. A proximity relation (on S) is called a *similarity relation* (on S) iff it is transitive:

Transitivity $\mathcal{R}(s_1, s_2) \geq \mathcal{R}(s_1, s) \wedge \mathcal{R}(s, s_2)$ for any $s_1, s_2, s \in S$.

In this paper, in the role of T-norm we take the *minimum* of two numbers, and write min instead of $\wedge$. In the role of S we take a syntactic domain, defined in the next section.

Terms, atoms, substitutions

Our alphabet $\mathbf{A}$ consists of the following pairwise disjoint sets of symbols:

- $\mathbf{V}_\mathsf{T}$: term variables, denoted by X, Y, Z,
- $\mathbf{V}_\mathsf{F}$: function variables, denoted by F, G, H,
- $\mathbf{C}_\mathsf{F}$: function constants, denoted by f, g, h,

By $\mathbf{V}$ we denote the set of variables $\mathbf{V} = \mathbf{V}_\mathsf{T} \cup \mathbf{V}_\mathsf{F}$, and V is used for its elements.

A *function symbol* is a function variable or a function constant, i.e., an element of the set $\mathbf{F} = \mathbf{C}_\mathsf{F} \cup \mathbf{V}_\mathsf{F}$. We use the letters $\mathsf{f}, \mathsf{g}, \mathsf{h}$ to denote function symbols. Each function symbol has a fixed arity.

Terms over $\mathbf{A}$ are defined by the grammar $t := X \mid \mathsf{f}(t_1, \ldots, t_n)$, where f is an n-ary function symbol. For terms we use the letters t, s, r. The set of terms over $\mathbf{A}$ is denoted by $\mathsf{Terms}(\mathbf{A})$.

For a term $\mathsf{f}(t_1, \ldots, t_n)$, if $n = 0$, we write just f instead of $\mathsf{f}()$. Usually, from the context it is clear whether we are talking about a symbol or about a term.

A *substitution* σ is a mapping from $\mathbf{V}$ to $\mathbf{F} \cup \mathsf{Terms}(\mathbf{A})$ such that

- $\sigma(X) \in \mathsf{Terms}(\mathbf{A})$ for all $X \in \mathbf{V}_\mathsf{T}$,
- $\sigma(F) \in \mathbf{F}$ for all $F \in \mathbf{V}_\mathsf{F}$,
- $\sigma(V) = V$ for all but finitely many variables $V \in \mathbf{V}$.

Substitutions are denoted by Greek letters σ, ϑ, φ. The identity substitution is denoted by *Id*. The domain of a substitution σ is the set $dom(\sigma) = \{V \mid V \in \mathbf{V}, \sigma(V) \neq V\}$. The *restriction* of σ to a set of variables $\mathcal{V}$ is the substitution $\sigma|_{\mathcal{V}}$ defined as $\sigma|_{\mathcal{V}}(V) = \sigma(V)$ if $V \in \mathcal{V}$ and $\sigma|_{\mathcal{V}}(V) = V$ otherwise. We will use the usual set representation of substitutions, writing σ as $\{V \mapsto \sigma(V) \mid V \in dom(\sigma)\}$.

Substitution application to variables, constants, and terms is defined as follows: $c\sigma = c$ for all $c \in \mathbf{C}_\mathsf{F}$, $V\sigma = \sigma(V)$ for all $V \in \mathbf{V}$, and $\mathsf{f}(t_1, \ldots, t_n)\sigma = (\mathsf{f}\sigma)(t_1\sigma, \ldots, t_n\sigma)$.

Similarity relations on syntactic domains

Our similarity relations are defined on the set of constants $\mathbf{C}_\mathsf{F}$. Any such relation $\mathcal{R}$ should satisfy the restriction: $\mathcal{R}(f, g) = 0$, if f and g have different arity.

Given an $\mathcal{R}$ defined on $\mathbf{C}_\mathsf{F}$, we extend it to $\mathbf{F} \cup \mathsf{Terms}(\mathbf{A})$:

- For variables: $\mathcal{R}(V, V) = 1$.
- For nonvar. terms: $\mathcal{R}(\mathsf{f}(t_1, \ldots, t_n), \mathsf{g}(s_1, \ldots, s_n)) = \min(\mathcal{R}(\mathsf{f}, \mathsf{g}), \mathcal{R}(t_1, s_1), \ldots, \mathcal{R}(t_n, s_n))$, when f and g are both n-ary.
- In all other cases, $\mathcal{R}(\tau_1, \tau_2) = 0$ for $\tau_1, \tau_2 \in \mathbf{F} \cup \mathsf{Terms}(\mathbf{A})$.

Given a similarity relation $\mathcal{R}$ and the cut value $\lambda \in (0, 1]$, we define $(\mathcal{R}, \lambda)$-*neighborhood* of τ as $\mathsf{N}(\tau, \mathcal{R}, \lambda) := \{\tau' \mid \mathcal{R}(\tau, \tau') \geq \lambda\}$, where $\tau, \tau' \in \mathbf{F} \cup \mathsf{Terms}(\mathbf{A})$. Based on the definition of similarity relations above, it is obvious that neighborhoods of function constants (resp. variables) contain only function constants (resp. variables) of the same arity. Neighborhoods of terms contain only terms. All terms in the same neighborhood have the same structure (same set of positions). We require for each $f \in \mathbf{C}_\mathsf{F}$, $\mathcal{R}$, and λ, the set $\mathsf{N}(f, \mathcal{R}, \lambda)$ to be finite. It implies that term neighborhoods are finite as well.

Constraints

In our constraint language, the elements of $\mathbf{F} \cup \mathsf{Terms}(\mathbf{A})$ are the basic objects. In the rest of the paper, the letter τ is used to denote its elements. Besides, we have the equality predicate constant $\doteq$ (interpreted as syntactic equality), one or more similarity predicate constants $\simeq_1$, $\simeq_2$, ..., (interpreted as similarity relations on $\mathbf{F} \cup \mathsf{Terms}(\mathbf{A})$), propositional constants true and false, connectives $\wedge, \vee$, and the quantifier $\exists$.

Primitive constraints $\mathcal{P}$ are defined by the grammar

$$\mathcal{P} ::= \ \mathsf{true} \mid \mathsf{false} \mid t \doteq s \mid t \simeq s \mid \mathsf{f} \doteq \mathsf{g} \mid \mathsf{f} \simeq \mathsf{g},$$

where $\simeq\ \in \{\simeq_1, \simeq_2 \ \ldots\}$. Primitive $\doteq$- and $\simeq$-constraints are called *primitive equality constraints* and *primitive similarity constraints*, respectively. A *literal* L is an atom or a primitive constraint. A (positive) *constraint* $\mathcal{C}$ over $\mathbf{A}$ is defined as $\mathcal{C} ::= \mathcal{P} \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \mid \exists x.\mathcal{C}$. In this paper we consider only positive constraints.

The domain of the *intended interpretation* of our constraint language is its Herbrand universe (the set of ground terms). The predicate constant $\doteq$ is interpreted as syntactic equality. Each similarity predicate constant $\simeq$ is interpreted as a similarity relation on the domain as defined in the previous section. When a predicate constant $\simeq$ is to be interpreted by a relation $\mathcal{R}$ with the cut value $\lambda \in (0, 1]$, we write $\simeq_{\mathcal{R},\lambda}$ instead of $\simeq$.

A *variable-predicate pair (VP-pair)* is either $\langle V, \simeq_{\mathcal{R},\lambda}\rangle$ or $\langle V, \doteq\rangle$. We say that a substitution σ is *more general* than ϑ on a set of VP-pairs $\mathcal{W}$ iff there exists a substitution φ such that $\mathcal{R}(V\sigma\varphi, V\vartheta) \geq \lambda$ for all $\langle V, \simeq_{\mathcal{R},\lambda}\rangle \in \mathcal{W}$ and $V\sigma\varphi = V\vartheta$ for all $\langle V, \doteq\rangle \in \mathcal{W}$. In this case we write $\sigma \preceq_{\mathcal{W}} \vartheta$.

▶ **Example 2.** Let $\mathcal{R}_1(a,b) = 0.7$, $\mathcal{R}_1(b,c) = 0.7$, $\mathcal{R}_1(a,c) = 0.8$, $\mathcal{R}_2(b,c) = 0.9$, and $\mathcal{W} = \{\langle X, \simeq_{\mathcal{R}_1,0.5}\rangle, \langle Y, \simeq_{\mathcal{R}_1,0.6}\rangle, \langle Y, \simeq_{\mathcal{R}_2,0.7}\rangle\}$.

- Let $\sigma = \{X \mapsto Y\}$ and $\vartheta = \{X \mapsto a, Y \mapsto b\}$. Then $\sigma \preceq_{\mathcal{W}} \vartheta$, because for $\varphi = \{X \mapsto b, Y \mapsto b\}$ we have $X\sigma\varphi = b \simeq_{\mathcal{R}_1,0.5} a = X\vartheta$, $Y\sigma\varphi = b \simeq_{\mathcal{R}_1,0.6} b = Y\vartheta$, and $Y\sigma\varphi = b \simeq_{\mathcal{R}_2,0.7} b = Y\vartheta$.
- Let $\sigma = \{X \mapsto Y\}$ and $\vartheta = \{X \mapsto a, Y \mapsto c\}$. Then $\sigma \preceq_{\mathcal{W}} \vartheta$, because for $\varphi = \{X \mapsto b, Y \mapsto b\}$ we have $X\sigma\varphi = b \simeq_{\mathcal{R}_1,0.5} a = X\vartheta$, $Y\sigma\varphi = b \simeq_{\mathcal{R}_1,0.6} c = Y\vartheta$, and $Y\sigma\varphi = b \simeq_{\mathcal{R}_2,0.7} c = Y\vartheta$.
- Let $\sigma = \{X \mapsto f(Y), Y \mapsto Z\}$ and $\vartheta = \{X \mapsto f(Z), Y \mapsto a, Z \mapsto X\}$. Then $\sigma \preceq_{\mathcal{W}} \vartheta$, because for $\varphi = \{Y \mapsto Z, Z \mapsto a\}$ we have $X\sigma\varphi = f(Z) \simeq_{\mathcal{R}_1,0.5} f(Z) = X\vartheta$, $Y\sigma\varphi = a \simeq_{\mathcal{R}_1,0.6} a = Y\vartheta$, and $Y\sigma\varphi = a \simeq_{\mathcal{R}_2,0.7} a = Y\vartheta$.

▶ **Theorem 3.** *$\preceq_{\mathcal{W}}$ is a quasi-ordering for all $\mathcal{W}$.*

Proof. Reflexivity is obvious. For transitivity, assume $\sigma_1 \preceq_{\mathcal{W}} \sigma_2$ and $\sigma_2 \preceq_{\mathcal{W}} \sigma_3$. We will show $\sigma_1 \preceq_{\mathcal{W}} \sigma_3$. Take $\langle V, \simeq_{\mathcal{R},\lambda}\rangle \in \mathcal{W}$. Then for some φ_1 and φ_2 we have $\mathcal{R}(V\sigma_1\varphi_1, V\sigma_2) \geq \lambda$ and $\mathcal{R}(V\sigma_2\varphi_2, V\sigma_3) \geq \lambda$. Since similarity is stable for substitutions [17, Proposition 3.1], we have $\mathcal{R}(V\sigma_1\varphi_1\varphi_2, V\sigma_2\varphi_2) \geq \lambda$. By transitivity of similarity, we get $\mathcal{R}(V\sigma_1\varphi_1\varphi_2, V\sigma_3) \geq \min(\mathcal{R}(V\sigma_1\varphi_1\varphi_2, V\sigma_2\varphi_2), \mathcal{R}(V\sigma_2\varphi_2, V\sigma_3)) \geq \lambda$, which implies that $\sigma_1 \preceq_{\mathcal{W}} \sigma_3$. ◀

We denote the equivalence relation induced by $\preceq_{\mathcal{W}}$ by $\cong$.

The notation $\mathcal{K}_{\doteq}$ denotes a conjunction of primitive equality constraints. By $\mathcal{K}_{\mathcal{R},\lambda}$ we denote a conjunction of primitive similarity constraints, all with the same relation $\mathcal{R}$ and the same λ-cut: $\mathcal{K}_{\mathcal{R},\lambda} = \tau_1 \simeq_{\mathcal{R},\lambda} \tau_1' \wedge \cdots \wedge \tau_n \simeq_{\mathcal{R},\lambda} \tau_n'$.

Given a constraint $\mathcal{K} = \mathcal{K}_{\doteq} \wedge \mathcal{K}_{\mathcal{R}_1,\lambda_1} \wedge \cdots \wedge \mathcal{K}_{\mathcal{R}_m,\lambda_m}$, we denote by $\mathcal{W}(\mathcal{K})$ the set of VP-pairs $\mathcal{W}(\mathcal{K}) := \{\langle V, \doteq\rangle \mid V \in var(\mathcal{K}_{\doteq})\} \cup \{\langle V, \simeq_{\mathcal{R}_i,\lambda_i}\rangle \mid V \in var(\mathcal{K}_{\mathcal{R}_i,\lambda_i}), 1 \leq i \leq m\}$.

▶ **Definition 4** (Solution). *A substitution σ is called a* solution *of a primitive constraint $\mathcal{P}$, if*

- *$\mathcal{P} = \tau_1 \doteq \tau_2$ and $\tau_1\sigma = \tau_2\sigma$, or*
- *$\mathcal{P} = \tau_1 \simeq_{\mathcal{R},\lambda} \tau_2$ and $\mathcal{R}(\tau_1\sigma, \tau_2\sigma) \geq \lambda$.*

Any substitution is a solution of true*, while* false *has no solution.*

A substitution σ is a solution of a conjunction of primitive constraints *$\mathcal{K}$ iff it solves each primitive constraint in $\mathcal{K}$. We denote the* set of all solutions of *$\mathcal{K}$ by $Sol(\mathcal{K})$. For a constraint $\mathcal{C} = \mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$ in disjunctive normal form (DNF), we define $Sol(\mathcal{C}) = \cup_{i=1}^{n} Sol(\mathcal{K}_i)$.*

Given similarity relations $\mathcal{R}_1, \ldots, \mathcal{R}_n$, a conjunction of primitive constraints $\mathcal{K}$, and its solution σ, we say that σ solves *$\mathcal{K}$* with approximation degrees *$\mathfrak{D} = \{\langle \mathcal{R}_1, \mathfrak{d}_1\rangle, \ldots, \langle \mathcal{R}_n, \mathfrak{d}_n\rangle\}$ if*

- *$\mathcal{K} =$ true or $\mathcal{K} = \mathcal{K}_{\doteq}$ and $\mathfrak{d}_1 = \cdots = \mathfrak{d}_n = 1$,*
- *$\mathcal{K} = \tau_1 \simeq_{\mathcal{R}_j,\lambda_j} \tau_2$ for some $1 \leq j \leq n$, $\mathfrak{d}_j = \mathcal{R}_j(\tau_1\sigma, \tau_2\sigma) \geq \lambda_j$, and $\mathfrak{d}_i = 1$ for all $1 \leq i \leq n$, $i \neq j$,*
- *$\mathcal{K} = \mathcal{P} \wedge \mathcal{K}'$, σ solves $\mathcal{P}$ and $\mathcal{K}'$ with approximation degrees $\{\langle \mathcal{R}_1, \mathfrak{d}_1^{\mathcal{P}}\rangle, \ldots, \langle \mathcal{R}_n, \mathfrak{d}_n^{\mathcal{P}}\rangle\}$ and $\{\langle \mathcal{R}_1, \mathfrak{d}_1'\rangle, \ldots, \langle \mathcal{R}_n, \mathfrak{d}_n'\rangle\}$, respectively, and $\mathfrak{d}_i = \min\{\mathfrak{d}_i^{\mathcal{P}}, \mathfrak{d}_i'\}$ for all $1 \leq i \leq n$.*

Such a definition of approximation degrees gives the flexibility to characterize approximations with respect to each involved relation independently from each other.

▶ **Theorem 5.** *Let $\mathcal{K} = \mathcal{K}_{\doteq} \wedge \mathcal{K}_{\mathcal{R}_1,\lambda_1} \wedge \cdots \wedge \mathcal{K}_{\mathcal{R}_m,\lambda_m}$ be a constraint. If σ is a solution of $\mathcal{K}$ and $\sigma \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$, then ϑ is a solution of $\mathcal{K}$.*

Proof. Let $s_1 \simeq_{\mathcal{R}_i,\lambda_i} s_2 \in \mathcal{K}_{\mathcal{R}_i,\lambda_i}$. From $\sigma \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$, by definition of $\preceq_{\mathcal{W}(\mathcal{K})}$, there exists a φ such that $\mathcal{R}(V\sigma\varphi, V\vartheta) \geq \lambda_i$ for each $V \in var(\mathcal{K}_{\mathcal{R}_i,\lambda_i})$. It implies that

$$\mathcal{R}(s_j\sigma\varphi, s_j\vartheta) \geq \lambda_i, \quad j = 1, 2. \tag{1}$$

On the other hand, for similarity relations $\mathcal{R}(s_1\sigma\varphi, s_2\sigma\varphi) = \mathcal{R}(s_1\sigma, s_2\sigma)$ (see [17]). Since σ is a solution of $\mathcal{K}$, $\mathcal{R}(s_1\sigma, s_2\sigma) \geq \lambda_i$. Hence, we have $\mathcal{R}(s_1\sigma\varphi, s_2\sigma\varphi) \geq \lambda_i$. From this inequality and (1), by symmetry and transitivity of $\mathcal{R}$, we get $\mathcal{R}(s_1\vartheta, s_2\vartheta) \geq \lambda_i$. Hence, ϑ is a solution of $s_1 \simeq_{\mathcal{R}_i,\lambda_i} s_2$.

It is straightforward that ϑ is a solution of any equation from $\mathcal{K}_{\doteq}$. Hence, ϑ is a solution of $\mathcal{K}$. ◀

▶ **Definition 6** (Solved form, approximately solved form). *A conjunction of primitive constraints $\mathcal{K}$ is in* solved form*, if $\mathcal{K}$ is either* true *or each primitive constraint in $\mathcal{K}$ has a form $V \doteq \tau$ or $V \simeq_{\mathcal{R},\lambda} \tau$, where V appears only once in $\mathcal{K}$. A constraint in DNF $\mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$ is in solved form, if each $\mathcal{K}_i$ is in solved form.*

A conjunction of primitive constraints $\mathcal{K}_{\text{sol}} \wedge \mathcal{K}_{\text{var}}$ is in approximately solved form (appr-solved form) *if $\mathcal{K}_{\text{sol}}$ is in solved form, and $\mathcal{K}_{\text{var}}$ is a conjunction of primitive similarity constraints between variables $V_1 \simeq_{\mathcal{R},\lambda} V_2$ such that neither V_1 nor V_2 appear in the left hand side of any primitive constraint in $\mathcal{K}_{\text{sol}}$. A constraint in DNF $\mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$ is in appr-solved form, if each $\mathcal{K}_i$ is in appr-solved form.*

Solved forms are also appr-solved forms, but not vice versa. Each solved form $\mathcal{K}$ *induces* a substitution, denoted by $\sigma_{\mathcal{K}}$: if $\mathcal{K} = \mathsf{true}$, then $\sigma_{\mathcal{K}} = Id$, otherwise $\sigma_{\mathcal{K}} = \{V \mapsto \tau \mid V \doteq \tau \in \mathcal{K} \text{ or } V \simeq_{\mathcal{R},\lambda} \tau \in \mathcal{K}\}$. Obviously, $\sigma_{\mathcal{K}}$ is a solution of $\mathcal{K}$. A constraint $\mathcal{K}_{\text{sol}} \wedge \mathcal{K}_{\text{var}}$ in appr-solved form is also solvable, because $\sigma_{\mathcal{K}_{\text{sol}}}$ solves $\mathcal{K}_{\text{sol}}$ and $\mathcal{K}_{\text{var}}$ always has at least a trivial solution mapping all terms variables to the same term variable and all function variables to the same function variable.

▶ **Example 7.** Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be defined as in Example 1 and $\mathcal{K} = \mathcal{K}_{\mathcal{R}_1,0.4} \wedge \mathcal{K}_{\mathcal{R}_2,0.5}$ be a constraint, where $\mathcal{K}_{\mathcal{R}_1,0.4} = X \simeq_{\mathcal{R}_1,0.4}$ *white-circle* $\wedge\, X \simeq_{\mathcal{R}_1,0.4} Y$ and $\mathcal{K}_{\mathcal{R}_2,0.5} = X \simeq_{\mathcal{R}_2,0.5}$ *gray-ellipse* $\wedge\, Y \simeq_{\mathcal{R}_2,0.5}$ *white-ellipse*.

One can bring $\mathcal{K}_{\mathcal{R}_1,0.4}$ to its equivalent solved form (e.g., by an algorithm along the lines of the weak unification algorithm in [17]). $\mathcal{K}_{\mathcal{R}_2,0.5}$ is already in the solved form. Hence, $\mathcal{K}$ is equivalent to the constraint

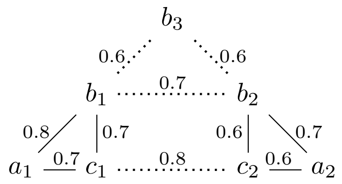
$$\begin{aligned}
&X \simeq_{\mathcal{R}_1,0.4} \textit{white-circle} \wedge Y \simeq_{\mathcal{R}_1,0.4} \textit{white-circle} \wedge \\
&X \simeq_{\mathcal{R}_2,0.5} \textit{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \textit{white-ellipse},
\end{aligned}$$

which is not yet in a solved form. A solved form, equivalent to $\mathcal{K}$, would be $X \doteq$ *gray-circle* $\wedge\, Y \doteq$ *white-circle*. It induces the substitution $\sigma = \{X \mapsto \textit{gray-circle}, Y \mapsto \textit{white-circle}\}$.

▶ **Example 8.** Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two similarity relations defined as

$$\begin{aligned}
\mathcal{R}_1: \quad & \mathcal{R}_1(a_1, c_1) = \mathcal{R}_1(b_1, c_1) = 0.7, \quad \mathcal{R}_1(a_1, b_1) = 0.8, \\
& \mathcal{R}_1(a_2, c_2) = \mathcal{R}_1(b_2, c_2) = 0.6, \quad \mathcal{R}_1(a_2, b_2) = 0.7, \\
\mathcal{R}_2: \quad & \mathcal{R}_2(b_1, b_3) = \mathcal{R}_2(b_2, b_3) = 0.6, \quad \mathcal{R}_2(b_1, b_2) = 0.7, \quad \mathcal{R}_2(c_1, c_2) = 0.8.
\end{aligned}$$

Visually:



$\mathcal{R}_1$: the solid lines, $\mathcal{R}_2$: the dotted lines.

Let $\mathcal{K} = X \simeq_{\mathcal{R}_1,0.5} f(a_1, a_2) \wedge X \simeq_{\mathcal{R}_2,0.6} f(Y, Y)$. It is equivalent to the disjunction of two solved forms, e.g., $(X \doteq f(b_1, b_2) \wedge Y \simeq_{\mathcal{R}_2,0.6} b_1) \vee (X \doteq f(c_1, c_2) \wedge Y \simeq_{\mathcal{R}_2,0.6} c_1)$. The solved forms induce two substitutions: $\sigma_1 = \{X \mapsto f(b_1, b_2), Y \mapsto b_1\}$ and $\sigma_2 = \{X \mapsto f(c_1, c_2), Y \mapsto c_1\}$. They are solutions of $\mathcal{K}$. There are other solutions of $\mathcal{K}$ that are $\cong$-equivalent to σ_1 or σ_2: $\vartheta_1 = \{X \mapsto f(b_1, b_2), Y \mapsto b_2\} \cong \sigma_1$, $\vartheta_2 = \{X \mapsto f(b_1, b_2), Y \mapsto b_3\} \cong \sigma_1$, and $\vartheta_3 = \{X \mapsto f(c_1, c_2), Y \mapsto c_2\} \cong \sigma_2$.

Now let $\mathcal{K} = X \simeq_{\mathcal{R}_1,0.8} g(Y) \wedge X \simeq_{\mathcal{R}_2,0.6} g(Z)$. A solved form $\mathcal{K}_{\mathrm{s}} = X \doteq g(Z) \wedge Y \doteq Z$ implies $\mathcal{K}$, but is not equivalent to it, because $\mathcal{K}$ has solutions $\{X \mapsto g(b_1), Y \mapsto a_1, Z \mapsto b_2\}$ and $\{X \mapsto g(b_1), Y \mapsto a_1, Z \mapsto b_3\}$, which do not solve $\mathcal{K}_{\mathrm{s}}$. On the other hand, if we take the approximate solved form $\mathcal{K}_{\mathrm{as}} = X \doteq g(X_1) \wedge X_1 \simeq_{\mathcal{R}_1,0.8} Y \wedge X_1 \simeq_{\mathcal{R}_2,0.6} Z$, then every solution of $\mathcal{K}_{\mathrm{as}}$ solves $\mathcal{K}$, and $(\exists X_1.\mathcal{K}_{\mathrm{as}})\sigma$ holds for any solution σ of $\mathcal{K}$. (Substitution application to a quantified constraint avoids variable capture.) Alternatively, we could have taken another solved form $\mathcal{K}'_{\mathrm{s}} = X \doteq g(X_1) \wedge Y \simeq_{\mathcal{R}_1,0.8} X_1 \wedge Z \simeq_{\mathcal{R}_2,0.6} X_1$ which has the same properties as $\mathcal{K}_{\mathrm{as}}$.

▶ **Example 9.** Let $\mathcal{R}_1(a, b_1) = \mathcal{R}_1(b_1, b_2) = \mathcal{R}_1(a, b_2) = 0.8$, $\mathcal{R}_2(c, b_1) = \mathcal{R}_2(b_1, b_2) = \mathcal{R}_2(b_2, c) = 0.7$ and consider a constraint $\mathcal{K} = X \simeq_{\mathcal{R}_1,0.6} f(Y, Y) \wedge X \simeq_{\mathcal{R}_2,0.5} f(Z, Z)$. The straightforward solved form $X \doteq f(Z, Z) \wedge Y \doteq Z$, as in the previous example, has fewer solutions than $\mathcal{K}$, e.g., $\{X \mapsto f(b_1, b_2), Y \mapsto a, Z \mapsto c\}$ would be lost. If we take an appr-solved form $\mathcal{K}_{\mathrm{as}} = X \doteq f(X_1, X_2) \wedge X_1 \simeq_{\mathcal{R}_1,0.6} Y \wedge X_1 \simeq_{\mathcal{R}_2,0.5} Z \wedge X_2 \simeq_{\mathcal{R}_1,0.6} Y \wedge X_2 \simeq_{\mathcal{R}_2,0.5} Z$, then all solutions of $\mathcal{K}_{\mathrm{as}}$ solve $\mathcal{K}$ and for each solution σ of $\mathcal{K}$, we have $(\exists X_1, X_2.\mathcal{K}_{\mathrm{as}})\sigma$. Unlike the previous example, we can not turn this appr-solved form into a solved form by swapping sides of variables-only equations.

3 Constraint solving

The constraint solving algorithm Solve presented in this section works on constraints in DNF. Its rules are divided into three groups: for equalities, for similarities, and for mixed problems. They are applied modulo associativity, commutativity, and idempotence of $\wedge$ and $\vee$, treating false as the unit element of $\vee$. We first introduce the rules and then show how Solve is defined using them.

In the rules, the superscript ? indicates that the constraints are supposed to be solved. The sides of an equation $V \doteq^? \tau$ belong to the same syntactic category, i.e., it stands either for $X \doteq^? t$ or for $F \doteq^? \mathsf{f}$. The same holds for $V \simeq^?_{\mathcal{R},\lambda} \tau$.

Equality rules

In this subsection we describe the rules that solve equality constraints. Essentially, these are first-order unification rules with a slight modification, which concerns dealing with function and predicate variables. The rules have the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$, which defines the transformation $\mathcal{K} \vee \mathcal{C} \rightsquigarrow \mathcal{K}' \vee \mathcal{C}$. Note that $\mathcal{C}$ does not change.

The rules are Del-eq (deletion), Dec-eq (decomposition), Ori-eq (orientation), Elim-eq (variable elimination), Confl-eq (conflict), Mism-eq (arity mismatch), Occ-eq (occurrence check), all formulated for the equality relation $\doteq$.

$$\textsf{Del-eq}: \quad \tau \doteq^? \tau \wedge \mathcal{K} \rightsquigarrow \mathcal{K}, \quad \text{where } \tau \in \mathbf{C}_\mathsf{F} \cup \mathbf{V}_\mathsf{F} \cup \mathbf{V}_\mathsf{T}.$$

$$\textsf{Dec-eq}: \quad \mathsf{f}(t_1,\dots,t_n) \doteq^? \mathsf{g}(s_1,\dots,s_n) \wedge \mathcal{K} \rightsquigarrow \mathsf{f} \doteq^? \mathsf{g} \wedge t_1 \doteq^? s_1 \wedge \cdots \wedge t_n \doteq^? s_n \wedge \mathcal{K}, \text{ where } n > 0.$$

$$\textsf{Ori-eq}: \quad \tau \doteq^? V \wedge \mathcal{K} \rightsquigarrow V \doteq^? \tau \wedge \mathcal{K}, \quad \text{if } \tau \notin \mathbf{V}.$$

$$\textsf{Elim-eq}: \quad V \doteq^? \tau \wedge \mathcal{K} \rightsquigarrow V \doteq^? \tau \wedge \mathcal{K}\{V \mapsto \tau\}, \quad \text{if } V \notin \mathit{var}(\tau) \text{ and } V \in \mathit{var}(\mathcal{K}).$$

$$\textsf{Confl-eq}: \quad f \doteq^? g \wedge \mathcal{K} \rightsquigarrow \mathsf{false}, \quad \text{where } f \neq g.$$

$$\textsf{Mism-eq}: \quad \mathsf{f}(t_1,\dots,t_n) \doteq^? \mathsf{g}(s_1,\dots,s_m) \wedge \mathcal{K} \rightsquigarrow \mathsf{false}, \text{ if } n \neq m.$$

$$\textsf{Occ-eq}: \quad X \doteq^? t \wedge \mathcal{K} \rightsquigarrow \mathsf{false}, \text{ if } X \in \mathit{var}(t) \text{ and } X \neq t.$$

Note that the Elim-eq rule replaces occurrences of a variable in the whole $\mathcal{K}$, i.e., the variable gets replaced both in equality and similarity constraints.

We define the algorithm Unif, which applies the equality rules as long as possible. When there are more than one applicable rule, the algorithm may choose one arbitrarily.

▶ **Theorem 10.** Unif *is terminating.*

Proof. Similar to the proof of termination of the unification algorithm in [3]. ◀

▶ **Lemma 11** (Soundness lemma for Unif)**.** *If $\mathcal{K} \rightsquigarrow \mathcal{K}'$ is a step performed by a rule in* Unif*, then $\mathit{Sol}(\mathcal{K}) = \mathit{Sol}(\mathcal{K}')$.*

Proof. When $\mathcal{K}$ consists of equational constraints only, then so is $\mathcal{K}'$ and the lemma can be proved as the analogous property of the unification algorithm in [3]. If $\mathcal{K}$ contains similarity constraints as well, the only nontrivial case to consider is the Elim-eq rule. We will need the fact that for any σ and ϑ, $\vartheta\sigma \in \mathit{Sol}(\mathcal{K})$ iff $\sigma \in \mathit{Sol}(\mathcal{K}\vartheta)$ (which is straightforward to show).

Let $\mathcal{K} = \{V \doteq^? \tau\} \wedge \mathcal{K}_0$ and $\vartheta = \{V \mapsto \tau\}$. Then $\mathcal{K}' = \{V \doteq^? \tau\} \wedge \mathcal{K}_0\vartheta$ and we have

$$\begin{aligned}
&\sigma \in \mathit{Sol}(\mathcal{K}) \text{ iff } \sigma \in \mathit{Sol}(\{V \doteq^? \tau\} \wedge \mathcal{K}_0) \text{ iff}\\
&\quad V\sigma = \tau\sigma \wedge \sigma \in \mathit{Sol}(\mathcal{K}_0) \text{ iff (because } V\sigma = \tau\sigma \text{ implies } \sigma = \vartheta\sigma)\\
&\quad V\sigma = \tau\sigma \wedge \vartheta\sigma \in \mathit{Sol}(\mathcal{K}_0) \text{ iff}\\
&\quad V\sigma = \tau\sigma \wedge \sigma \in \mathit{Sol}(\mathcal{K}_0\vartheta) \text{ iff}\\
&\quad \sigma \in \mathit{Sol}(\{V \doteq^? \tau\} \wedge \mathcal{K}_0\vartheta) \text{ iff}\\
&\sigma \in \mathit{Sol}(\mathcal{K}').
\end{aligned}$$

◀

Similarity rules

The rules in this section are designed for similarity relations. They resemble weak unification rules [9, 17], with the difference that function variables and multiple similarity relations are permitted, and aims at computing the answer in solved form, instead of a substitution.

In the Elim-sim rule, the variable V is replaced by τ only in the constraints for the same similarity relation. This is justified by the fact that although a λ-cut of each similarity relation is transitive, from $t \simeq_{\mathcal{R}_1,\lambda_1} s$, $s \simeq_{\mathcal{R}_2,\lambda_2} r$ we can not conclude anything about similarity between t and r.

The similarity rules have the same form as the equality rules: $\mathcal{K} \rightsquigarrow \mathcal{K}'$, which defines the transformation $\mathcal{K} \vee \mathcal{C} \rightsquigarrow \mathcal{K}' \vee \mathcal{C}$. The names are also similar to those for equalities, using sim instead of eq.

$$\mathsf{Del\text{-}sim}: \quad \tau_1 \simeq^?_{\mathcal{R},\lambda} \tau_2 \wedge \mathcal{K} \rightsquigarrow \mathcal{K}, \quad \text{where } \tau_1, \tau_2 \in \mathbf{C}_\mathsf{F} \cup \mathbf{V}_\mathsf{F} \cup \mathbf{V}_\mathsf{T} \text{ and } \mathcal{R}(\tau_1, \tau_2) \geq \lambda.$$

$$\mathsf{Dec\text{-}sim}: \quad \mathsf{f}(t_1, \ldots, t_n) \simeq^?_{\mathcal{R},\lambda} \mathsf{g}(s_1, \ldots, s_n) \wedge \mathcal{K} \rightsquigarrow \mathsf{f} \simeq^?_{\mathcal{R},\lambda} \mathsf{g} \wedge t_1 \simeq^?_{\mathcal{R},\lambda} s_1 \wedge \cdots \wedge t_n \simeq^?_{\mathcal{R},\lambda} s_n \wedge \mathcal{K}, \quad \text{where } n > 0.$$

$$\mathsf{Ori\text{-}sim}: \quad \tau \simeq^?_{\mathcal{R},\lambda} V \wedge \mathcal{K} \rightsquigarrow V \simeq^?_{\mathcal{R},\lambda} \tau \wedge \mathcal{K}, \quad \text{where } \tau \notin \mathbf{V}.$$

$$\mathsf{Elim\text{-}sim}: \quad V \simeq^?_{\mathcal{R},\lambda} \tau \wedge \mathcal{K}_{\mathcal{R},\lambda} \wedge \mathcal{K} \rightsquigarrow V \simeq^?_{\mathcal{R},\lambda} \tau \wedge \mathcal{K}_{\mathcal{R},\lambda}\{V \mapsto \tau\} \wedge \mathcal{K}$$
where $\mathcal{K}$ does not contain primitive $\simeq^?_{\mathcal{R},\lambda}$-constraints, $V \notin \mathit{var}(\tau)$, and $V \in \mathcal{K}_{\mathcal{R},\lambda}$.

$$\mathsf{Confl\text{-}sim}: \quad \tau_1 \simeq^?_{\mathcal{R},\lambda} \tau_2 \wedge \mathcal{K} \rightsquigarrow \mathsf{false}, \quad \text{where } \tau_1, \tau_2 \in \mathbf{C}_\mathsf{F} \cup \mathbf{V}_\mathsf{F} \cup \mathbf{V}_\mathsf{T}, \text{ and } \mathcal{R}(\tau_1, \tau_2) < \lambda.$$

$$\mathsf{Mism\text{-}sim}: \quad \mathsf{f}(t_1, \ldots, t_n) \simeq^?_{\mathcal{R},\lambda} \mathsf{g}(s_1, \ldots, s_m) \wedge \mathcal{K} \rightsquigarrow \mathsf{false}, \quad \text{if } n \neq m.$$

$$\mathsf{Occ\text{-}sim}: \quad X \simeq^?_{\mathcal{R},\lambda} t \wedge \mathcal{K} \rightsquigarrow \mathsf{false}, \quad \text{if } X \in \mathit{var}(t) \text{ and } X \neq t.$$

The algorithm Sim applies the similarity rules as long as possible. When there are more than one applicable rule, the algorithm may choose one nondeterministically.

Termination of Sim can be proved as termination of Unif:

▶ **Theorem 12.** Sim *is terminating.*

▶ **Lemma 13** (Soundness lemma for Sim)**.** *If $\mathcal{K} \rightsquigarrow \mathcal{K}'$ is a step performed by a rule in* Sim*, then $\mathit{Sol}(\mathcal{K}) = \mathit{Sol}(\mathcal{K}')$.*

Proof. When we have only one similarity relation, soundness follows from soundness of weak unification algorithm [17]. For the extension to multiple similarity relations, the only nontrivial rule is Elim-sim. (For the others, $\mathit{Sol}(\mathcal{K}) = \mathit{Sol}(\mathcal{K}')$ holds directly.) It is important to notice that in this rule, $\{V \mapsto \tau\}$ applies only to $\mathcal{K}_{\mathcal{R},\lambda}$. Then for $V \simeq^?_{\mathcal{R},\lambda} \tau \wedge \mathcal{K}_{\mathcal{R},\lambda}$ we have $\mathit{Sol}(V \simeq^?_{\mathcal{R},\lambda} \tau \wedge \mathcal{K}_{\mathcal{R},\lambda}) = \mathit{Sol}(V \simeq^?_{\mathcal{R},\lambda} \tau \wedge \mathcal{K}_{\mathcal{R},\lambda}\{V \mapsto \tau\})$. (It follows from soundness of weak unification algorithm [17], since the constraint is over a single similarity relation.) Constraints for all other relations remain unchanged. It implies that the solution sets for constraints in both sides of the Elim-sim rule are the same. ◀

▶ **Example 14.** let $\mathcal{K}_{\mathcal{R}_1,0.4} = X \simeq_{\mathcal{R}_1,0.4} \mathit{white\text{-}circle} \wedge X \simeq_{\mathcal{R}_1,0.4} Y$ and $\mathcal{K}_{\mathcal{R}_2,0.5} = X \simeq_{\mathcal{R}_2,0.5} \mathit{gray\text{-}ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \mathit{white\text{-}ellipse}$ be the constraints from Example 7. The reduction mentioned in that example is modeled by performing the Elim-sim step and replacing X by $\mathit{white\text{-}circle}$ in $\mathcal{K}_{\mathcal{R}_1,0.4}$:

$$\begin{aligned}
&X \simeq_{\mathcal{R}_1,0.4} \mathit{white\text{-}circle} \wedge X \simeq_{\mathcal{R}_1,0.4} Y \wedge \\
&\quad X \simeq_{\mathcal{R}_2,0.5} \mathit{gray\text{-}ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \mathit{white\text{-}ellipse} \rightsquigarrow_{\mathsf{Elim\text{-}sim}} \\
&X \simeq_{\mathcal{R}_1,0.4} \mathit{white\text{-}circle} \wedge \mathit{white\text{-}circle} \simeq_{\mathcal{R}_1,0.4} Y \wedge \\
&\quad X \simeq_{\mathcal{R}_2,0.5} \mathit{gray\text{-}ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \mathit{white\text{-}ellipse}.
\end{aligned}$$

If Elim-sim permitted to replace X not only in $\mathcal{K}_{\mathcal{R}_1,0.4}$, but also in $\mathcal{K}_{\mathcal{R}_2,0.5}$, we would get

$$\begin{aligned}
&X \simeq_{\mathcal{R}_1,0.4} \mathit{white\text{-}circle} \wedge \mathit{white\text{-}circle} \simeq_{\mathcal{R}_1,0.4} Y \wedge \\
&\quad \mathit{white\text{-}circle} \simeq_{\mathcal{R}_2,0.5} \mathit{gray\text{-}ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \mathit{white\text{-}ellipse},
\end{aligned}$$

but $\mathit{white\text{-}circle} \simeq_{\mathcal{R}_2,0.5} \mathit{gray\text{-}ellipse}$ is unsolvable. Hence, we would lose a solution.

Mixed rules

The rules in this section apply when there are at least two primitive constraints over different similarity relations. The notation $t[X]$ below means that the variable X occurs in the term t.

▶ **Definition 15** (Occurrence cycle). *An* occurrence cycle *for a variable X_1 is called the conjunction of primitive constraints $X_1 \simeq^?_{\mathcal{R}_1,\lambda_1} t_1[X_2] \wedge X_2 \simeq^?_{\mathcal{R}_2,\lambda_2} t_2[X_3] \wedge \cdots \wedge X_n \simeq^?_{\mathcal{R}_n,\lambda_n} t_n[X_1]$, where $n > 1$, $\mathcal{R}_i \neq \mathcal{R}_{i+1}$ for all $1 \leq i \leq n-1$, $\mathcal{R}_n \neq \mathcal{R}_1$, and at least one t is not a variable.*

▶ Remark 16. Note that in the definition of occurrence cycle, if two neighboring primitive similarity constraints use the same relation, they can be contracted into one constraint by transitivity, i.e., instead of $X_i \simeq^?_{\mathcal{R}_i,\lambda_i} t_i[X_{i+1}] \wedge X_{i+1} \simeq^?_{\mathcal{R}_i,\lambda_i} t_{i+1}[X_{i+2}]$ we can have $X_i \simeq^?_{\mathcal{R}_i,\lambda_i} t_i[t_{i+1}[X_{i+2}]]$, getting rid of consecutive identical similarity relations. The same is true for the last and the first constraints.

▶ **Theorem 17.** *If a conjunction of primitive constraints contains an occurrence cycle modulo symmetry of $\simeq^?_{\mathcal{R},\lambda}$, then it has no solution.*

Proof. In similarity relations, symbols of different arities can not be similar. Therefore, similar terms have the same set of positions, i.e., as trees they are the same up to renaming of nodes.

Assume by contradiction that the given occurrence cycle has a solution ϑ. It means that the following term pairs have the same structure: $X_1\vartheta$ and $t_1[X_2]\vartheta$, $X_2\vartheta$ and $t_2[X_3]\vartheta$, …, $X_n\vartheta$ and $t_n[X_1]\vartheta$. Then $X_1\vartheta$ and $t_1[t_2[\cdots[t_n[X_1]]\cdots]]\vartheta$ have the same structure. Since at least one of t_i's is not a variable, $X_1\vartheta$ is a proper subterm of $t_1[t_2[\cdots[t_n[X_1]]\cdots]]\vartheta$. But a term and its proper subterm can not have the same structure. A contradiction.

The phrase "modulo symmetry of $\simeq^?_{\mathcal{R},\lambda}$" in the theorem means that the sides of primitive constraints can be swapped, in order to detect an occurrence cycle. Since side swapping does not affect solvability of constraints, the theorem remains true if an occurrence cycle is not in the explicit form in the constraint. ◀

Below, when we talk about existence of an occurrence cycle in a constraint, we mean existence modulo symmetry of the similarity predicate.

The rules in the mixed group are rules for occurrence check (Occ-mix), mismatch (Mism-mix), and elimination of term and function variables (TVE-mix and FVE-mix, respectively). All of them except FVE-mix have the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$. As usual, they define the constraint transformation $\mathcal{K} \vee \mathcal{C} \rightsquigarrow \mathcal{K}' \vee \mathcal{C}$. As for FVE-mix, its form is $\mathcal{K} \rightsquigarrow \mathcal{K}'_1 \vee \cdots \vee \mathcal{K}'_n$, defining a transformation $\mathcal{K} \vee \mathcal{C} \rightsquigarrow \mathcal{K}'_1 \vee \cdots \vee \mathcal{K}'_n \vee \mathcal{C}$. Note that in any of these rules, $\mathcal{C}$ does not change.

In all the rules it is assumed that the constraint to be transformed (i.e., the constraint in the left hand side of $\rightsquigarrow$) has the form $\mathcal{K}_{\doteq} \wedge \mathcal{K}_{\mathcal{R}_1,\lambda_1} \wedge \cdots \wedge \mathcal{K}_{\mathcal{R}_m,\lambda_m}$, where $\mathcal{K}_{\doteq}$ and each $\mathcal{K}_{\mathcal{R}_i,\lambda_i}$, $1 \leq i \leq m$, are in *solved form*.

The TVE-mix rule uses the *renaming function* $\mathsf{\rho}$. Applied to a term, $\mathsf{\rho}$ gives its fresh copy, obtained by replacing each occurrence of a constant from $\mathbf{C}_\mathsf{F}$ by a new function variable, each occurrence of a term variable by a fresh term variable, and each occurrence of a function variable by a fresh function variable. For instance, if the term is $f(F(a, X, X, f(a)))$, we have $\mathsf{\rho}(f(F(a, X, X, f(a)))) = G_1(G_2(G_3(), Y_1, Y_2, G_4(G_5())))$, where $G_1, G_2, G_3, G_4, G_5 \in \mathbf{V}_\mathsf{F}$ are new function variables and $Y_1, Y_2 \in \mathbf{V}_\mathsf{T}$ are new term variables.

$$\textsf{Occ-mix}: \; X \simeq^?_{\mathcal{R},\lambda} t \wedge \mathcal{K} \rightsquigarrow \textsf{false}, \text{ if } X \simeq^?_{\mathcal{R},\lambda} t \wedge \mathcal{K} \text{ contains an occurrence cycle for } X.$$

$$\textsf{Mism-mix}: \; X \simeq^?_{\mathcal{R}_1,\lambda_1} \mathsf{f}(t_1, \ldots, t_n) \wedge X \simeq^?_{\mathcal{R}_2,\lambda_2} \mathsf{g}(s_1, \ldots, s_m) \wedge \mathcal{K} \rightsquigarrow \textsf{false},$$

$$\text{if } \mathcal{R}_1 \neq \mathcal{R}_2 \text{ and } m \neq n.$$

$$\mathsf{TVE\text{-}mix}: \quad X \simeq^?_{\mathcal{R},\lambda} \mathsf{f}(t_1,\ldots,t_n) \wedge \mathcal{K} \rightsquigarrow$$
$$X \doteq F(t'_1,\ldots,t'_n) \wedge F \simeq^?_{\mathcal{R},\lambda} \mathsf{f} \wedge t'_1 \simeq^?_{\mathcal{R},\lambda} t_1 \wedge \cdots \wedge t'_n \simeq^?_{\mathcal{R},\lambda} t_n \wedge \mathcal{K}\vartheta,$$

where $X \in \mathit{var}(\mathcal{K})$, $X \simeq^?_{\mathcal{R},\lambda} \mathsf{f}(t_1,\ldots,t_n) \wedge \mathcal{K}$ does not contain an occurrence cycle for X, $F(t'_1,\ldots,t'_n) = \rho(\mathsf{f}(t_1,\ldots,t_n))$, and $\vartheta = \{X \mapsto F(t'_1,\ldots,t'_n)\}$.

$$\mathsf{FVE\text{-}mix}: \quad F \simeq^?_{\mathcal{R},\lambda} f \wedge \mathcal{K} \rightsquigarrow \vee_{g\in \mathsf{N}(f,\mathcal{R},\lambda)}\big(F \doteq g \wedge \mathcal{K}\{F \mapsto g\}\big), \quad \text{where } F \in \mathit{var}(\mathcal{K}).$$

By Mix we denote one application of any of the mixed rules.

▶ **Lemma 18** (Soundness lemma for Mix)**.** *If $\mathcal{K} \rightsquigarrow \mathcal{C}$ is a step performed by a rule in* Mix*, and $\sigma \in \mathit{Sol}(\mathcal{C})$, then $\sigma \in \mathit{Sol}(\mathcal{K})$.*

Proof. For failing rules it is trivial as false has no solution. For FVE-mix, the definition of neighborhood implies it. For TVE-mix we reason as follows: Let $\mathcal{K} = \{X \simeq^?_{\mathcal{R},\lambda} \mathsf{f}(t_1,\ldots,t_n)\} \wedge \mathcal{K}_1$ and σ be a solution of the right hand side of this rule. Then $X\sigma = F(t'_1,\ldots,t'_n)\sigma \simeq_{\mathcal{R},\lambda} \mathsf{f}(t_1,\ldots,t_n)\sigma$ and σ solves $X \simeq^?_{\mathcal{R},\lambda} \mathsf{f}(t_1,\ldots,t_n)$. As for any other equation $\mathit{eq} \in \mathcal{K}_1$, we have $\mathit{eq}\vartheta$ in the right hand side, where $\vartheta = \{X \mapsto F(t'_1,\ldots,t'_n)\}$. Moreover, σ is a solution of $\mathit{eq}\vartheta$ iff $\vartheta\sigma$ is a solution of eq. The equality $X\sigma = F(t'_1,\ldots,t'_n)\sigma$ implies $\vartheta\sigma = \sigma$. Hence, σ is a solution of eq. ◀

Our constraint solving algorithm Solve is designed as a strategy of applying Unif, Sim, and Mix. To solve a conjunction of primitive equality and similarity constraints $\mathcal{K} = \mathcal{K}_{\doteq} \wedge \mathcal{K}_{\mathcal{R}_1,\lambda_1} \wedge \cdots \wedge \mathcal{K}_{\mathcal{R}_m,\lambda_m}$, it performs the following steps:

$\mathcal{C} := \mathcal{K}$
while $\mathcal{C}$ is not in the appr-solved form **do**
  $\mathcal{C} := \mathsf{Unif}(\mathcal{C})$, **if** $\mathcal{C} = \mathsf{false}$, **return** false
  $\mathcal{C} := \mathsf{Sim}(\mathcal{C})$, **if** $\mathcal{C} = \mathsf{false}$, **return** false
  $\mathcal{C} := \mathsf{Mix}(\mathcal{C})$, **if** $\mathcal{C} = \mathsf{false}$, **return** false
return $\mathcal{C}$

We write $\mathsf{Solve}(\mathcal{K}) = \mathcal{C}$, if the algorithm returns $\mathcal{C}$ for the input $\mathcal{K}$. Respectively, $\mathsf{Solve}(\mathcal{K}) = \mathsf{false}$ if false is returned.

▶ **Example 19.** Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the relations defined in Example 8 and illustrate the steps Solve would make to solve $X \simeq^?_{\mathcal{R}_1,0.5} f(a_1,a_2) \wedge X \simeq^?_{\mathcal{R}_2,0.6} f(Y,Y)$. We will explicitly distinguish between function variables and terms made of a function variable only, i.e., between F and $F()$. For the same reason, we write constant-terms a_1 and a_2 in their full form $a_1()$ and $a_2()$. Primitive constraints selected to perform a particular step are underlined.

$$\underline{X \simeq^?_{\mathcal{R}_1,0.5} f(a_1(),a_2())} \wedge X \simeq^?_{\mathcal{R}_2,0.6} f(Y,Y) \rightsquigarrow_{\mathsf{TVE\text{-}mix}}$$
$$X \doteq F(G_1(),G_2()) \wedge F \simeq^?_{\mathcal{R}_1,0.5} f \wedge \underline{G_1() \simeq^?_{\mathcal{R}_1,0.5} a_1()} \wedge \underline{G_2() \simeq^?_{\mathcal{R}_1,0.5} a_2()} \wedge$$
$$F(G_1(),G_2()) \simeq^?_{\mathcal{R}_2,0.6} f(Y,Y) \rightsquigarrow_{\mathsf{Dec\text{-}sim}\times 2}$$
$$X \doteq F(G_1(),G_2()) \wedge F \simeq^?_{\mathcal{R}_1,0.5} f \wedge G_1 \simeq^?_{\mathcal{R}_1,0.5} a_1 \wedge G_2 \simeq^?_{\mathcal{R}_1,0.5} a_2 \wedge$$
$$\underline{F(G_1(),G_2()) \simeq^?_{\mathcal{R}_2,0.6} f(Y,Y)} \rightsquigarrow_{\mathsf{Dec\text{-}sim,Ori\text{-}sim}}$$
$$X \doteq F(G_1(),G_2()) \wedge F \simeq^?_{\mathcal{R}_1,0.5} f \wedge G_1 \simeq^?_{\mathcal{R}_1,0.5} a_1 \wedge G_2 \simeq^?_{\mathcal{R}_1,0.5} a_2 \wedge$$
$$F \simeq^?_{\mathcal{R}_2,0.6} f \wedge \underline{Y \simeq^?_{\mathcal{R}_2,0.6} G_1()} \wedge G_2() \simeq^?_{\mathcal{R}_2,0.6} Y \rightsquigarrow_{\mathsf{Elim\text{-}sim}}$$
$$X \doteq F(G_1(),G_2()) \wedge F \simeq^?_{\mathcal{R}_1,0.5} f \wedge G_1 \simeq^?_{\mathcal{R}_1,0.5} a_1 \wedge G_2 \simeq^?_{\mathcal{R}_1,0.5} a_2 \wedge$$

$$
\begin{aligned}
&\quad F \simeq^?_{\mathcal{R}_2,0.6} f \wedge Y \simeq^?_{\mathcal{R}_2,0.6} G_1() \wedge \underline{G_2() \simeq^?_{\mathcal{R}_2,0.6} G_1()} \rightsquigarrow_{\mathsf{Dec\text{-}sim}}\\
&X \doteq F(G_1(), G_2()) \wedge \underline{F \simeq^?_{\mathcal{R}_1,0.5} f} \wedge G_1 \simeq^?_{\mathcal{R}_1,0.5} a_1 \wedge G_2 \simeq^?_{\mathcal{R}_1,0.5} a_2 \wedge\\
&\quad F \simeq^?_{\mathcal{R}_2,0.6} f \wedge Y \simeq^?_{\mathcal{R}_2,0.6} G_1() \wedge G_2 \simeq^?_{\mathcal{R}_2,0.6} G_1 \rightsquigarrow_{\mathsf{FVE\text{-}mix}}\\
&X \doteq f(G_1(), G_2()) \wedge F \doteq f \wedge G_1 \simeq^?_{\mathcal{R}_1,0.5} a_1 \wedge G_2 \simeq^?_{\mathcal{R}_1,0.5} a_2 \wedge\\
&\quad \underline{f \simeq^?_{\mathcal{R}_2,0.6} f} \wedge Y \simeq^?_{\mathcal{R}_2,0.6} G_1() \wedge G_2 \simeq^?_{\mathcal{R}_2,0.6} G_1 \rightsquigarrow_{\mathsf{Del\text{-}sim}}\\
&X \doteq f(G_1(), G_2()) \wedge F \doteq f \wedge \underline{G_1 \simeq^?_{\mathcal{R}_1,0.5} a_1} \wedge \underline{G_2 \simeq^?_{\mathcal{R}_1,0.5} a_2} \wedge\\
&\quad Y \simeq^?_{\mathcal{R}_2,0.6} G_1() \wedge G_2 \simeq^?_{\mathcal{R}_2,0.6} G_1 \rightsquigarrow_{\mathsf{FVE\text{-}mix}\times 2} \text{(showing only successful branches)}\\
&\big(X \doteq f(b_1(), b_2()) \wedge F \doteq f \wedge G_1 \doteq b_1 \wedge G_2 \doteq b_2 \wedge\\
&\quad Y \simeq^?_{\mathcal{R}_2,0.6} b_1() \wedge \underline{b_2 \simeq^?_{\mathcal{R}_2,0.6} b_1}\big) \vee\\
&\big(X \doteq f(c_1(), c_2()) \wedge F \doteq f \wedge G_1 \doteq c_1 \wedge G_2 \doteq c_2 \wedge\\
&\quad Y \simeq^?_{\mathcal{R}_2,0.6} c_1() \wedge \underline{c_2 \simeq^?_{\mathcal{R}_2,0.6} c_1}\big) \rightsquigarrow_{\mathsf{Del\text{-}sim}\times 2}\\
&\big(X \doteq f(b_1(), b_2()) \wedge F \doteq f \wedge G_1 \doteq b_1 \wedge G_2 \doteq b_2 \wedge Y \simeq^?_{\mathcal{R}_2,0.6} b_1()\big) \vee\\
&\big(X \doteq f(c_1(), c_2()) \wedge F \doteq f \wedge G_1 \doteq c_1 \wedge G_2 \doteq c_2 \wedge Y \simeq^?_{\mathcal{R}_2,0.6} c_1()\big).
\end{aligned}
$$

Restricting the obtained result to the original variables (and writing constant-terms in the conventional way), we get the solved form

$$(X \doteq f(b_1, b_2) \wedge Y \simeq_{\mathcal{R}_2,0.6} b_1) \vee (X \doteq f(c_1, c_2) \wedge Y \simeq_{\mathcal{R}_2,0.6} c_1).$$

▶ **Example 20.** Now we show how Solve computes an appr-solved form for the constraint from Example 9:

$$
\begin{aligned}
&\underline{X \simeq^?_{\mathcal{R}_1,0.6} f(Y, Y)} \wedge X \simeq^?_{\mathcal{R}_2,0.5} f(Z, Z) \rightsquigarrow_{\mathsf{TVE\text{-}mix}}\\
&X \doteq F(X_1, X_2) \wedge \underline{F \simeq^?_{\mathcal{R}_1,0.6} f} \wedge X_1 \simeq^?_{\mathcal{R}_1,0.6} Y \wedge X_2 \simeq^?_{\mathcal{R}_1,0.6} Y \wedge\\
&\quad F(X_1, X_2) \simeq^?_{\mathcal{R}_2,0.5} f(Z, Z) \rightsquigarrow_{\mathsf{FVE\text{-}mix}}\\
&X \doteq f(X_1, X_2) \wedge F \simeq_{\mathcal{R}_1,0.6} f \wedge X_1 \simeq^?_{\mathcal{R}_1,0.6} Y \wedge X_2 \simeq^?_{\mathcal{R}_1,0.6} Y \wedge\\
&\quad \underline{f(X_1, X_2) \simeq^?_{\mathcal{R}_2,0.5} f(Z, Z)} \rightsquigarrow_{\mathsf{Dec}}\\
&X \doteq f(X_1, X_2) \wedge F \simeq^?_{\mathcal{R}_1,0.6} f \wedge X_1 \simeq^?_{\mathcal{R}_1,0.6} Y \wedge X_2 \simeq^?_{\mathcal{R}_1,0.6} Y \wedge\\
&\quad X_1 \simeq^?_{\mathcal{R}_2,0.5} Z \wedge X_2 \simeq^?_{\mathcal{R}_2,0.5} Z.
\end{aligned}
$$

The result gives an appr-solved form. If we did not generate new copies for each variable occurrence in the $\mathsf{TVE\text{-}mix}$ rule, we would end up with $X \doteq^? f(Z, Z) \wedge F \doteq^? f \wedge Y \doteq^? Z$. As we saw in Example 9, some solutions would be lost in this case.

To prove termination of Solve, we will need an ordering on directed acyclic graphs (dags).

▶ **Definition 21** (The relation $<_{\mathrm{dag}}$)**.** *We consider dags, which have a finite set of symbols associated to each vertex. These sets are called the marks of vertices. For a graph G and a vertex v, we denote the mark of v in G by $mark(v, G)$. The relation $<_{\mathrm{dag}}$ is defined on such graphs, having the same set of vertices.*

Let $G_1 = (Vert, E_1)$ and $G_2 = (Vert, E_2)$ be two dags with the same set of vertices $Vert$. The vertices are marked in G_1 and G_2. Then $G_1 <_{\mathrm{dag}} G_2$ iff $E_1 \supseteq E_2$ and the following condition holds:

- *Let $\emptyset \neq D \subseteq Vert$ be the set of all vertices, for which the marks in the graphs differ (i.e., $D := \{v \in Vert \mid mark(v, G_1) \neq mark(v, G_2)\}$: same vertex, different markings), and*

$\emptyset \neq M \subseteq D$ be the set of those elements of D, which are not reachable from any of the elements in D in G_1. (Such a subset of D exists, because the graphs are acyclic.) Then $mark(v, G_1) \subset mark(v, G_2)$ for all $v \in M$.

We write $G >_{\text{dag}} G'$ if $G' <_{\text{dag}} G$.

▶ **Theorem 22.** *The relation $>_{\text{dag}}$ is a well-founded ordering on dags.*

Proof. First, we show that $>_{\text{dag}}$ is a strict partial order (irreflexive and transitive relation). Irreflexivity is obvious. For transitivity, assume $G_1 >_{\text{dag}} G_2$, $G_2 >_{\text{dag}} G_3$ and show $G_1 >_{\text{dag}} G_3$. Let $G_i = (Vert, E_i)$ for $i = 1, 2, 3$. By transitivity of set inclusion, we have $E_1 \subseteq E_3$.

Let $D(G_i, G_j) := \{v \in Vert \mid mark(v, G_i) \neq mark(v, G_j)\}$ and $M(G_i, G_j)$ be the set of all those elements in $D(G_i, G_j)$ that are not reachable in G_j from $D(G_i, G_j)$, $1 \leq i < j \leq 3$. Assume first that $D(G_1, G_3) \neq \emptyset$ and take $v \in M(G_1, G_3)$. We want to show that $mark(v, G_1) \supset mark(v, G_3)$. The possible cases are

i. $mark(v, G_1) \neq mark(v, G_2)$ and $mark(v, G_2) = mark(v, G_3)$.
ii. $mark(v, G_1) = mark(v, G_2)$ and $mark(v, G_2) \neq mark(v, G_3)$.
iii. $mark(v, G_1) \neq mark(v, G_2)$ and $mark(v, G_2) \neq mark(v, G_3)$.

In case **i**, $v \in D(G_1, G_2)$. If $v \in M(G_1, G_2)$, then $mark(v, G_1) \supset mark(v, G_2) = mark(v, G_3)$. Now we show that the case $v \notin M(G_1, G_2)$ is impossible. Assume by contradiction that $v \notin M(G_1, G_2)$. It means that there exists $v' \in M(G_1, G_2)$ such that $v' \to^+ v$ in G_2. But then, since $E_2 \subseteq E_3$, we have $v' \to^+ v$ in G_3. Since $v \in M(G_1, G_3)$, we should have $v' \notin D(G_1, G_3)$, i.e., $mark(v', G_1) = mark(v', G_3)$. On the other hand, from $G_1 >_{\text{dag}} G_2$ and $v' \in M(G_1, G_2)$, we have $mark(v', G_1) \supset mark(v', G_2)$, which implies $mark(v', G_2) \subset mark(v', G_3)$, $v' \in D(G_2, G_3)$ and $v' \notin M(G_2, G_3)$. Then there should exist $v'' \in M(G_2, G_3)$ such that $v'' \to^+ v'$ in G_3. Hence, we get $v'' \to^+ v' \to^+ v$ in G_3. Therefore, we can not have $v'' \in D(G_1, G_3)$, because there would be a contradiction: $v \in M(G_1, G_3)$ and v is reachable in G_3 from $v'' \in D(G_1, G_3)$. Hence, we get $mark(v'', G_1) = mark(v'', G_3)$, which, together with $v'' \in M(G_2, G_3)$ implies that $mark(v'', G_1) \subset mark(v'', G_2)$. Hence, $v'' \in D(G_1, G_2)$ and since $G_1 >_{\text{dag}} G_2$, there should exist $v''' \in M(G_1, G_2)$ such that $mark(v''', G_1) \supset mark(v''', G_2)$ and $v''' \to^+ v''$ in G_2. Then we have also $v''' \to^+ v''$ in G_3, since $E_2 \subseteq E_3$. Moreover, $mark(v''', G_2) = mark(v''', G_3)$, because otherwise we would have a contradiction with $v'' \in M(G_2, G_3)$ (there would be $v''' \in D(G_2, G_3)$ with $v''' \to^+ v''$ in G_3). Hence, $mark(v''', G_1) \supset mark(v''', G_2) = mark(v''', G_3)$ and we get $v''' \in D(G_1, G_3)$. But it contradicts our assumption that $v \in M(G_1, G_3)$, because we got $v''' \in D(G_1, G_3)$ with $v''' \to^+ v$ in G_3. The obtained contradiction shows that the case $v \notin M(G_1, G_2)$ is impossible. (See also the diagram in Appendix A.)

The case **ii** can be proved analogously. In case **iii**, if we have $mark(v, G_1) \supset mark(v, G_2)$ and $mark(v, G_2) \supset mark(v, G_3)$, we immediately get $mark(v, G_1) \supset mark(v, G_3)$. The other cases are not possible. For instance, assuming $mark(v, G_1) \supset mark(v, G_2)$ and $mark(v, G_2) \subset mark(v, G_3)$ will lead to contradiction by the same reasoning as in the proof of case **i**, where we reached $mark(v', G_1) \supset mark(v', G_2)$ and $mark(v', G_2) \subset mark(v', G_3)$.

For the assumption $D(G_1, G_3) = \emptyset$ we get a contradiction analogously.

Well-foundedness follows from well-foundedness of $\supset$, from the facts that the set of vertices is fixed and the set of edges can not be infinitely increased, and from boundedness of the length of paths due to acyclicity. ◀

▶ **Theorem 23** (Termination of Solve). *The algorithm* Solve *terminates and gives either* false *or a constraint in appr-solved form.*

Proof. Let $\mathcal{K}_0$ be a given conjunction of primitive constraints. We build a term variable dependency graph $G = (\mathit{Vert}, E)$ from $\mathcal{K}_0$ and maintain it during the process of solving. The vertices of G correspond to term variables in $\mathcal{K}_0$ so that to each variable a single vertex is assigned. For instance, if $\mathcal{K}_0$ contains three term variables X_1, X_2, and Y_3, then $\mathit{Vert} = \{v_{X_1}, v_{X_2}, v_{X_3}\}$. The initial marking is defined as $\mathit{mark}(v_X, G) = \{X\}$ for each $v_X \in \mathit{Vert}$. Next, to define E, we do the following: If $\mathcal{K}_0$ contains a solved primitive constraint $X \doteq^? t$ or $X \simeq^?_{\mathcal{R},\lambda} t$ (i.e., if X occurs in $\mathcal{K}_0$ once), then $(v_X, v_Y) \in E$ for all $Y \in \mathit{var}(t)$ and $\mathit{mark}(v_X, G)$ is updated as $\mathit{mark}(v_X, G) \setminus \{X\}$. This is how the initial version of the graph is created. It is denoted by $G_{\mathcal{K}_0}$.

In the process of the application of Solve, the graph gets modified as follows:

a) *The applied rule is of the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$.* Then from the graph $G_{\mathcal{K}}$ we obtain the graph $G_{\mathcal{K}'}$ depending on the rule:

- Elim-eq with $X \doteq^? t$ adds edges and removes X from the marking set of the vertex v_X exactly as described above.
- Elim-sim with $X \simeq^?_{\mathcal{R},\lambda} t$ either keeps the graph unchanged (when X occurs more than once in the resulting constraint after the application of Elim-sim), or modifies it as described above (when X occurs exactly once in the resulting constraint after the application of Elim-sim).
- TVE-mix with $X \simeq^?_{\mathcal{R},\lambda} t$ does the same modification as Elim-eq and, in addition, modifies marking: Let $Y_1, \dots, Y_m$, $m \geq 0$, be all the copies of a term variable $Y \in \mathit{var}(t)$ created by the renaming function ρ in the TVE-mix step. Then each Y_i is associated with the vertex v_Y (i.e., $v_{Y_i} = v_Y$) and $\mathit{mark}(v_Y, G)$ gets updated as $\mathit{mark}(v_Y, G) \cup \{Y_1, \dots, Y_m\}$.
- No other rule of the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$ modifies the graph.

b) *The applied rule is of the form $\mathcal{K} \rightsquigarrow \mathcal{K}'_1 \vee \cdots \vee \mathcal{K}'_n$, $n > 1$.* Then $G_{\mathcal{K}} = G_{\mathcal{K}'_1} = \cdots = G_{\mathcal{K}'_n}$.

One can see that the set Vert remains unchanged during the process.

Let $G_1 = (\mathit{Vert}, E_1)$ be the graph before applying a rule, and $G_2 = (\mathit{Vert}, E_2)$ be the one after a rule application so that $G_1 \neq G_2$, i.e., Elim-eq, Elim-sim, or TVE-mix is applied. Let the chosen primitive constraint be $X \doteq^? t$ (for Elim-eq) or $X \simeq^?_{\mathcal{R},\lambda} t$ (for Elim-sim and TVE-mix). Then $E_1 \subseteq E_2$, because new edges from v_X to v_Y for each $Y \in \mathit{var}(t)$ is created and none are removed. Besides, $\mathit{mark}(X, G_1) \supset \mathit{mark}(X, G_2) = \mathit{mark}(X, G_1) \setminus \{X\}$, and markings in G_2 are not changed for any of the vertices which is not reachable from X in G_2. Hence, $G_1 >_{\mathrm{dag}} G_2$.

Let us consider the pair $(G_{\mathcal{K}}, \mathbf{V}_{\mathsf{F}}(\mathcal{K}))$ of such a term variable dependency graph $G_{\mathcal{K}}$ associated to a constraint $\mathcal{K}$ and a set of function variables $\mathbf{V}_{\mathsf{F}}(\mathcal{K})$ occurring in $\mathcal{K}$. These pairs are ordered lexicographically by $>_{\mathrm{dag}}$ and $>$. By Theorem 22, $>_{\mathrm{dag}}$ is well-founded. The relation $>$ on natural numbers is well-founded. Therefore, their lexicographic combination is well-founded. Since Unif and Sim are terminating (Theorems 10 and 12), each iteration of the **while** loop in the definition of Solve either stops with false, or reaches the application of Mix. In this process, measure of the pair $(G_{\mathcal{K}}, \mathbf{V}_{\mathsf{F}}(\mathcal{K}))$ does not increase, because the Unif and Sim rules, as we have already seen, either decrease or keep unchanged $G_{\mathcal{K}}$, and do not add new function variables. The application of Mix either fails, or strictly reduces $(G_{\mathcal{K}}, \mathbf{V}_{\mathsf{F}}(\mathcal{K}))$: TVE-mix strictly decreases $G_{\mathcal{K}}$, and FVE-mix does not change $G_{\mathcal{K}}$ but strictly decreases $\mathbf{V}_{\mathsf{F}}(\mathcal{K})$. Hence, the **while** loop in Solve can be executed only finitely many times.

If Solve does not stop with false, the only possible non-solved primitive constraints are those between variables, whose left hand side has occurrences in at least two different kind of constraints. For any other case, there is an applicable rule. Hence, the obtained constraint is in appr-solved form. ◀

▶ **Theorem 24** (Soundness of Solve). *Let $\mathcal{K}$ be a conjunction of primitive constraints. Then every solution of the constraint* $\mathsf{Solve}(\mathcal{K})$ *is a solution of* $\mathcal{K}$.

Proof. By induction on the length of a rule application sequence leading from $\mathcal{K}$ to $\mathsf{Solve}(\mathcal{K})$, using the soundness lemmas for equality, similarity, and mixed rules (Lemmas 11, 13, 18). ◀

▶ **Theorem 25** (Completeness of Solve). *Let $\mathcal{K}$ be a conjunction of primitive constraints, and ϑ be its solution. Then* $\mathsf{Solve}(\mathcal{K})$ *is a constraint $(\mathcal{K}_{\mathrm{sol}} \wedge \mathcal{K}_{\mathrm{var}}) \vee \mathcal{C}$, where $\mathcal{K}_{\mathrm{sol}} \wedge \mathcal{K}_{\mathrm{var}}$ is in appr-solved form, and $\sigma_{\mathcal{K}_{\mathrm{sol}}}\sigma_{\mathcal{K}_{\mathrm{var}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$, where $\sigma_{\mathcal{K}_{\mathrm{sol}}}$ is the substitution induced by $\mathcal{K}_{\mathrm{sol}}$, and $\sigma_{\mathcal{K}_{\mathrm{var}}}$ is a solution of $\mathcal{K}_{\mathrm{var}}$.*

Proof. In the proof we use completeness of unification and weak unification algorithms [4,9,17]. First, note that if one of the failure rules is applicable to a constraint, then it has no solution. For Occ-mix it follows from Theorem 17. For Mism-mix, it is guaranteed by the fact that symbols with different arities are not similar. For failure rules in Unif and Sim it is known from their completeness results.

Application of Unif to $\mathcal{K}$ leads to a new constraint $\mathcal{C}_{\mathrm{un}}$, which contains a solved form $\mathcal{K}_{\text{un-sol}}$ such that $\sigma_{\mathcal{K}_{\text{un-sol}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. Application of Sim to $\mathcal{C}_{\mathrm{un}}$ gives $\mathcal{C}_{\mathrm{sim}}$, which contains a solved form $\mathcal{K}_{\text{sim-sol}}$ (an extension of $\mathcal{K}_{\text{un-sol}}$) such that $\sigma_{\mathcal{K}_{\text{sim-sol}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. After that, if TVE-mix is applicable, we have an equation $X \simeq^{?}_{\mathcal{R},\lambda} \mathsf{f}(t_1,\ldots,t_n)$. TVE-mix extends the solved form by a new equation $X \doteq^{?} \rho(\mathsf{f}(t_1,\ldots,t_n))$, obtaining $\mathcal{K}_{\text{tve-sol}}$. By definition of ρ, the term $\rho(\mathsf{f}(t_1,\ldots,t_n))$ contains fresh variables for each symbol in $\mathsf{f}(t_1,\ldots,t_n)$ and, hence, $\sigma_{\mathcal{K}_{\text{tve-sol}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. It is important at this step to record which fresh variable is a copy of which original variable, maintaining an function $\mathit{original\text{-}of}(V') = V$, where $V \in \mathit{var}(\mathcal{K})$ and V' is zero or more applications of ρ to it (i.e., V' is V, or its copy, or a copy of its copy etc.). If the rule FVE-mix is applicable, we have an equation $F \simeq^{?}_{\mathcal{R},\lambda} f$. We make a step by this rule, adding a new equation $F \doteq^{?} \mathit{original\text{-}of}(F)\vartheta$ and obtaining a new solved form $\mathcal{K}_{\text{fve-sol}}$. Let φ be the substitution $\{\mathit{original\text{-}of}(F) \mapsto \mathit{original\text{-}of}(F)\vartheta\}$. Then we have $\sigma_{\mathcal{K}_{\text{fve-sol}}}\varphi \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. Iterating this process, we do not get false, since $\mathcal{K}$ was solvable. By Theorem 23, the process terminates with an appr-solved form $\mathcal{K}_{\mathrm{sol}} \wedge \mathcal{K}_{\mathrm{var}}$ such that $\sigma_{\mathcal{K}_{\mathrm{sol}}}\varphi_1 \cdots \varphi_k \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$, where the φ's are substitutions of the form $\{\mathit{original\text{-}of}(V) \mapsto \mathit{original\text{-}of}(V)\vartheta\}$. Let $\sigma_{\mathcal{K}_{\mathrm{var}}}$ be the restriction of ϑ to the variables of $\mathcal{K}_{\mathrm{var}}$. Then $\sigma_{\mathcal{K}_{\mathrm{var}}}$ is a solution of $\mathcal{K}_{\mathrm{var}}$. And we have $\sigma_{\mathcal{K}_{\mathrm{sol}}}\varphi_1 \cdots \varphi_k \sigma_{\mathcal{K}_{\mathrm{var}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$.

For the φ's, composition is commutative, because they are ground substitutions with disjoint domains. For some of them we have $\sigma_{\mathcal{K}_{\mathrm{sol}}}\varphi_i = \sigma_{\mathcal{K}_{\mathrm{sol}}}$, because at some step we might have solved an equation with $\mathit{original\text{-}of}(V)$ variable in its left for $\mathit{original\text{-}of}(V) \in \mathit{dom}(\varphi_i)$. We assume that the φ's in the composition are rearranged so that $\sigma_{\mathcal{K}_{\mathrm{sol}}}\varphi_1 \cdots \varphi_i = \sigma_{\mathcal{K}_{\mathrm{sol}}}\varphi_{i+1} \cdots \varphi_k$. These remaining φ's are those for which the algorithm reached a variables-only equation containing $\mathit{original\text{-}of}(V)$, which occurs in the domain of one of the φ's. But then $\varphi_{i+1} \cdots \varphi_k$ is a part of $\sigma_{\mathcal{K}_{\mathrm{var}}}$. Hence, we can get rid of them, obtaining $\sigma_{\mathcal{K}_{\mathrm{sol}}}\sigma_{\mathcal{K}_{\mathrm{var}}}$. Hence, we get $\sigma_{\mathcal{K}_{\mathrm{sol}}}\sigma_{\mathcal{K}_{\mathrm{var}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. ◀

4 Computing approximation degrees

In the algorithm, we have not included the computation of approximation degrees, but it can be done easily. Instead of constraints in DNF of the form $\mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$, we will be working with expressions (we call them extended constraints) $(K_1, \mathfrak{D}_1) \vee \cdots \vee (K_n, \mathfrak{D}_n)$, where $\mathfrak{D}_1, \ldots, \mathfrak{D}_n$ are approximation degrees. The rules will carry the degree ("computed so far") as an additional parameter, but only two rules would change them: Del-sim and

FVE-mix. Their variants with degree modification would work on constraint-degree pairs ($\uplus$ stands for disjoint union):

$$\begin{aligned}
\mathsf{Del\text{-}sim\text{-}deg}: \quad & \big(\tau_1 \simeq^?_{\mathcal{R},\lambda} \tau_2 \wedge \mathcal{K},\ \{\langle \mathcal{R}, \mathfrak{d}\rangle\} \uplus \mathfrak{D}\big) \rightsquigarrow (\mathcal{K},\ \{\langle \mathcal{R}, \min\{\mathfrak{d}, \mathcal{R}(\tau_1,\tau_2)\}\rangle\} \cup \mathfrak{D}) \\
& \text{where } \tau_1, \tau_2 \in \mathbf{C}_{\mathsf{F}} \cup \mathbf{V}_{\mathsf{F}} \cup \mathbf{V}_{\mathsf{T}} \text{ and } \mathcal{R}(\tau_1,\tau_2) \geq \lambda. \\
\mathsf{FVE\text{-}mix\text{-}deg}: \quad & \big(F \simeq^?_{\mathcal{R},\lambda} f \wedge \mathcal{K},\ \{\langle \mathcal{R}, \mathfrak{d}\rangle\} \uplus \mathfrak{D}\big) \rightsquigarrow \\
& \qquad \textstyle\bigvee_{g \in \mathsf{N}(f,\mathcal{R},\lambda)} \big(F \doteq g \wedge \mathcal{K}\{F \mapsto g\},\ \{\langle \mathcal{R}, \min\{\mathfrak{d}, \mathcal{R}(f,g)\}\rangle\} \cup \mathfrak{D}\big), \\
& \text{where } F \in \mathit{var}(\mathcal{K}).
\end{aligned}$$

For any other rule R of the form $\mathcal{K} \rightsquigarrow \mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$, $n \geq 1$, its degree variant R-deg will have the form $(\mathcal{K}, \mathfrak{D}) \rightsquigarrow (\mathcal{K}_1, \mathfrak{D}) \vee \cdots \vee (\mathcal{K}_n, \mathfrak{D})$, i.e., $\mathfrak{D}$ will not change. Let us denote the corresponding versions of Unif, Sim, and Mix by Unif-deg, Sim-deg, and Mix-deg. The notions of solved and approx-solved forms generalize directly to extended constraints. Then we can define Solve-deg along the lines of Solve: To solve a conjunction of primitive equality and similarity constraints $\mathcal{K}$ with respect to similarity relations $\mathcal{R}_1, \dots, \mathcal{R}_m$, it performs the following steps:

$\mathcal{C} := (\mathcal{K}, \{\langle \mathcal{R}_1, 1\rangle, \dots, \langle \mathcal{R}_m, 1\rangle\})$
while $\mathcal{C}$ is not in the appr-solved form **do**
  $\mathcal{C}$:= Unif-deg($\mathcal{C}$), **if** $\mathcal{C}$ = false, **return** false
  $\mathcal{C}$:= Sim-deg($\mathcal{C}$), **if** $\mathcal{C}$ = false, **return** false
  $\mathcal{C}$:= Mix-deg($\mathcal{C}$), **if** $\mathcal{C}$ = false, **return** false
return $\mathcal{C}$

5 Discussion and summary

The proposed solver can be used in constraint-based formalisms such as, for instance, constraint logic programming [11] or term rewriting with constraints [13]. We can envisage an instance of the CLP schema with constraints over multiple similarities. Without going into much details, a simple constraint logic program below can illustrate this possibility:

▶ **Example 26.** The letter P in the program stands for a predicate variable. In constraints, it is treated as a function variable.

$$\begin{aligned}
& P(X,Y) \leftarrow P \simeq_{\mathcal{R}_1,\lambda_1} p,\ X \simeq_{\mathcal{R}_1,\lambda_1} F(Y),\ Y \simeq_{\mathcal{R}_2,\lambda_2} c,\ r(X),\ r(F(Y)). \\
& r(F(X)) \leftarrow F \simeq_{\mathcal{R}_2,\lambda_2} h,\ X \simeq_{\mathcal{R}_1,\lambda_1} a.
\end{aligned}$$

Assume $\mathcal{R}_1(p,q) = 0.9$, $\mathcal{R}_1(a,b) = 0.8$, $\mathcal{R}_1(f,g) = 0.6$, $\mathcal{R}_2(g,h) = 0.5$, $\mathcal{R}_2(b,c) = 0.7$, $\lambda_1 = \lambda_2 = 0.4$. Then by performing the usual CLP inference (i.e., syntactically unifying the selected query and the head of the corresponding clause) for the query $\leftarrow q(X,Y)$, the resulting constraint (together with the approximation degrees) will be $(X \doteq g(b) \wedge Y \doteq b;\ \{\langle \mathcal{R}_1, 0.8\rangle, \langle \mathcal{R}_2, 0.5\rangle\}) \vee (X \doteq h(b) \wedge Y \doteq b; \{\langle \mathcal{R}_1, 0.8\rangle, \langle \mathcal{R}_2, 0.7\rangle\})$.

To summarize, the algorithm Solve presented in the paper solves positive equational and similarity constraints, where multiple similarity relations are permitted. Given such a constraint in DNF, it computes disjunction of approximately solved forms, from which solution substitutions can be read off. It can be easily extended to include the computation of approximation degrees of the solutions. The algorithm is terminating, sound, and complete.

References

1. Hassan Aït-Kaci and Gabriella Pasi. Fuzzy unification and generalization of first-order terms over similar signatures. In Fabio Fioravanti and John 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. URL: `https://doi.org/10.1007/978-3-319-94460-9_13`, `doi:10.1007/978-3-319-94460-9\_13`.
2. Hassan Aït-Kaci and Gabriella Pasi. Lattice operations on terms with fuzzy signatures. *CoRR*, abs/1709.00964, 2017. URL: `http://arxiv.org/abs/1709.00964`, `arXiv:1709.00964`.
3. Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.
4. Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.
5. Francesca Arcelli Fontana and Ferrante Formato. Likelog: A logic programming language for flexible data retrieval. In Barrett R. Bryant, Gary B. Lamont, Hisham Haddad, and Janice H. Carroll, editors, *Proceedings of the 1999 ACM Symposium on Applied Computing, SAC'99, San Antonio, Texas, USA, February 28 - March 2, 1999*, pages 260–267. ACM, 1999. URL: `https://doi.org/10.1145/298151.298348`, `doi:10.1145/298151.298348`.
6. Francesca Arcelli Fontana and Ferrante Formato. A similarity-based resolution rule. *Int. J. Intell. Syst.*, 17(9):853–872, 2002. URL: `https://doi.org/10.1002/int.10067`, `doi:10.1002/int.10067`.
7. Ferrante Formato, Giangiacomo Gerla, and Maria I. Sessa. Extension of logic programming by similarity. In Maria Chiara Meo and Manuel Vilares Ferro, editors, *1999 Joint Conference on Declarative Programming, AGP'99, L'Aquila, Italy, September 6-9, 1999*, pages 397–410, 1999.
8. Ferrante Formato, Giangiacomo Gerla, and Maria I. Sessa. Similarity-based unification. *Fundam. Inform.*, 41(4):393–414, 2000. URL: `https://doi.org/10.3233/FI-2000-41402`, `doi:10.3233/FI-2000-41402`.
9. Pascual Julián Iranzo and Clemente Rubio-Manzano. A sound and complete semantics for a similarity-based logic programming language. *Fuzzy Sets and Systems*, 317:1–26, 2017. URL: `https://doi.org/10.1016/j.fss.2016.12.016`, `doi:10.1016/j.fss.2016.12.016`.
10. Pascual Julián Iranzo and Fernando Sáenz-Pérez. An efficient proximity-based unification algorithm. In *2018 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–8. IEEE, 2018. URL: `https://doi.org/10.1109/FUZZ-IEEE.2018.8491593`, `doi:10.1109/FUZZ-IEEE.2018.8491593`.
11. Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994. URL: `https://doi.org/10.1016/0743-1066(94)90033-7`, `doi:10.1016/0743-1066(94)90033-7`.
12. Pascual Julián-Iranzo and Clemente Rubio-Manzano. Proximity-based unification theory. *Fuzzy Sets and Systems*, 262:21–43, 2015. URL: `https://doi.org/10.1016/j.fss.2014.07.006`, `doi:10.1016/j.fss.2014.07.006`.
13. Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2013. URL: `https://doi.org/10.1007/978-3-642-40885-4_24`, `doi:10.1007/978-3-642-40885-4\_24`.
14. Stanislav Krajci, Rastislav Lencses, Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtás. A similarity-based unification model for flexible querying. In Troels Andreasen, Amihai Motro, Henning Christiansen, and Henrik Legind Larsen, editors, *Flexible Query Answering Systems, 5th International Conference, FQAS 2002, Copenhagen, Denmark, October 27-29, 2002, Proceedings*, volume 2522 of *Lecture Notes in Computer Science*, pages 263–273. Springer, 2002. URL: `https://doi.org/10.1007/3-540-36109-X_21`, `doi:10.1007/3-540-36109-X\_21`.

15 Temur Kutsia and Cleo Pau. Solving proximity constraints. In Maurizio Gabbrielli, editor, *Logic-Based Program Synthesis and Transformation - 29th International Symposium, LOPSTR 2019, Porto, Portugal, October 8-10, 2019, Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-45260-5_7, doi:10.1007/978-3-030-45260-5_7.

16 Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtás. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146(1):43–62, 2004. URL: https://doi.org/10.1016/j.fss.2003.11.005, doi:10.1016/j.fss.2003.11.005.

17 Maria I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theor. Comput. Sci.*, 275(1-2):389–426, 2002. URL: https://doi.org/10.1016/S0304-3975(01)00188-8, doi:10.1016/S0304-3975(01)00188-8.

18 Mariya I. Vasileva, Bryan A. Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar, and David A. Forsyth. Learning type-aware embeddings for fashion compatibility. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, volume 11220 of *Lecture Notes in Computer Science*, pages 405–421. Springer, 2018. URL: https://doi.org/10.1007/978-3-030-01270-0_24, doi:10.1007/978-3-030-01270-0_24.

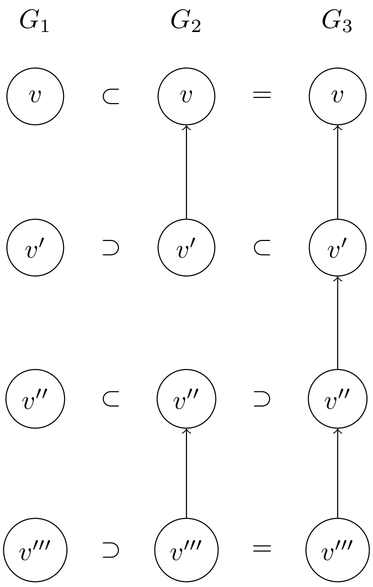
19 Andreas Veit, Serge J. Belongie, and Theofanis Karaletsos. Conditional similarity networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1781–1789. IEEE Computer Society, 2017. URL: https://doi.org/10.1109/CVPR.2017.193, doi:10.1109/CVPR.2017.193.

20 Harry Virtanen. Vague domains, s-unification, logic programming. *Electr. Notes Theor. Comput. Sci.*, 66(5):86–103, 2002. URL: https://doi.org/10.1016/S1571-0661(04)80516-4, doi:10.1016/S1571-0661(04)80516-4.

21 Peter Vojtás. Declarative and procedural semantics of fuzzy similarity based unification. *Kybernetika*, 36(6):707–720, 2000. URL: http://www.kybernetika.cz/content/2000/6/707.

A Intuitive explanation of a part of the proof of Theorem 3

In the proof of Theorem 3, case **i**, we assumed $v \notin M(G_1, G_2)$ and reached a contradiction. The reasoning line there can be illustrated by the diagram below, where we step by step construct the vertices v', v'', v''' until we reach a contradiction with another assumption $v \in M(G_1, G_3)$. The symbols $\subset$, $\supset$, $=$ between the vertices show the relation between the marked sets of those vertices. The leftmost graph is a part of G_1, the middle one of G_2, and the rightmost one of G_3.



$$mark(v, G_1) \subset mark(v, G_2) = mark(v, G_3)$$

$$mark(v, G_1) \supset mark(v, G_2) \subset mark(v, G_3)$$

$$mark(v, G_1) \subset mark(v, G_2) \supset mark(v, G_3)$$

$$mark(v, G_1) \supset mark(v, G_2) = mark(v, G_3)$$