

1 Constraint solving over multiple similarity relations

2 Besik Dundua

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

6 Temur Kutsia

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

10 Mircea Marin

11 West University of Timișoara, Romania
12 <http://staff.fmi.uvt.ro/~mircea.marin/>
13 mircea.marin@e-uvt.ro

14 Cleopatra Pau

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

17 — Abstract —

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

28 **2012 ACM Subject Classification** Theory of computation → Logic; Computing methodologies →
29 Symbolic and algebraic manipulation; Theory of computation → Semantics and reasoning.

30 **Keywords and phrases** Fuzzy relations, similarity, constraint solving.

31 **Digital Object Identifier** 10.4230/LIPIcs.FSCD.2020.35

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

35 1 Introduction

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

41 For many problems in this area, exact equality is replaced by its approximation. Several
42 approaches use similarity relations to express the approximation, modeling the corresponding
43 imprecise information. Similarity relations are fuzzy binary relations, specifying to which
44 degree two objects are similar to each other. They satisfy reflexivity, symmetry, and fuzzy
45 min-transitivity properties, and can be characterized by “level-wise” equivalence. Once the



© B. Dundua, T. Kutsia, M. Marin, and C. Pau;
licensed under Creative Commons License CC-BY

5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020).

Editor: Zena M. Ariola; Article No. 35; pp. 35:1–35:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 $X \simeq_{\mathcal{R}_1, 0.4} \textit{white-circle}$ and $X \simeq_{\mathcal{R}_2, 0.5} \textit{gray-ellipse}$. The expected answer is $X = \textit{gray-circle}$. 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: $X = \textit{gray-circle}$ and $X = \textit{white-ellipse}$. On the other hand, the algorithm proposed in this paper computes the right solution $X = \textit{gray-circle}$. 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

93 for terms, we permit also variables for function symbols, since they are necessary in the
94 process of finding an “intermediate object” between terms in different similarity relations.

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

100 2 Preliminaries

101 Similarity relations

102 We define basic notions about similarity relations following [9, 17]. A binary *fuzzy relation*
103 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
104 and λ is a number $0 < \lambda \leq 1$ (called *cut value*), then the λ -*cut* of \mathcal{R} on S , denoted \mathcal{R}_λ , is an
105 ordinary (crisp) relation on S defined as $\mathcal{R}_\lambda := \{(s_1, s_2) \mid \mathcal{R}(s_1, s_2) \geq \lambda\}$.

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

107 **Reflexivity:** $\mathcal{R}(s, s) = 1$ for all $s \in S$;

108 **Symmetry:** $\mathcal{R}(s_1, s_2) = \mathcal{R}(s_2, s_1)$ for all $s_1, s_2 \in S$.

109 Let \wedge be a T-norm: an associative, commutative, non-decreasing binary operation on
110 $[0, 1]$ with 1 as the unit element. A proximity relation (on S) is called a *similarity relation*
111 (on S) iff it is transitive:

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

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

115 Terms, atoms, substitutions

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

- 117 ■ \mathbf{V}_T : term variables, denoted by X, Y, Z ,
- 118 ■ \mathbf{V}_F : function variables, denoted by F, G, H ,
- 119 ■ \mathbf{C}_F : function constants, denoted by f, g, h ,

120 By \mathbf{V} we denote the set of variables $\mathbf{V} = \mathbf{V}_T \cup \mathbf{V}_F$, and V is used for its elements.

121 A *function symbol* is a function variable or a function constant, i.e., an element of the set
122 $\mathbf{F} = \mathbf{C}_F \cup \mathbf{V}_F$. We use the letters f, g, h to denote function symbols. Each function symbol
123 has a fixed arity.

124 *Terms* over \mathbf{A} are defined by the grammar $t := X \mid f(t_1, \dots, t_n)$, where f is an n -ary
125 function symbol. For terms we use the letters t, s, r . The set of terms over \mathbf{A} is denoted by
126 $\text{Terms}(\mathbf{A})$.

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

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

- 130 ■ $\sigma(X) \in \text{Terms}(\mathbf{A})$ for all $X \in \mathbf{V}_T$,
- 131 ■ $\sigma(F) \in \mathbf{F}$ for all $F \in \mathbf{V}_F$,
- 132 ■ $\sigma(V) = V$ for all but finitely many variables $V \in \mathbf{V}$.

35:4 Constraint solving over multiple similarity relations

133 Substitutions are denoted by Greek letters $\sigma, \vartheta, \varphi$. The identity substitution is denoted
 134 by Id . The domain of a substitution σ is the set $dom(\sigma) = \{V \mid V \in \mathbf{V}, \sigma(V) \neq V\}$. The
 135 *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
 136 $V \in \mathcal{V}$ and $\sigma|_{\mathcal{V}}(V) = V$ otherwise. We will use the usual set representation of substitutions,
 137 writing σ as $\{V \mapsto \sigma(V) \mid V \in dom(\sigma)\}$.

138 *Substitution application* to variables, constants, and terms is defined as follows: $c\sigma = c$
 139 for all $c \in \mathbf{C}_F$, $V\sigma = \sigma(V)$ for all $V \in \mathbf{V}$, and $f(t_1, \dots, t_n)\sigma = (f\sigma)(t_1\sigma, \dots, t_n\sigma)$.

140 Similarity relations on syntactic domains

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

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

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

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

155 Constraints

156 In our constraint language, the elements of $\mathbf{F} \cup \mathbf{Terms}(\mathbf{A})$ are the basic objects. In the rest of
 157 the paper, the letter τ is used to denote its elements. Besides, we have the equality predicate
 158 constant \doteq (interpreted as syntactic equality), one or more similarity predicate constants
 159 $\simeq_1, \simeq_2, \dots$, (interpreted as similarity relations on $\mathbf{F} \cup \mathbf{Terms}(\mathbf{A})$), propositional constants
 160 **true** and **false**, connectives \wedge, \vee , and the quantifier \exists .

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

$$162 \quad \mathcal{P} ::= \text{true} \mid \text{false} \mid t \doteq s \mid t \simeq s \mid f \doteq g \mid f \simeq g,$$

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

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

173 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
 174 σ is *more general* than ϑ on a set of VP-pairs \mathcal{W} iff there exists a substitution φ such that
 175 $\mathcal{R}(V\sigma\varphi, V\vartheta) \geq \lambda$ 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
 176 write $\sigma \preceq_{\mathcal{W}} \vartheta$.

177 ► **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
 178 $\mathcal{W} = \{\langle X, \simeq_{\mathcal{R}_1, 0.5} \rangle, \langle Y, \simeq_{\mathcal{R}_1, 0.6} \rangle, \langle Y, \simeq_{\mathcal{R}_2, 0.7} \rangle\}$.

179 ■ Let $\sigma = \{X \mapsto Y\}$ and $\vartheta = \{X \mapsto a, Y \mapsto b\}$. Then $\sigma \preceq_{\mathcal{W}} \vartheta$, because for $\varphi = \{X \mapsto b,$
 180 $Y \mapsto b\}$ 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 $Y\sigma\varphi =$
 181 $b \simeq_{\mathcal{R}_2, 0.7} b = Y\vartheta$.

182 ■ Let $\sigma = \{X \mapsto Y\}$ and $\vartheta = \{X \mapsto a, Y \mapsto c\}$. Then $\sigma \preceq_{\mathcal{W}} \vartheta$, because for $\varphi = \{X \mapsto b,$
 183 $Y \mapsto b\}$ 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 $Y\sigma\varphi =$
 184 $b \simeq_{\mathcal{R}_2, 0.7} c = Y\vartheta$.

185 ■ Let $\sigma = \{X \mapsto f(Y), Y \mapsto Z\}$ and $\vartheta = \{X \mapsto f(Z), Y \mapsto a, Z \mapsto X\}$. Then $\sigma \preceq_{\mathcal{W}} \vartheta$,
 186 because for $\varphi = \{Y \mapsto Z, Z \mapsto a\}$ we have $X\sigma\varphi = f(Z) \simeq_{\mathcal{R}_1, 0.5} f(Z) = X\vartheta$, $Y\sigma\varphi =$
 187 $a \simeq_{\mathcal{R}_1, 0.6} a = Y\vartheta$, and $Y\sigma\varphi = a \simeq_{\mathcal{R}_2, 0.7} a = Y\vartheta$.

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

189 **Proof.** Reflexivity is obvious. For transitivity, assume $\sigma_1 \preceq_{\mathcal{W}} \sigma_2$ and $\sigma_2 \preceq_{\mathcal{W}} \sigma_3$. We will
 190 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 $\mathcal{R}(V\sigma_1\varphi_1, V\sigma_2) \geq \lambda$
 191 and $\mathcal{R}(V\sigma_2\varphi_2, V\sigma_3) \geq \lambda$. Since similarity is stable for substitutions [17, Proposition 3.1],
 192 we have $\mathcal{R}(V\sigma_1\varphi_1\varphi_2, V\sigma_2\varphi_2) \geq \lambda$. By transitivity of similarity, we get $\mathcal{R}(V\sigma_1\varphi_1\varphi_2, V\sigma_3) \geq$
 193 $\min(\mathcal{R}(V\sigma_1\varphi_1\varphi_2, V\sigma_2\varphi_2), \mathcal{R}(V\sigma_2\varphi_2, V\sigma_3)) \geq \lambda$, which implies that $\sigma_1 \preceq_{\mathcal{W}} \sigma_3$. ◀

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

195 The notation $\mathcal{K}_{\underline{\quad}}$ denotes a conjunction of primitive equality constraints. By $\mathcal{K}_{\mathcal{R}, \lambda}$ we
 196 denote a conjunction of primitive similarity constraints, all with the same relation \mathcal{R} and the
 197 same λ -cut: $\mathcal{K}_{\mathcal{R}, \lambda} = \tau_1 \simeq_{\mathcal{R}, \lambda} \tau'_1 \wedge \dots \wedge \tau_n \simeq_{\mathcal{R}, \lambda} \tau'_n$.

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

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

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

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

204 A substitution σ is a solution of a conjunction of primitive constraints \mathcal{K} iff it solves each
 205 primitive constraint in \mathcal{K} . We denote the set of all solutions of \mathcal{K} by $\text{Sol}(\mathcal{K})$. For a constraint
 206 $\mathcal{C} = \mathcal{K}_1 \wedge \dots \wedge \mathcal{K}_n$ in disjunctive normal form (DNF), we define $\text{Sol}(\mathcal{C}) = \cup_{i=1}^n \text{Sol}(\mathcal{K}_i)$.

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

- 210 ■ $\mathcal{K} = \text{true}$ or $\mathcal{K} = \mathcal{K}_{\underline{\quad}}$ and $\mathfrak{d}_1 = \dots = \mathfrak{d}_n = 1$,
- 211 ■ $\mathcal{K} = \tau_1 \simeq_{\mathcal{R}_j, \lambda_j} \tau_2$ 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
 212 $1 \leq i \leq n, i \neq j$,
- 213 ■ $\mathcal{K} = \mathcal{P} \wedge \mathcal{K}'$, σ solves \mathcal{P} and \mathcal{K}' with approximation degrees $\{\langle \mathcal{R}_1, \mathfrak{d}_1^{\mathcal{P}} \rangle, \dots, \langle \mathcal{R}_n, \mathfrak{d}_n^{\mathcal{P}} \rangle\}$ and
 214 $\{\langle \mathcal{R}_1, \mathfrak{d}'_1 \rangle, \dots, \langle \mathcal{R}_n, \mathfrak{d}'_n \rangle\}$, respectively, and $\mathfrak{d}_i = \min\{\mathfrak{d}_i^{\mathcal{P}}, \mathfrak{d}'_i\}$ for all $1 \leq i \leq n$.

215 Such a definition of approximation degrees gives the flexibility to characterize approxima-
 216 tions with respect to each involved relation independently from each other.

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

35:6 Constraint solving over multiple similarity relations

219 **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
 220 φ such that $\mathcal{R}(V\sigma\varphi, V\vartheta) \geq \lambda_i$ for each $V \in \text{var}(\mathcal{K}_{\mathcal{R}_i, \lambda_i})$. It implies that

$$221 \quad \mathcal{R}(s_j\sigma\varphi, s_j\vartheta) \geq \lambda_i, \quad j = 1, 2. \quad (1)$$

223 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
 224 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
 225 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
 226 a solution of $s_1 \simeq_{\mathcal{R}_i, \lambda_i} s_2$.

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

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

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

238 Appr-solved forms are also solved forms, but not vice versa. Each solved form \mathcal{K} induces
 239 a substitution, denoted by $\sigma_{\mathcal{K}}$: if $\mathcal{K} = \text{true}$, then $\sigma_{\mathcal{K}} = \text{Id}$, otherwise $\sigma_{\mathcal{K}} = \{V \mapsto \tau \mid V \doteq$
 240 $\tau \in \mathcal{K} \text{ or } V \simeq_{\mathcal{R}, \lambda} \tau \in \mathcal{K}\}$. Obviously, $\sigma_{\mathcal{K}}$ is a solution of \mathcal{K} . A constraint $\mathcal{K}_{\text{sol}} \wedge \mathcal{K}_{\text{var}}$ in
 241 appr-solved form is also solvable, because $\sigma_{\mathcal{K}_{\text{sol}}}$ solves \mathcal{K}_{sol} and \mathcal{K}_{var} always has at least
 242 a trivial solution mapping all terms variables to the same term variable and all function
 243 variables to the same function variable.

244 **► 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
 245 constraint, where $\mathcal{K}_{\mathcal{R}_1, 0.4} = X \simeq_{\mathcal{R}_1, 0.4} \text{white-circle} \wedge X \simeq_{\mathcal{R}_1, 0.4} Y$ and $\mathcal{K}_{\mathcal{R}_2, 0.5} = X \simeq_{\mathcal{R}_2, 0.5}$
 246 $\text{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2, 0.5} \text{white-ellipse}$.

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

$$250 \quad X \simeq_{\mathcal{R}_1, 0.4} \text{white-circle} \wedge Y \simeq_{\mathcal{R}_1, 0.4} \text{white-circle} \wedge$$

$$251 \quad X \simeq_{\mathcal{R}_2, 0.5} \text{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2, 0.5} \text{white-ellipse},$$

253 which is not yet in a solved form. A solved form, equivalent to \mathcal{K} , would be $X \doteq \text{gray-circle} \wedge$
 254 $Y \doteq \text{white-circle}$. It induces the substitution $\sigma = \{X \mapsto \text{gray-circle}, Y \mapsto \text{white-circle}\}$.

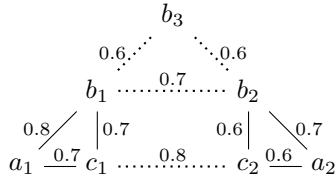
255 **► Example 8.** Let \mathcal{R}_1 and \mathcal{R}_2 be two similarity relations defined as

$$256 \quad \mathcal{R}_1 : \quad \mathcal{R}_1(a_1, c_1) = \mathcal{R}_1(b_1, c_1) = 0.7, \quad \mathcal{R}_1(a_1, b_1) = 0.8,$$

$$257 \quad \mathcal{R}_1(a_2, c_2) = \mathcal{R}_1(b_2, c_2) = 0.6, \quad \mathcal{R}_1(a_2, b_2) = 0.7,$$

$$258 \quad \mathcal{R}_2 : \quad \mathcal{R}_2(b_1, b_3) = \mathcal{R}_2(b_2, b_3) = 0.6, \quad \mathcal{R}_2(b_1, b_2) = 0.7, \quad \mathcal{R}_2(c_1, c_2) = 0.8.$$

260 Visually:



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

261

262 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
 263 of two solved forms, e.g., $(X \doteq f(b_1, b_2) \wedge Y \simeq_{\mathcal{R}_2, 0.6} b_1) \vee (X \doteq f(c_1, c_2) \wedge Y \simeq_{\mathcal{R}_2, 0.6} c_1)$.
 264 The solved forms induce two substitutions: $\sigma_1 = \{X \mapsto f(b_1, b_2), Y \mapsto b_1\}$ and $\sigma_2 = \{X \mapsto$
 265 $f(c_1, c_2), Y \mapsto c_1\}$. They are solutions of \mathcal{K} . There are other solutions of \mathcal{K} that are
 266 \cong -equivalent to σ_1 or σ_2 : $\vartheta_1 = \{X \mapsto f(b_1, b_2), Y \mapsto b_2\} \cong \sigma_1$, $\vartheta_2 = \{X \mapsto f(b_1, b_2), Y \mapsto$
 267 $b_3\} \cong \sigma_1$, and $\vartheta_3 = \{X \mapsto f(c_1, c_2), Y \mapsto c_2\} \cong \sigma_2$.

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

276 **► Example 9.** Let $\mathcal{R}_1(a, b_1) = \mathcal{R}_1(b_1, b_2) = \mathcal{R}_1(a, b_2) = 0.8$, $\mathcal{R}_2(c, b_1) = \mathcal{R}_2(b_1, b_2) =$
 277 $\mathcal{R}_2(b_2, c) = 0.7$ 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
 278 straightforward solved form $X \doteq f(Z, Z) \wedge Y \doteq Z$, as in the previous example, has fewer
 279 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
 280 form $\mathcal{K}_{as} = X \doteq f(X_1, X_2) \wedge X_1 \simeq_{\mathcal{R}_1, 0.6} Y \wedge X_1 \simeq_{\mathcal{R}_2, 0.5} Z \wedge X_2 \simeq_{\mathcal{R}_1, 0.6} Y \wedge X_2 \simeq_{\mathcal{R}_2, 0.5} Z$,
 281 then all solutions of \mathcal{K}_{as} solve \mathcal{K} and for each solution σ of \mathcal{K} , we have $(\exists X_1, X_2. \mathcal{K}_{as})\sigma$.
 282 Unlike the previous example, we can not turn this appr-solved form into a solved form by
 283 swapping sides of variables-only equations.

284 3 Constraint solving

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

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

293 Equality rules

294 In this subsection we describe the rules that solve equality constraints. Essentially, these are
 295 first-order unification rules with a slight modification, which concerns dealing with function
 296 and predicate variables. The rules have the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$, which defines the transformation
 297 $\mathcal{K} \vee \mathcal{C} \rightsquigarrow \mathcal{K}' \vee \mathcal{C}$. Note that \mathcal{C} does not change.

298 The rules are `Del-eq` (deletion), `Dec-eq` (decomposition), `Ori-eq` (orientation), `Elim-eq`
 299 (variable elimination), `Confl-eq` (conflict), `Mism-eq` (arity mismatch), `Occ-eq` (occurrence

35:8 Constraint solving over multiple similarity relations

300 check), all formulated for the equality relation \doteq .

301 Del-eq : $\tau \doteq^? \tau \wedge \mathcal{K} \rightsquigarrow \mathcal{K}$, where $\tau \in \mathbf{C}_F \cup \mathbf{V}_F \cup \mathbf{V}_T$.

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

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

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

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

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

308 Occ-eq : $X \doteq^? t \wedge \mathcal{K} \rightsquigarrow \text{false}$, if $X \in \text{var}(t)$ and $X \neq t$.
309

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

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

314 ► **Theorem 10.** *Unif is terminating.*

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

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

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

322 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

$$\begin{aligned} \sigma \in \text{Sol}(\mathcal{K}) \text{ iff } \sigma \in \text{Sol}(\{V \doteq^? \tau\} \wedge \mathcal{K}_0) \text{ iff} \\ V\sigma = \tau\sigma \wedge \sigma \in \text{Sol}(\mathcal{K}_0) \text{ iff (because } V\sigma = \tau\sigma \text{ implies } \sigma = \vartheta\sigma) \\ V\sigma = \tau\sigma \wedge \vartheta\sigma \in \text{Sol}(\mathcal{K}_0) \text{ iff} \\ 323 V\sigma = \tau\sigma \wedge \sigma \in \text{Sol}(\mathcal{K}_0\vartheta) \text{ iff} \\ \sigma \in \text{Sol}(\{V \doteq^? \tau\} \wedge \mathcal{K}_0\vartheta) \text{ iff} \\ \sigma \in \text{Sol}(\mathcal{K}'). \end{aligned}$$

324 ◀

325 Similarity rules

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

329 In the Elim-sim rule, the variable V is replaced by τ only in the constraints for the same
330 similarity relation. This is justified by the fact that although a λ -cut of each similarity
331 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
332 similarity between t and r .

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

336 Del-sim : $\tau_1 \simeq_{\mathcal{R},\lambda}^? \tau_2 \wedge \mathcal{K} \rightsquigarrow \mathcal{K}$, where $\tau_1, \tau_2 \in \mathbf{C}_F \cup \mathbf{V}_F \cup \mathbf{V}_T$ and $\mathcal{R}(\tau_1, \tau_2) \geq \lambda$.

337 Dec-sim : $f(t_1, \dots, t_n) \simeq_{\mathcal{R},\lambda}^? g(s_1, \dots, s_n) \wedge \mathcal{K} \rightsquigarrow$
 338 $f \simeq_{\mathcal{R},\lambda}^? g \wedge t_1 \simeq_{\mathcal{R},\lambda}^? s_1 \wedge \dots \wedge t_n \simeq_{\mathcal{R},\lambda}^? s_n \wedge \mathcal{K}$, where $n > 0$.

339 Ori-sim : $\tau \simeq_{\mathcal{R},\lambda}^? V \wedge \mathcal{K} \rightsquigarrow V \simeq_{\mathcal{R},\lambda}^? \tau \wedge \mathcal{K}$, where $\tau \notin \mathbf{V}$.

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

342 Confl-sim : $\tau_1 \simeq_{\mathcal{R},\lambda}^? \tau_2 \wedge \mathcal{K} \rightsquigarrow \text{false}$, where $\tau_1, \tau_2 \in \mathbf{C}_F \cup \mathbf{V}_F \cup \mathbf{V}_T$, and $\mathcal{R}(\tau_1, \tau_2) < \lambda$.

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

344 Occ-sim : $X \simeq_{\mathcal{R},\lambda}^? t \wedge \mathcal{K} \rightsquigarrow \text{false}$, if $X \in \text{var}(t)$ and $X \neq t$.
 345

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

348 Termination of `Sim` can be proved as termination of `Unif`:

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

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

352 **Proof.** When we have only one similarity relation, soundness follows from soundness of
 353 weak unification algorithm [17]. For the extension to multiple similarity relations, the only
 354 nontrivial rule is `Elim-sim`. (For the others, $\text{Sol}(\mathcal{K}) = \text{Sol}(\mathcal{K}')$ holds directly.) It is important
 355 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
 356 have $\text{Sol}(V \simeq_{\mathcal{R},\lambda}^? \tau \wedge \mathcal{K}_{\mathcal{R},\lambda}) = \text{Sol}(V \simeq_{\mathcal{R},\lambda}^? \tau \wedge \mathcal{K}_{\mathcal{R},\lambda}\{V \mapsto \tau\})$. (It follows from soundness
 357 of weak unification algorithm [17], since the constraint is over a single similarity relation.)
 358 Constraints for all other relations remain unchanged. It implies that the solution sets for
 359 constraints in both sides of the `Elim-sim` rule are the same. ◀

360 ▶ **Example 14.** let $\mathcal{K}_{\mathcal{R}_1,0.4} = X \simeq_{\mathcal{R}_1,0.4} \text{white-circle} \wedge X \simeq_{\mathcal{R}_1,0.4} Y$ and $\mathcal{K}_{\mathcal{R}_2,0.5} = X \simeq_{\mathcal{R}_2,0.5}$
 361 $\text{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \text{white-ellipse}$ be the constraints from Example 7. The reduction
 362 mentioned in that example is modeled by performing the `Elim-sim` step and replacing X by
 363 white-circle in $\mathcal{K}_{\mathcal{R}_1,0.4}$:

364 $X \simeq_{\mathcal{R}_1,0.4} \text{white-circle} \wedge X \simeq_{\mathcal{R}_1,0.4} Y \wedge$
 365 $X \simeq_{\mathcal{R}_2,0.5} \text{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \text{white-ellipse} \rightsquigarrow_{\text{Elim-sim}}$
 366 $X \simeq_{\mathcal{R}_1,0.4} \text{white-circle} \wedge \text{white-circle} \simeq_{\mathcal{R}_1,0.4} Y \wedge$
 367 $X \simeq_{\mathcal{R}_2,0.5} \text{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \text{white-ellipse}.$
 368

369 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

370 $X \simeq_{\mathcal{R}_1,0.4} \text{white-circle} \wedge \text{white-circle} \simeq_{\mathcal{R}_1,0.4} Y \wedge$
 371 $\text{white-circle} \simeq_{\mathcal{R}_2,0.5} \text{gray-ellipse} \wedge Y \simeq_{\mathcal{R}_2,0.5} \text{white-ellipse},$
 372

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

374 **Mixed rules**

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

377 ► **Definition 15** (Occurrence cycle). *An occurrence cycle for a variable X_1 is called the*
378 *conjunction of primitive constraints $X_1 \simeq_{\mathcal{R}_1, \lambda_1}^? t_1[X_2] \wedge X_2 \simeq_{\mathcal{R}_2, \lambda_2}^? t_2[X_3] \wedge \dots \wedge X_n \simeq_{\mathcal{R}_n, \lambda_n}^?$*
379 *$t_n[X_1]$, 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*
380 *variable.*

381 ► **Remark 16.** Note that in the definition of occurrence cycle, if two neighboring primitive
382 similarity constraints use the same relation, they can be contracted into one constraint
383 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
384 $X_i \simeq_{\mathcal{R}_i, \lambda_i}^? t_i[t_{i+1}[X_{i+2}]]$, getting rid of consecutive identical similarity relations. The same
385 is true for the last and the first constraints.

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

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

391 Assume by contradiction that the given occurrence cycle has a solution ϑ . It means that
392 the following term pairs have the same structure: $X_1\vartheta$ and $t_1[X_2]\vartheta$, $X_2\vartheta$ and $t_2[X_3]\vartheta$, ...,
393 $X_n\vartheta$ and $t_n[X_1]\vartheta$. Then $X_1\vartheta$ and $t_1[t_2[\dots[t_n[X_1]]\dots]]\vartheta$ have the same structure. Since at
394 least one of t_i 's is not a variable, $X_1\vartheta$ is a proper subterm of $t_1[t_2[\dots[t_n[X_1]]\dots]]\vartheta$. But a
395 term and its proper subterm can not have the same structure. A contradiction.

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

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

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

408 In all the rules it is assumed that the constraint to be transformed (i.e., the constraint
409 in the left hand side of \rightsquigarrow) has the form $\mathcal{K}_{\cdot} \wedge \mathcal{K}_{\mathcal{R}_1, \lambda_1} \wedge \dots \wedge \mathcal{K}_{\mathcal{R}_m, \lambda_m}$, where \mathcal{K}_{\cdot} and each
410 $\mathcal{K}_{\mathcal{R}_i, \lambda_i}$, $1 \leq i \leq m$, are in *solved form*.

411 The TVE-mix rule uses the *renaming function* ρ . Applied to a term, ρ gives its fresh copy,
412 obtained by replacing each occurrence of a constant from \mathbf{C}_F by a new function variable,
413 each occurrence of a term variable by a fresh term variable, and each occurrence of a function
414 variable by a fresh function variable. For instance, if the term is $f(F(a, X, X, f(a)))$, we have
415 $\rho(f(F(a, X, X, f(a)))) = G_1(G_2(G_3(), Y_1, Y_2, G_4(G_5())))$, where $G_1, G_2, G_3, G_4, G_5 \in \mathbf{V}_F$ are
416 new function variables and $Y_1, Y_2 \in \mathbf{V}_T$ are new term variables.

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

418 Mism-mix : $X \simeq_{\mathcal{R}_1, \lambda_1}^? f(t_1, \dots, t_n) \wedge X \simeq_{\mathcal{R}_2, \lambda_2}^? g(s_1, \dots, s_m) \wedge \mathcal{K} \rightsquigarrow \text{false}$,
 419 if $\mathcal{R}_1 \neq \mathcal{R}_2$ and $m \neq n$.

420 TVE-mix : $X \simeq_{\mathcal{R}, \lambda}^? f(t_1, \dots, t_n) \wedge \mathcal{K} \rightsquigarrow$
 421 $X \doteq F(t'_1, \dots, t'_n) \wedge F \simeq_{\mathcal{R}, \lambda}^? f \wedge t'_1 \simeq_{\mathcal{R}, \lambda}^? t_1 \wedge \dots \wedge t'_n \simeq_{\mathcal{R}, \lambda}^? t_n \wedge \mathcal{K}\vartheta$,
 422 where $X \in \text{var}(\mathcal{K})$, $X \simeq_{\mathcal{R}_1, \lambda_1}^? f(t_1, \dots, t_n) \wedge \mathcal{K}$ does not contain an oc-
 currence cycle for X , $F(t'_1, \dots, t'_n) = \rho(f(t_1, \dots, t_n))$, and $\vartheta = \{X \mapsto$
 $F(t'_1, \dots, t'_n)\}$.

423 FVE-mix : $F \simeq_{\mathcal{R}, \lambda}^? f \wedge \mathcal{K} \rightsquigarrow \bigvee_{g \in \mathcal{N}(f, \mathcal{R}, \lambda)} (F \doteq g \wedge \mathcal{K}\{F \mapsto g\})$, where $F \in \text{var}(\mathcal{K})$.

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

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

428 **Proof.** For failing rules it is trivial as false has no solution. For FVE-mix, the definition of
 429 neighborhood implies it. For TVE-mix we reason as follows: Let $\mathcal{K} = \{X \simeq_{\mathcal{R}, \lambda}^? f(t_1, \dots, t_n)\} \wedge$
 430 \mathcal{K}_1 and σ be a solution of the right hand side of this rule. Then $X\sigma = F(t'_1, \dots, t'_n)\sigma \simeq_{\mathcal{R}, \lambda}$
 431 $f(t_1, \dots, t_n)\sigma$ and σ solves $X \simeq_{\mathcal{R}, \lambda}^? f(t_1, \dots, t_n)$. As for any other equation $eq \in \mathcal{K}_1$, we have
 432 $eq\vartheta$ in the right hand side, where $\vartheta = \{X \mapsto F(t'_1, \dots, t'_n)\}$. Moreover, σ is a solution of $eq\vartheta$
 433 iff $\vartheta\sigma$ is a solution of eq . The equality $X\sigma = F(t'_1, \dots, t'_n)\sigma$ implies $\vartheta\sigma = \sigma$. Hence, σ is a
 434 solution of eq . ◀

435 Our constraint solving algorithm Solve is designed as a strategy of applying Unif, Sim,
 436 and Mix. To solve a conjunction of primitive equality and similarity constraints $\mathcal{K} =$
 437 $\mathcal{K}_{\leq} \wedge \mathcal{K}_{\mathcal{R}_1, \lambda_1} \wedge \dots \wedge \mathcal{K}_{\mathcal{R}_m, \lambda_m}$, it performs the following steps:

```

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

```

439 We write $\text{Solve}(\mathcal{K}) = \mathcal{C}$, if the algorithm returns \mathcal{C} for the input \mathcal{K} . Respectively,
 440 $\text{Solve}(\mathcal{K}) = \text{false}$ if false is returned.

441 ► **Example 19.** Let \mathcal{R}_1 and \mathcal{R}_2 be the relations defined in Example 8 and illustrate the
 442 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
 443 distinguish between function variables and terms made of a function variable only, i.e.,
 444 between F and $F()$. For the same reason, we write constant-terms a_1 and a_2 in their full
 445 form $a_1()$ and $a_2()$. Primitive constraints selected to perform a particular step are underlined.

446 $X \simeq_{\mathcal{R}_1, 0.5}^? f(a_1(), a_2()) \wedge X \simeq_{\mathcal{R}_2, 0.6}^? f(Y, Y) \rightsquigarrow_{\text{TVE-mix}}$
 447 $X \doteq F(G_1(), G_2()) \wedge F \simeq_{\mathcal{R}_1, 0.5}^? f \wedge G_1() \simeq_{\mathcal{R}_1, 0.5}^? a_1() \wedge G_2() \simeq_{\mathcal{R}_1, 0.5}^? a_2() \wedge$
 448 $F(G_1(), G_2()) \simeq_{\mathcal{R}_2, 0.6}^? f(Y, Y) \rightsquigarrow_{\text{Dec-sim} \times 2}$
 449 $X \doteq F(G_1(), G_2()) \wedge F \simeq_{\mathcal{R}_1, 0.5}^? f \wedge G_1 \simeq_{\mathcal{R}_1, 0.5}^? a_1 \wedge G_2 \simeq_{\mathcal{R}_1, 0.5}^? a_2 \wedge$
 450 $F(G_1(), G_2()) \simeq_{\mathcal{R}_2, 0.6}^? f(Y, Y) \rightsquigarrow_{\text{Dec-sim, Ori-sim}}$
 451 $X \doteq F(G_1(), G_2()) \wedge F \simeq_{\mathcal{R}_1, 0.5}^? f \wedge G_1 \simeq_{\mathcal{R}_1, 0.5}^? a_1 \wedge G_2 \simeq_{\mathcal{R}_1, 0.5}^? a_2 \wedge$

35:12 Constraint solving over multiple similarity relations

$$\begin{aligned}
452 \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}^? Y \rightsquigarrow_{\text{Elim-sim}} \\
453 \quad & 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 \\
454 \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_{\text{Dec-sim}} \\
455 \quad & 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 \\
456 \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_{\text{FVE-mix}} \\
457 \quad & 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 \\
458 \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_{\text{Del-sim}} \\
459 \quad & 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 \\
460 \quad & Y \simeq_{\mathcal{R}_2,0.6}^? G_1() \wedge G_2 \simeq_{\mathcal{R}_2,0.6}^? G_1 \rightsquigarrow_{\text{FVE-mix} \times 2} \text{ (showing only successful branches)} \\
461 \quad & (X \doteq f(b_1(), b_2()) \wedge F \doteq f \wedge G_1 \doteq b_1 \wedge G_2 \doteq b_2 \wedge \\
462 \quad & Y \simeq_{\mathcal{R}_2,0.6}^? b_1() \wedge b_2 \simeq_{\mathcal{R}_2,0.6}^? b_1) \vee \\
463 \quad & (X \doteq f(c_1(), c_2()) \wedge F \doteq f \wedge G_1 \doteq c_1 \wedge G_2 \doteq c_2 \wedge \\
464 \quad & Y \simeq_{\mathcal{R}_2,0.6}^? c_1() \wedge c_2 \simeq_{\mathcal{R}_2,0.6}^? c_1) \rightsquigarrow_{\text{Del-sim} \times 2} \\
465 \quad & (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()) \vee \\
466 \quad & (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()).
\end{aligned}$$

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

$$470 \quad (X \doteq f(b_1, b_2) \wedge Y \simeq_{\mathcal{R}_2,0.6} b_1) \vee (X \doteq f(c_1, c_2) \wedge Y \simeq_{\mathcal{R}_2,0.6} c_1).$$

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

$$\begin{aligned}
474 \quad & X \simeq_{\mathcal{R}_1,0.6}^? f(Y, Y) \wedge X \simeq_{\mathcal{R}_2,0.5}^? f(Z, Z) \rightsquigarrow_{\text{TVE-mix}} \\
475 \quad & X \doteq F(X_1, X_2) \wedge F \simeq_{\mathcal{R}_1,0.6}^? f \wedge X_1 \simeq_{\mathcal{R}_1,0.6}^? Y \wedge X_2 \simeq_{\mathcal{R}_1,0.6}^? Y \wedge \\
476 \quad & F(X_1, X_2) \simeq_{\mathcal{R}_2,0.5}^? g(Z, Z) \rightsquigarrow_{\text{FVE-mix}} \\
477 \quad & X \doteq f(X_1, X_2) \wedge F \simeq_{\mathcal{R}_1,0.6}^? f \wedge X_1 \simeq_{\mathcal{R}_1,0.6}^? Y \wedge X_2 \simeq_{\mathcal{R}_1,0.6}^? Y \wedge \\
478 \quad & f(X_1, X_2) \simeq_{\mathcal{R}_2,0.5}^? g(Z, Z) \rightsquigarrow_{\text{Dec}} \\
479 \quad & X \doteq f(X_1, X_2) \wedge F \simeq_{\mathcal{R}_1,0.6}^? f \wedge X_1 \simeq_{\mathcal{R}_1,0.6}^? Y \wedge X_2 \simeq_{\mathcal{R}_1,0.6}^? Y \wedge \\
480 \quad & X_1 \simeq_{\mathcal{R}_2,0.5}^? Z \wedge X_2 \simeq_{\mathcal{R}_2,0.5}^? Z.
\end{aligned}$$

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

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

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

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

493 ■ Let $\emptyset \neq D \subseteq \text{Vert}$ be the set of all vertices, for which the marks in the graphs differ (i.e.,
 494 $D := \{v \in \text{Vert} \mid \text{mark}(v, G_1) \neq \text{mark}(v, G_2)\}$: same vertex, different markings), and
 495 $\emptyset \neq M \subseteq D$ be the set of those elements of D , which are not reachable from any of the
 496 elements in D in G_1 . (Such a subset of D exists, because the graphs are acyclic.) Then
 497 $\text{mark}(v, G_1) \subset \text{mark}(v, G_2)$ for all $v \in M$.

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

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

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

503 Let $D(G_i, G_j) := \{v \in \text{Vert} \mid \text{mark}(v, G_i) \neq \text{mark}(v, G_j)\}$ and $M(G_i, G_j)$ be the set
 504 of all those elements in $D(G_i, G_j)$ that are not reachable in G_j from $D(G_i, G_j)$, $1 \leq i <$
 505 $j \leq 3$. Assume first that $D(G_1, G_3) \neq \emptyset$ and take $v \in M(G_1, G_3)$. We want to show that
 506 $\text{mark}(v, G_1) \supset \text{mark}(v, G_3)$. The possible cases are

- 507 i. $\text{mark}(v, G_1) \neq \text{mark}(v, G_2)$ and $\text{mark}(v, G_2) = \text{mark}(v, G_3)$.
- 508 ii. $\text{mark}(v, G_1) = \text{mark}(v, G_2)$ and $\text{mark}(v, G_2) \neq \text{mark}(v, G_3)$.
- 509 iii. $\text{mark}(v, G_1) \neq \text{mark}(v, G_2)$ and $\text{mark}(v, G_2) \neq \text{mark}(v, G_3)$.

510 In case **i**, $v \in D(G_1, G_2)$. If $v \in M(G_1, G_2)$, then $\text{mark}(v, G_1) \supset \text{mark}(v, G_2) =$
 511 $\text{mark}(v, G_3)$. Now we show that the case $v \notin M(G_1, G_2)$ is impossible. Assume by contra-
 512 diction that $v \notin M(G_1, G_2)$. It means that there exists $v' \in M(G_1, G_2)$ such that $v' \rightarrow^+ v$
 513 in G_2 . But then, since $E_2 \subseteq E_3$, we have $v' \rightarrow^+ v$ in G_3 . Since $v \in M(G_1, G_3)$, we
 514 should have $v' \notin D(G_1, G_3)$, i.e., $\text{mark}(v', G_1) = \text{mark}(v', G_3)$. On the other hand, from
 515 $G_1 >_{\text{dag}} G_2$ and $v' \in M(G_1, G_2)$, we have $\text{mark}(v', G_1) \supset \text{mark}(v', G_2)$, which implies
 516 $\text{mark}(v', G_2) \subset \text{mark}(v', G_3)$, $v' \in D(G_2, G_3)$ and $v' \notin M(G_2, G_3)$. Then there should exist
 517 $v'' \in M(G_2, G_3)$ such that $v'' \rightarrow^+ v'$ in G_3 . Hence, we get $v'' \rightarrow^+ v' \rightarrow^+ v$ in G_3 . Therefore,
 518 we can not have $v'' \in D(G_1, G_3)$, because there would be a contradiction: $v \in M(G_1, G_3)$
 519 and v is reachable in G_3 from $v'' \in D(G_1, G_3)$. Hence, we get $\text{mark}(v'', G_1) = \text{mark}(v'', G_3)$,
 520 which, together with $v'' \in M(G_2, G_3)$ implies that $\text{mark}(v'', G_1) \subset \text{mark}(v'', G_2)$. Hence,
 521 $v'' \in D(G_1, G_2)$ and since $G_1 >_{\text{dag}} G_2$, there should exist $v''' \in M(G_1, G_2)$ such that
 522 $\text{mark}(v''', G_1) \supset \text{mark}(v''', G_2)$ and $v''' \rightarrow^+ v''$ in G_2 . Then we have also $v''' \rightarrow^+ v''$ in G_3 ,
 523 since $E_2 \subseteq E_3$. Moreover, $\text{mark}(v''', G_2) = \text{mark}(v''', G_3)$, because otherwise we would have
 524 a contradiction with $v'' \in M(G_2, G_3)$ (there would be $v''' \in D(G_2, G_3)$ with $v''' \rightarrow^+ v''$ in
 525 G_3). Hence, $\text{mark}(v''', G_1) \supset \text{mark}(v''', G_2) = \text{mark}(v''', G_3)$ and we get $v''' \in D(G_1, G_3)$.
 526 But it contradicts our assumption that $v \in M(G_1, G_3)$, because we got $v''' \in D(G_1, G_3)$
 527 with $v''' \rightarrow^+ v$ in G_3 . The obtained contradiction shows that the case $v \notin M(G_1, G_2)$ is
 528 impossible. (See also the diagram in Appendix A.)

529 The case **ii** can be proved analogously. In case **iii**, if we have $\text{mark}(v, G_1) \supset \text{mark}(v, G_2)$
 530 and $\text{mark}(v, G_2) \supset \text{mark}(v, G_3)$, we immediately get $\text{mark}(v, G_1) \supset \text{mark}(v, G_3)$. The other
 531 cases are not possible. For instance, assuming $\text{mark}(v, G_1) \supset \text{mark}(v, G_2)$ and $\text{mark}(v, G_2) \subset$
 532 $\text{mark}(v, G_3)$ will lead to contradiction by the same reasoning as in the proof of case **i**, where
 533 we reached $\text{mark}(v', G_1) \supset \text{mark}(v', G_2)$ and $\text{mark}(v', G_2) \subset \text{mark}(v', G_3)$.

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

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

35:14 Constraint solving over multiple similarity relations

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

540 **Proof.** Let \mathcal{K}_0 be a given conjunction of primitive constraints. We build a term variable
 541 dependency graph $G = (Vert, E)$ from \mathcal{K}_0 and maintain it during the process of solving.
 542 The vertices of G correspond to term variables in \mathcal{K}_0 so that to each variable a single
 543 vertex is assigned. For instance, if \mathcal{K}_0 contains three term variables $X_1, X_2,$ and $Y_3,$ then
 544 $Vert = \{v_{X_1}, v_{X_2}, v_{X_3}\}$. The initial marking is defined as $mark(v_X, G) = \{X\}$ for each
 545 $v_X \in Vert$. Next, to define E , we do the following: If \mathcal{K}_0 contains a solved primitive
 546 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
 547 $Y \in var(t)$ and $mark(v_X, G)$ is updated as $mark(v_X, G) \setminus \{X\}$. This is how the initial version
 548 of the graph is created. It is denoted by $G_{\mathcal{K}_0}$.

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

- 550 a) *The applied rule is of the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$.* Then from the graph $G_{\mathcal{K}}$ we obtain the graph
 551 $G_{\mathcal{K}'}$ depending on the rule:
- 552 ■ Elim-eq with $X \doteq^? t$ adds edges and removes X from the marking set of the vertex v_X
 553 exactly as described above.
 - 554 ■ Elim-sim with $X \simeq_{\mathcal{R}, \lambda}^? t$ either keeps the graph unchanged (when X occurs more than
 555 once in the resulting constraint after the application of Elim-sim), or modifies it as
 556 described above (when X occurs exactly once in the resulting constraint after the
 557 application of Elim-sim).
 - 558 ■ TVE-mix with $X \simeq_{\mathcal{R}, \lambda}^? t$ does the same modification as Elim-eq and, in addition, modifies
 559 marking: Let $Y_1, \dots, Y_m, m \geq 0,$ be all the copies of a term variable $Y \in var(t)$ created
 560 by the renaming function ρ in the TVE-mix step. Then each Y_i is associated with the
 561 vertex v_Y (i.e., $v_{Y_i} = v_Y$) and $mark(v_Y, G)$ gets updated as $mark(v_Y, G) \cup \{Y_1, \dots, Y_m\}$.
 - 562 ■ No other rule of the form $\mathcal{K} \rightsquigarrow \mathcal{K}'$ modifies the graph.
- 563 b) *The applied rule is of the form $\mathcal{K} \rightsquigarrow \mathcal{K}'_1 \vee \dots \vee \mathcal{K}'_n, n > 1.$* Then $G_{\mathcal{K}} = G_{\mathcal{K}'_1} = \dots = G_{\mathcal{K}'_n}$.

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

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

572 Let us consider the pair $(G_{\mathcal{K}}, \mathbf{V}_F(\mathcal{K}))$ of such a term variable dependency graph $G_{\mathcal{K}}$
 573 associated to a constraint \mathcal{K} and a set of function variables $\mathbf{V}_F(\mathcal{K})$ occurring in \mathcal{K} . These pairs
 574 are ordered lexicographically by $>_{\text{dag}}$ and $>$. By Theorem 22, $>_{\text{dag}}$ is well-founded. The
 575 relation $>$ on natural numbers is well-founded. Therefore, their lexicographic combination is
 576 well-founded. Since Unif and Sim are terminating (Theorems 10 and 12), each iteration of
 577 the **while** loop in the definition of Solve either stops with false, or reaches the application
 578 of Mix. In this process, measure of the pair $(G_{\mathcal{K}}, \mathbf{V}_F(\mathcal{K}))$ does not increase, because the
 579 Unif and Sim rules, as we have already seen, either decrease or keep unchanged $G_{\mathcal{K}}$, and
 580 do not add new function variables. The application of Mix either fails, or strictly reduces
 581 $(G_{\mathcal{K}}, \mathbf{V}_F(\mathcal{K}))$: TVE-mix strictly decreases $G_{\mathcal{K}}$, and FVE-mix does not change $G_{\mathcal{K}}$ but strictly
 582 decreases $\mathbf{V}_F(\mathcal{K})$. Hence, the **while** loop in Solve can be executed only finitely many times.

583 If Solve does not stop with false, the only possible non-solved primitive constraints are
 584 those between variables, whose left hand side has occurrences in at least two different kind of

585 constraints. For any other case, there is an applicable rule. Hence, the obtained constraint is
586 in appr-solved form. ◀

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

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

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

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

600 Application of Unif to \mathcal{K} leads to a new constraint \mathcal{C}_{un} , which contains a solved from
601 $\mathcal{K}_{\text{un-sol}}$ such that $\sigma_{\mathcal{K}_{\text{un-sol}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. Application of Sim to \mathcal{C}_{un} gives \mathcal{C}_{sim} , which contains
602 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
603 TVE-mix is applicable, we have an equation $X \simeq_{\mathcal{R},\lambda}^? f(t_1, \dots, t_n)$. TVE-mix extends the
604 solved form by a new equation $X \doteq^? \rho(f(t_1, \dots, t_n))$, obtaining $\mathcal{K}_{\text{tve-sol}}$. By definition of ρ ,
605 the term $\rho(f(t_1, \dots, t_n))$ contains fresh variables for each symbol in $f(t_1, \dots, t_n)$ and, hence,
606 $\sigma_{\mathcal{K}_{\text{tve-sol}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. It is important at this step to record which fresh variable is a copy of
607 which original variable, maintaining an function *original-of*(V') = V , where $V \in \text{var}(\mathcal{K})$ and
608 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.).
609 If the rule FVE-mix is applicable, we have an equation $F \simeq_{\mathcal{R},\lambda}^? f$. We make a step by this rule,
610 adding a new equation $F \doteq^? \text{original-of}(F)\vartheta$ and obtaining a new solved form $\mathcal{K}_{\text{fve-sol}}$. Let φ
611 be the substitution $\{\text{original-of}(F) \mapsto \text{original-of}(F)\vartheta\}$. Then we have $\sigma_{\mathcal{K}_{\text{fve-sol}}}\varphi \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$.
612 Iterating this process, we do not get false, since \mathcal{K} was solvable. By Theorem 23, the process
613 terminates with an appr-solved form $\mathcal{K}_{\text{sol}} \wedge \mathcal{K}_{\text{var}}$ such that $\sigma_{\mathcal{K}_{\text{sol}}}\varphi_1 \cdots \varphi_k \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$, where
614 the φ 's are substitutions of the form $\{\text{original-of}(V) \mapsto \text{original-of}(V)\vartheta\}$. Let $\sigma_{\mathcal{K}_{\text{var}}}$ be the
615 restriction of ϑ to the variables of \mathcal{K}_{var} . Then $\sigma_{\mathcal{K}_{\text{var}}}$ is a solution of \mathcal{K}_{var} . And we have
616 $\sigma_{\mathcal{K}_{\text{sol}}}\varphi_1 \cdots \varphi_k \sigma_{\mathcal{K}_{\text{var}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$.

617 For the φ 's, composition is commutative, because they are ground substitutions with
618 disjoint domains. For some of them we have $\sigma_{\mathcal{K}_{\text{sol}}}\varphi_i = \sigma_{\mathcal{K}_{\text{sol}}}$, because at some step we
619 might have solved an equation with *original-of*(V) variable in its left for *original-of*(V) \in
620 $\text{dom}(\varphi_i)$. We assume that the φ 's in the composition are rearranged so that $\sigma_{\mathcal{K}_{\text{sol}}}\varphi_1 \cdots \varphi_i =$
621 $\sigma_{\mathcal{K}_{\text{sol}}}\varphi_{i+1} \cdots \varphi_k$. These remaining φ 's are those for which the algorithm reached a variables-
622 only equation containing *original-of*(V), which occurs in the domain of one of the φ 's. But
623 then $\varphi_{i+1} \cdots \varphi_k$ is a part of $\sigma_{\mathcal{K}_{\text{var}}}$. Hence, we can get rid of them, obtaining $\sigma_{\mathcal{K}_{\text{sol}}}\sigma_{\mathcal{K}_{\text{var}}}$.
624 Hence, we get $\sigma_{\mathcal{K}_{\text{sol}}}\sigma_{\mathcal{K}_{\text{var}}} \preceq_{\mathcal{W}(\mathcal{K})} \vartheta$. ◀

625 4 Computing approximation degrees

626 In the algorithm, we have not included the computation of approximation degrees, but it
627 can be done easily. Instead of constraints in DNF of the form $\mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$, we will be
628 working with expressions (we call them extended constraints) $(K_1, \mathcal{D}_1) \vee \cdots \vee (K_n, \mathcal{D}_n)$,

35:16 Constraint solving over multiple similarity relations

629 where $\mathfrak{D}_1, \dots, \mathfrak{D}_n$ are approximation degrees. The rules will carry the degree (“computed
630 so far”) as an additional parameter, but only two rules would change them: **Del-sim** and
631 **FVE-mix**. Their variants with degree modification would work on constraint-degree pairs (\uplus
632 stands for disjoint union):

633 **Del-sim-deg** : $(\tau_1 \simeq_{\mathcal{R}, \lambda}^? \tau_2 \wedge \mathcal{K}, \{\langle \mathcal{R}, \mathfrak{d} \rangle\} \uplus \mathfrak{D}) \rightsquigarrow (\mathcal{K}, \{\langle \mathcal{R}, \min\{\mathfrak{d}, \mathcal{R}(\tau_1, \tau_2) \rangle\} \rangle\} \cup \mathfrak{D})$
634 where $\tau_1, \tau_2 \in \mathbf{C}_F \cup \mathbf{V}_F \cup \mathbf{V}_T$ and $\mathcal{R}(\tau_1, \tau_2) \geq \lambda$.

635 **FVE-mix-deg** : $(F \simeq_{\mathcal{R}, \lambda}^? f \wedge \mathcal{K}, \{\langle \mathcal{R}, \mathfrak{d} \rangle\} \uplus \mathfrak{D}) \rightsquigarrow$
636 $\bigvee_{g \in \mathbf{N}(f, \mathcal{R}, \lambda)} (F \doteq g \wedge \mathcal{K}\{F \mapsto g\}, \{\langle \mathcal{R}, \min\{\mathfrak{d}, \mathcal{R}(f, g) \rangle\} \rangle\} \cup \mathfrak{D}),$
637 where $F \in \text{var}(\mathcal{K})$.
638

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

```

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

```

5 Discussion and summary

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

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

654 $P(X, Y) \leftarrow P \simeq_{\mathcal{R}_1, \lambda_1} p, X \simeq_{\mathcal{R}_1, \lambda_1} F(Y), Y \simeq_{\mathcal{R}_2, \lambda_2} c, r(X), r(F(Y)).$
655 $r(F(X)) \leftarrow F \simeq_{\mathcal{R}_2, \lambda_2} h, X \simeq_{\mathcal{R}_1, \lambda_1} a.$
656

657 Assume $\mathcal{R}_1(p, q) = 0.9$, $\mathcal{R}_1(a, b) = 0.8$, $\mathcal{R}_1(f, g) = 0.6$, $\mathcal{R}_2(g, h) = 0.5$, $\mathcal{R}_2(b, c) = 0.7$,
658 $\lambda_1 = \lambda_2 = 0.4$. Then by performing the usual CLP inference (i.e., syntactically unifying
659 the selected query and the head of the corresponding clause) for the query $\leftarrow q(X, Y)$, the
660 resulting constraint (together with the approximation degrees) will be $(X \doteq g(b) \wedge Y \doteq b;$
661 $\{\langle \mathcal{R}_1, 0.8 \rangle, \langle \mathcal{R}_2, 0.5 \rangle\}) \vee (X \doteq h(b) \wedge Y \doteq b; \{\langle \mathcal{R}_1, 0.8 \rangle, \langle \mathcal{R}_2, 0.7 \rangle\})$.

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

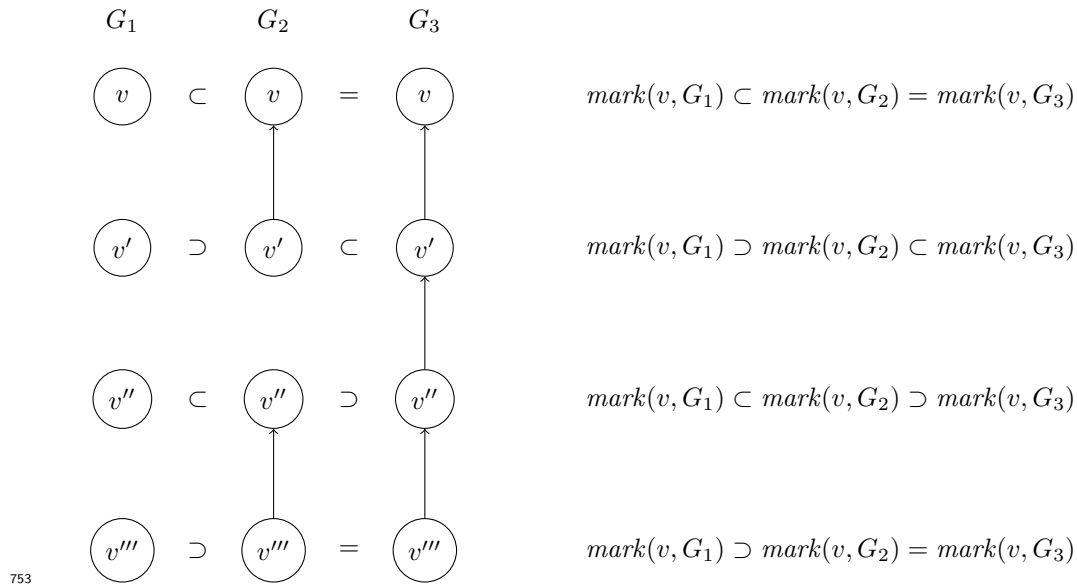
667 — References

- 668 1 Hassan Ait-Kaci and Gabriella Pasi. Fuzzy unification and generalization of first-order terms
669 over similar signatures. In Fabio Fioravanti and John P. Gallagher, editors, *Logic-Based*
670 *Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017,*
671 *Namur, Belgium, October 10-12, 2017, Revised Selected Papers*, volume 10855 of *Lecture*
672 *Notes in Computer Science*, pages 218–234. Springer, 2017. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-319-94460-9_13)
673 [978-3-319-94460-9_13](https://doi.org/10.1007/978-3-319-94460-9_13), doi:10.1007/978-3-319-94460-9_13.
- 674 2 Hassan Ait-Kaci and Gabriella Pasi. Lattice operations on terms with fuzzy signatures. *CoRR*,
675 abs/1709.00964, 2017. URL: <http://arxiv.org/abs/1709.00964>, arXiv:1709.00964.
- 676 3 Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press,
677 1998.
- 678 4 Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei
679 Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier
680 and MIT Press, 2001.
- 681 5 Francesca Arcelli Fontana and Ferrante Formato. Likelog: A logic programming language for
682 flexible data retrieval. In Barrett R. Bryant, Gary B. Lamont, Hisham Haddad, and Janice H.
683 Carroll, editors, *Proceedings of the 1999 ACM Symposium on Applied Computing, SAC'99,*
684 *San Antonio, Texas, USA, February 28 - March 2, 1999*, pages 260–267. ACM, 1999. URL:
685 <https://doi.org/10.1145/298151.298348>, doi:10.1145/298151.298348.
- 686 6 Francesca Arcelli Fontana and Ferrante Formato. A similarity-based resolution rule. *Int.*
687 *J. Intell. Syst.*, 17(9):853–872, 2002. URL: <https://doi.org/10.1002/int.10067>, doi:10.
688 1002/int.10067.
- 689 7 Ferrante Formato, Giangiacomo Gerla, and Maria I. Sessa. Extension of logic programming by
690 similarity. In Maria Chiara Meo and Manuel Vilares Ferro, editors, *1999 Joint Conference on*
691 *Declarative Programming, AGP'99, L'Aquila, Italy, September 6-9, 1999*, pages 397–410, 1999.
- 692 8 Ferrante Formato, Giangiacomo Gerla, and Maria I. Sessa. Similarity-based unification.
693 *Fundam. Inform.*, 41(4):393–414, 2000. URL: <https://doi.org/10.3233/FI-2000-41402>,
694 doi:10.3233/FI-2000-41402.
- 695 9 Pascual Julián Iranzo and Clemente Rubio-Manzano. A sound and complete semantics for a
696 similarity-based logic programming language. *Fuzzy Sets and Systems*, 317:1–26, 2017. URL:
697 <https://doi.org/10.1016/j.fss.2016.12.016>, doi:10.1016/j.fss.2016.12.016.
- 698 10 Pascual Julián Iranzo and Fernando Sáenz-Pérez. An efficient proximity-based unification
699 algorithm. In *2018 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2018, Rio*
700 *de Janeiro, Brazil, July 8-13, 2018*, pages 1–8. IEEE, 2018. URL: [https://doi.org/10.1109/](https://doi.org/10.1109/FUZZ-IEEE.2018.8491593)
701 [FUZZ-IEEE.2018.8491593](https://doi.org/10.1109/FUZZ-IEEE.2018.8491593), doi:10.1109/FUZZ-IEEE.2018.8491593.
- 702 11 Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *J. Log.*
703 *Program.*, 19/20:503–581, 1994. URL: [https://doi.org/10.1016/0743-1066\(94\)90033-7](https://doi.org/10.1016/0743-1066(94)90033-7),
704 doi:10.1016/0743-1066(94)90033-7.
- 705 12 Pascual Julián-Iranzo and Clemente Rubio-Manzano. Proximity-based unification theory. *Fuzzy*
706 *Sets and Systems*, 262:21–43, 2015. URL: <https://doi.org/10.1016/j.fss.2014.07.006>,
707 doi:10.1016/j.fss.2014.07.006.
- 708 13 Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In Pascal Fontaine,
709 Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th*
710 *International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*,
711 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.
- 712 14 Stanislav Krajci, Rastislav Lencses, Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtás. A
713 similarity-based unification model for flexible querying. In Troels Andreasen, Amihai Motro,
714 Henning Christiansen, and Henrik Legind Larsen, editors, *Flexible Query Answering Systems,*
715 *5th International Conference, FQAS 2002, Copenhagen, Denmark, October 27-29, 2002,*
716 *Proceedings*, volume 2522 of *Lecture Notes in Computer Science*, pages 263–273. Springer, 2002.
717 URL: https://doi.org/10.1007/3-540-36109-X_21, doi:10.1007/3-540-36109-X_21.
718

- 719 **15** Temur Kutsia and Cleo Pau. Solving proximity constraints. In Maurizio Gabbrielli, editor,
720 *Logic-Based Program Synthesis and Transformation - 29th International Symposium, LOPSTR*
721 *2019, Porto, Portugal, October 8-10, 2019, Revised Selected Papers*, volume 12042 of *Lecture*
722 *Notes in Computer Science*, pages 107–122. Springer, 2019. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-030-45260-5_7)
723 [978-3-030-45260-5_7](https://doi.org/10.1007/978-3-030-45260-5_7), doi:10.1007/978-3-030-45260-5_7.
- 724 **16** Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtás. Similarity-based unification: a multi-
725 adjacent approach. *Fuzzy Sets and Systems*, 146(1):43–62, 2004. URL: [https://doi.org/10.](https://doi.org/10.1016/j.fss.2003.11.005)
726 [1016/j.fss.2003.11.005](https://doi.org/10.1016/j.fss.2003.11.005), doi:10.1016/j.fss.2003.11.005.
- 727 **17** Maria I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theor. Comput.*
728 *Sci.*, 275(1-2):389–426, 2002. URL: [https://doi.org/10.1016/S0304-3975\(01\)00188-8](https://doi.org/10.1016/S0304-3975(01)00188-8), doi:
729 [10.1016/S0304-3975\(01\)00188-8](https://doi.org/10.1016/S0304-3975(01)00188-8).
- 730 **18** Mariya I. Vasileva, Bryan A. Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar,
731 and David A. Forsyth. Learning type-aware embeddings for fashion compatibility. In
732 Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Com-*
733 *puter Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September*
734 *8-14, 2018, Proceedings, Part XVI*, volume 11220 of *Lecture Notes in Computer Science*,
735 pages 405–421. Springer, 2018. URL: https://doi.org/10.1007/978-3-030-01270-0_24,
736 doi:10.1007/978-3-030-01270-0_24.
- 737 **19** Andreas Veit, Serge J. Belongie, and Theofanis Karaletsos. Conditional similarity networks. In
738 *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu,*
739 *HI, USA, July 21-26, 2017*, pages 1781–1789. IEEE Computer Society, 2017. URL: [https:](https://doi.org/10.1109/CVPR.2017.193)
740 [//doi.org/10.1109/CVPR.2017.193](https://doi.org/10.1109/CVPR.2017.193), doi:10.1109/CVPR.2017.193.
- 741 **20** Harry Virtanen. Vague domains, s-unification, logic programming. *Electr. Notes Theor.*
742 *Comput. Sci.*, 66(5):86–103, 2002. URL: [https://doi.org/10.1016/S1571-0661\(04\)80516-4](https://doi.org/10.1016/S1571-0661(04)80516-4),
743 doi:10.1016/S1571-0661(04)80516-4.
- 744 **21** Peter Vojtás. Declarative and procedural semantics of fuzzy similarity based unification.
745 *Kybernetika*, 36(6):707–720, 2000. URL: <http://www.kybernetika.cz/content/2000/6/707>.

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

747 In the proof of Theorem 3, case **i**, we assumed $v \notin M(G_1, G_2)$ and reached a contradiction.
 748 The reasoning line there can be illustrated by the diagram below, where we step by step
 749 construct the vertices v', v'', v''' until we reach a contradiction with another assumption
 750 $v \in M(G_1, G_3)$. The symbols $\subset, \supset, =$ between the vertices show the relation between the
 751 marked sets of those vertices. The leftmost graph is a part of G_1 , the middle one of G_2 , and
 752 the rightmost one of G_3 .



753