SYMBOLIC CONSTRAINTS AND QUANTITATIVE EXTENSIONS OF EQUALITY

Temur Kutsia
RISC, Johannes Kepler University Linz

February 9, 2023. University of Brasilia

JOHANNES KEPLER
UNIVERSITY LINZ

Symbolic constraints

Usually: conjunctions of primitive (atomic) constraints in some logic language.

Examples of primitive constraints:

- equations,
- disequations,
- atomic formulas expressing e.g., ordering, membership, generalization, or dominance relations,
- etc.

Solutions: variable substitutions that satisfy the given formula.

Symbolic constraints

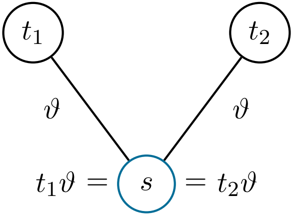
Our focus: equational and generalization constraints.

Solving methods: unification, matching, anti-unification.

Appear in many areas of computational logic:

- automated reasoning
- term rewriting
- declarative programming
- pattern-based calculi
- unification theory
- …

Dual problems: unification / anti-unification



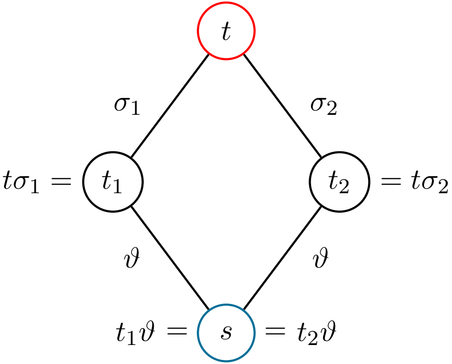
s: most general instance

ϑ solves the unification problem $t_1 =^? t_2$

Dual problems: unification / anti-unification

t: least general generalization

$X = t$ solves the anti-unification problem $X : t_1 \triangleq t_2$



s: most general instance

ϑ solves the unification problem $t_1 =^? t_2$

Dual problems: unification / anti-unification

$$
\begin{tikzcd}[column sep=small]
f(x, g(x), g(y)) \arrow[dr, dash, "\{x \mapsto a, z \mapsto g(y)\}"'] & & f(a, g(a), z) \arrow[dl, dash, "\{x \mapsto a, z \mapsto g(y)\}"] \\
& f(a, g(a), g(y)) &
\end{tikzcd}
$$

most general instance

Dual problems: unification / anti-unification

least general generalization

$$\begin{tikzcd}
& f(u, g(u), v) \arrow[dl, dash, "\{u \mapsto x, v \mapsto g(y)\}"'] \arrow[dr, dash, "\{u \mapsto a, v \mapsto z\}"] & \\
f(x, g(x), g(y)) \arrow[dr, dash, "\{x \mapsto a, z \mapsto g(y)\}"'] & & f(a, g(a), z) \arrow[dl, dash, "\{x \mapsto a, z \mapsto g(y)\}"] \\
& f(a, g(a), g(y)) &
\end{tikzcd}$$

most general instance

Precise vs imprecise

In these examples, the given information was precise.

Two symbols, terms, etc. are either equal or not.

How to deal with cases when the information is not perfect?

Outline

From equalities to tolerances

Overview

Quantitative relations over terms

Similarity-based unification

Proximity-based unification using blocks, basic signatures

Proximity constraints using classes

- Unification and matching in basic signatures
- Generalization in basic signatures
- Unification and matching in fully fuzzy signatures
- Generalization in fully fuzzy signatures

Future research directions

Outline

From equalities to tolerances

Reasoning with incomplete, imperfect information is very common in human communication.

Its modeling is a highly nontrivial task.

There are various notions associated to such information (e.g., uncertainty, imprecision, vagueness, fuzziness).

From equalities to tolerances

Reasoning with incomplete, imperfect information is very common in human communication.

Its modeling is a highly nontrivial task.

There are various notions associated to such information (e.g., uncertainty, imprecision, vagueness, fuzziness).

Different methodologies have been proposed to deal with them (e.g., approaches based on default logic, probability, fuzzy sets, etc.)

From equalities to tolerances

For many problems in this area, exact equality is replaced by its approximation.

Tolerance relations are a tool to express the approximation, modeling the corresponding imprecise information.

They are reflexive and symmetric but not necessarily transitive relations, expressing the idea of closeness or resemblance.

From equalities to tolerances

Examples of tolerance relations include some well-known mathematical notions, e.g.,

- a and b are vertices of the same edge in an undirected graph,

From equalities to tolerances

Examples of tolerance relations include some well-known mathematical notions, e.g.,

- a and b are vertices of the same edge in an undirected graph,
- a and b are points in a metric space that are within a given positive distance from each other,

From equalities to tolerances

Examples of tolerance relations include some well-known mathematical notions, e.g.,

- a and b are vertices of the same edge in an undirected graph,
- a and b are points in a metric space that are within a given positive distance from each other,
- Two binary sequences a and b differ from each other in at most e positions for some given error level e.

From equalities to tolerances

Examples of tolerance relations include some well-known mathematical notions, e.g.,

- a and b are vertices of the same edge in an undirected graph,
- a and b are points in a metric space that are within a given positive distance from each other,
- Two binary sequences a and b differ from each other in at most e positions for some given error level e.
- For a topological space T and its fixed covering ω, the relation "a and b are points in T that belong to the same element of ω".

From equalities to tolerances

The term “tolerance relation” has been coined by Zeeman (1962).

His research on tolerance spaces was motivated by their applications in describing the brain and visual perspective.

From equalities to tolerances

The original ideas date back to Poincaré in 1890s.

In physical world, he argued, accumulation of measurement errors lead to the violation of transitivity of equality (in contrast to the ideal mathematical world).

In his view, tolerance has the fundamental importance in distinguishing mathematics applied to the physical world from ideal mathematics.

From equalities to tolerances

Tolerance space theory has been studied from different points of view (e.g., topology or category theory).

Related notions: rough sets, near sets, approximation spaces, …

Some modern applications include, e.g., information systems, granular computing, image analysis.

From equalities to tolerances

Tolerance space theory has been studied from different points of view (e.g., topology or category theory).

Related notions: rough sets, near sets, approximation spaces, ...

Some modern applications include, e.g., information systems, granular computing, image analysis.

Relatively recent references include, e.g.,

- J. F. Peters and P. Wasilewski. Tolerance spaces: origins, theoretical aspects and applications. *Inf. Sci.*, 195:211–225, 2012.
- A. B. Sossinsky. Tolerance spaces revisited I: almost solutions. *Mathematical Notes,* 106:439–445, 2019.

From equalities to tolerances

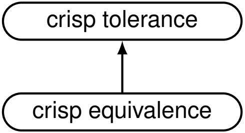
In the original version, tolerance relations were crisp
(two objects are either close to each other or not).

Later, their graded counterparts appeared which led, among others, to tolerance relations in the fuzzy setting.

From equalities to tolerances

In the original version, tolerance relations were crisp (two objects are either close to each other or not).

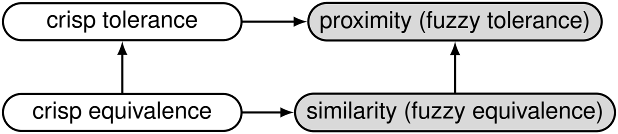
Later, their graded counterparts appeared which led, among others, to tolerance relations in the fuzzy setting.



From equalities to tolerances

In the original version, tolerance relations were crisp (two objects are either close to each other or not).

Later, their graded counterparts appeared which led, among others, to tolerance relations in the fuzzy setting.



Fuzzy tolerances and equivalences

A fuzzy relation on a set S: a mapping from S to $[0, 1]$.

Fuzzy tolerances and equivalences

A fuzzy relation on a set S: a mapping from S to $[0, 1]$.

A fuzzy relation $\mathcal{R}$ on S is a proximity (fuzzy tolerance) relation on S iff 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 $s_1, s_2 \in S$.

Fuzzy tolerances and equivalences

A fuzzy relation on a set S: a mapping from S to $[0, 1]$.

A fuzzy relation $\mathcal{R}$ on S is a proximity (fuzzy tolerance) relation on S iff 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 $s_1, s_2 \in S$.

$\mathcal{R}(s_1, s_2)$: the degree of proximity between s_1 and s_2.

Fuzzy tolerances and equivalences

A fuzzy relation on a set S: a mapping from S to $[0, 1]$.

A fuzzy relation $\mathcal{R}$ on S is a proximity (fuzzy tolerance) relation on S iff 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 $s_1, s_2 \in S$.

$\mathcal{R}(s_1, s_2)$: the degree of proximity between s_1 and s_2.

Not to confuse the proximity degree between two objects with the proximity between them in terms of distance!

Fuzzy tolerances and equivalences

A fuzzy relation on a set S: a mapping from S to $[0, 1]$.

A fuzzy relation $\mathcal{R}$ on S is a proximity (fuzzy tolerance) relation on S iff 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 $s_1, s_2 \in S$.

$\mathcal{R}(s_1, s_2)$: the degree of proximity between s_1 and s_2.

Not to confuse the proximity degree between two objects with the proximity between them in terms of distance!

A proximity relation on S is a strict if $\mathcal{R}(s_1, s_2) = 1$ implies $s_1 = s_2$ for all $s_1, s_2 \in S$.

Fuzzy tolerances and equivalences

A proximity relation on S is a similarity (fuzzy equivalence) relation on S if it is transitive:

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

where $\wedge$ is a T-norm: an associative, commutative, non-decreasing (monotonic) binary operation on $[0, 1]$ with 1 as the unit element.

Fuzzy tolerances and equivalences

T-norm (triangular norm) generalizes intersection in a lattice and conjunction in logic.

Some well-known T-norms:

- Minimum T-norm (aka Gödel T-norm): $s \wedge t = \min(s, t)$.
- Product T-norm: $s \wedge t = s * t$.
- Łukasiewicz T-norm: $s \wedge t = \max\{0, s + t - 1\}$.

In the rest, we use the $\min$ T-norm.

Fuzzy tolerances and equivalences

Given $0 \leq \lambda \leq 1$, the λ-cut of $\mathcal{R}$ on S is the crisp relation

$$\mathcal{R}_\lambda := \{(s_1, s_2) \mid \mathcal{R}(s_1, s_2) \geq \lambda\}.$$

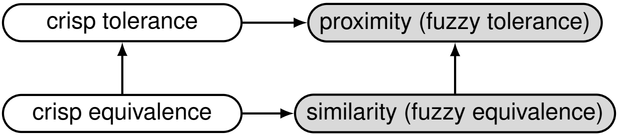
Notation: $s_1 \simeq_{\mathcal{R},\lambda} s_2$ means $(s_1, s_2) \in \mathcal{R}_\lambda$.

The cut value λ provides a threshold: defines which objects are treated proximal to each other ($(\mathcal{R}, \lambda)$-proximal) and which are not.

Fuzzy tolerances and equivalences

λ-cut of a proximity relation is a crisp tolerance relation.

λ-cut of a similarity relation is a crisp equivalence relation.



Outline

From equalities to tolerances

Overview

Quantitative relations over terms

Similarity-based unification

Proximity-based unification using blocks, basic signatures

Proximity constraints using classes

- Unification and matching in basic signatures
- Generalization in basic signatures
- Unification and matching in fully fuzzy signatures
- Generalization in fully fuzzy signatures

Future research directions

Terms and substitutions

$\mathcal{V}$: a set of variables.

$\mathcal{F}$: a set of function symbols of fixed arity.

$\mathcal{F} \cap \mathcal{V} = \emptyset$.

Terms and substitutions

$\mathcal{V}$: a set of variables.

$\mathcal{F}$: a set of function symbols of fixed arity.

$\mathcal{F} \cap \mathcal{V} = \emptyset$.

Terms over $\mathcal{F}$ and $\mathcal{V}$:

$$t := x \mid f(t_1, \ldots, t_n), \text{ where } f \text{ is } n\text{-ary.}$$

Terms and substitutions

$\mathcal{V}$: a set of variables.

$\mathcal{F}$: a set of function symbols of fixed arity.

$\mathcal{F} \cap \mathcal{V} = \emptyset$.

Terms over $\mathcal{F}$ and $\mathcal{V}$:

$$t := x \mid f(t_1, \ldots, t_n), \text{ where } f \text{ is } n\text{-ary}.$$

Substitutions: mappings from variables to terms, where all but finitely many variables are mapped to themselves.

Quantitative relations over terms

We need to define the notions of proximity and similarity for terms.

Idea: start from a corresponding relation on the given alphabet and extend it to terms.

Basic and fully fuzzy signatures

Two kinds of signature, depending how fuzzy relations are defined on the set of function symbols:

- More special: basic fuzzy signatures.
  Proximal/similar function symbols can have different names, but not different arities.
- More general: fully fuzzy signatures.
  Proximal/similar function symbols can have different names and different arities.

Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

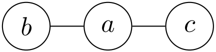
block-based	vs	class-based
block of a:		class of a:
a clique to which a belongs		the neighborhood of a

Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

block-based	vs	class-based
block of a:		class of a:
a clique to which a belongs		the neighborhood of a

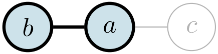


Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

block-based	vs	class-based
block of a:		class of a:
a clique to which a belongs		the neighborhood of a

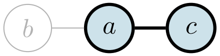


Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

block-based	vs	class-based
block of a:		class of a:
a clique to which a belongs		the neighborhood of a

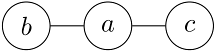


Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

block-based	vs	class-based
block of a:		class of a:
a clique to which a belongs		the neighborhood of a

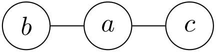


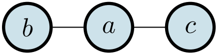
Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

block-based	vs	class-based
block of a: a clique to which a belongs		class of a: the neighborhood of a

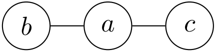
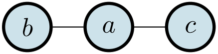




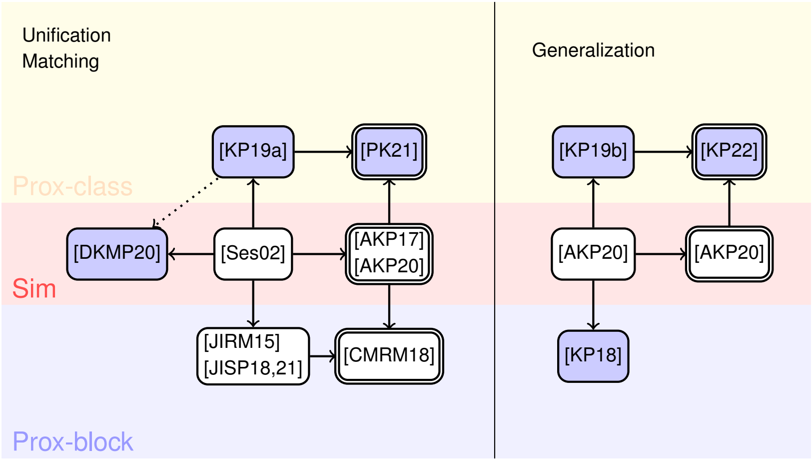
Block- and class-based approaches

Looking at proximity relations as undirected graphs, one can talk about cliques and neighborhoods in them.

One distinguishes between block- and class-based approaches towards solving symbolic constraints for proximity relations.

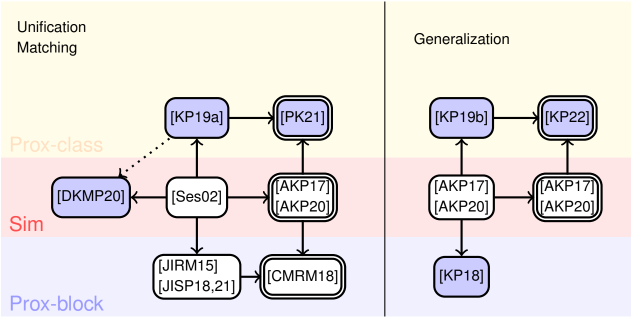
block-based	vs	class-based
block of a:		class of a:
a clique to which a belongs		the neighborhood of a
		
$\{x \simeq^?_{\mathcal{R},\lambda} b,\ x \simeq^?_{\mathcal{R},\lambda} c\}$		$\{x \simeq^?_{\mathcal{R},\lambda} b,\ x \simeq^?_{\mathcal{R},\lambda} c\}$
not solvable		solved by $\{x \mapsto a\}$

Overview



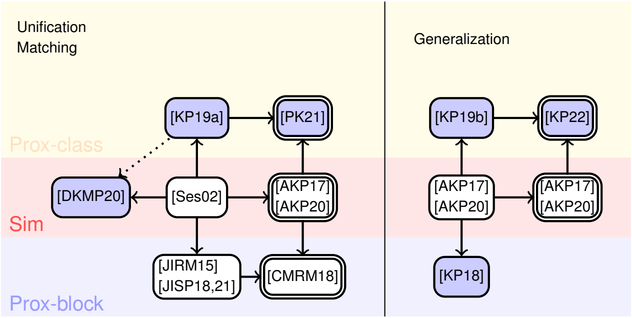
Entries with double borders consider fully fuzzy signatures.

Overview



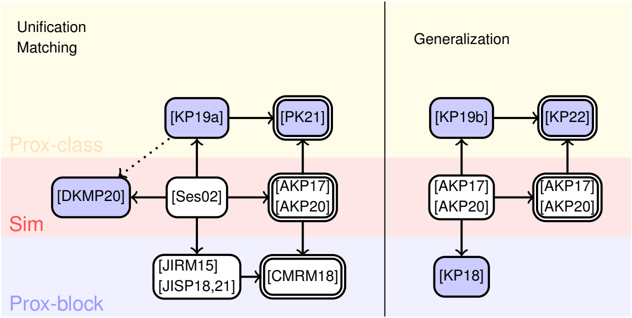
- Similarity-based unification: investigated quite intensively in the context of approximate reasoning, fuzzy logic programming, query languages; works by Ying, Fontana, Fermato, Gerla, Sessa [Ses02], Medina, Ojeda-Aciego, Vojtas and others. Aït-Kaci and Pasi [AKP17,20] extended Sessa's work to fully fuzzy signatures, preparing a ground to similarity-based unification under background theories.

Overview



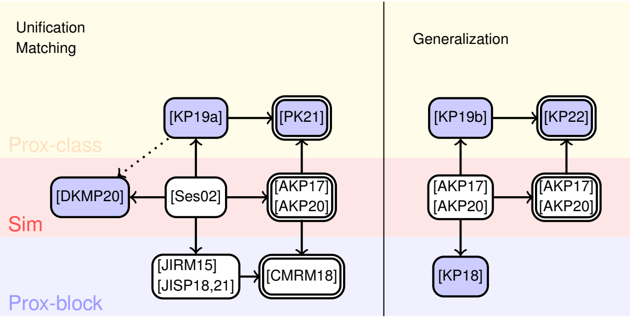
- Unification with multiple similarity relations: arises in the context of e.g., understanding visual similarities in learning image embeddings; addressed in [DKMP20]; generalizes Sessa's work from single to multiple similarity relations; transitivity is lost; is related to class-based proximity unification.

Overview



- Similarity-based generalization: Aït-Kaci and Pasi [AKP17,20] investigated the problem for fully fuzzy signatures; the results apply to basic fuzzy signatures as well.

Overview

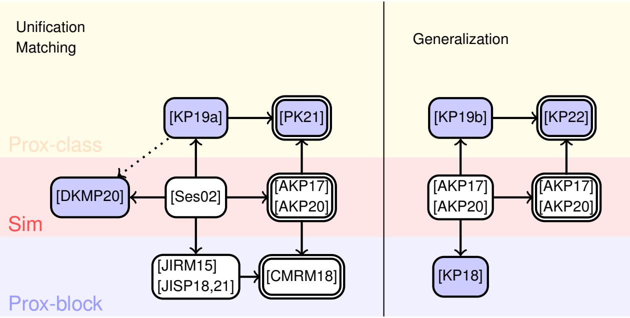


- Proximity-based unification: Proximity relations help to represent fuzzy information in situations, where similarity is not adequate.

  Proximity-based unification helps to manage imprecise information in the context of approximate reasoning and (fuzzy) logic programming.

  Block-based approach in basic signatures: Julián-Iranzo, Sáenz-Pérez, Rubio-Manzano [JIRM15], [JISP18,21], etc. Class-based approach in basic signatures to unification/matching, Kutsia and Pau [KP19a] and Pau's PhD thesis [Pau22].

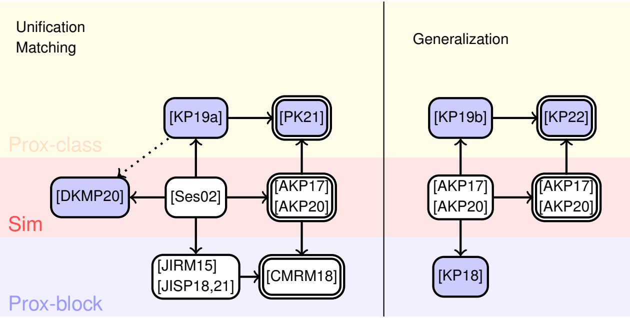
Overview



- Proximity-based unification: block-based approach in fully fuzzy signatures, restricted case, used in fuzzy logic programming; work by Cornejo et al. [CMRM18]

  Class-based approach in fully fuzzy signatures by Pau and Kutsia [PK21], generalizing class-based proximity unification/matching and fully fuzzy similarity unification/matching. Details in Pau's PhD thesis [Pau22].

Overview



- Proximity-based generalization: block-based approach in basic fuzzy signatures; requires an algorithm for enumerating all maximal clique-partitions in an undirected graph; Kutsia and Pau [KP18].

  Class-based approach in fully fuzzy signatures is presented by a generic framework by Kutsia and Pau [KP22]; an algorithm for basic fuzzy signatures is a special case. See also Pau's PhD thesis [Pau22].

Overview

Technique	Signature	Relation	Approach
Unification	Basic fuzzy	Similarity	
Matching	Basic fuzzy	Similarity	
Generalization	Basic fuzzy	Similarity	
Unification	Basic fuzzy	Proximity	Block-based
Matching	Basic fuzzy	Proximity	Block-based
Generalization	Basic fuzzy	Proximity	Block-based
Unification	Basic fuzzy	Proximity	Class-based
Matching	Basic fuzzy	Proximity	Class-based
Generalization	Basic fuzzy	Proximity	Class-based

Overview

Technique	Signature	Relation	Approach
Unification	Fully fuzzy	Similarity	
Matching	Fully fuzzy	Similarity	
Generalization	Fully fuzzy	Similarity	
Unification	Fully fuzzy	Proximity	Block-based
Matching	Fully fuzzy	Proximity	Block-based
Generalization	Fully fuzzy	Proximity	Block-based
Unification	Fully fuzzy	Proximity	Class-based
Matching	Fully fuzzy	Proximity	Class-based
Generalization	Fully fuzzy	Proximity	Class-based

Outline

Quantitative term relations: basic signatures

$\mathcal{R}$: a given proximity relation on a basic fuzzy signature $\mathcal{F}$.

In basic signatures, $\mathcal{R}(f, g) = 0$ if $arity(f) \neq arity(g)$.

Extending $\mathcal{R}$ to terms:

- $\mathcal{R}(x, x) = 1$ for all $x \in \mathcal{V}$.
- $\mathcal{R}(f(t_1, \dots, t_n), g(s_1, \dots, s_n)) = \mathcal{R}(f, g) \wedge \bigwedge_{i=1}^{n} \mathcal{R}(t_i, s_i)$.
- $\mathcal{R}(t, s) = 0$ in all other cases.

Quantitative term relations: basic signatures

$\mathcal{R}$: a given proximity relation on a basic fuzzy signature $\mathcal{F}$.

In basic signatures, $\mathcal{R}(f, g) = 0$ if $arity(f) \neq arity(g)$.

Extending $\mathcal{R}$ to terms:

- $\mathcal{R}(x, x) = 1$ for all $x \in \mathcal{V}$.
- $\mathcal{R}(f(t_1, \ldots, t_n), g(s_1, \ldots, s_n)) = \mathcal{R}(f, g) \wedge \bigwedge_{i=1}^{n} \mathcal{R}(t_i, s_i)$.
- $\mathcal{R}(t, s) = 0$ in all other cases.

Then $\mathcal{R}$ is a proximity relation on terms.

If $\mathcal{R}$ is a similarity relations on $\mathcal{F}$, then its extension to terms is also similarity.

Quantitative term relations: fully fuzzy case

$\mathcal{R}$: a given proximity relation on a fully fuzzy signature $\mathcal{F}$.

To be able to extend proximity from alphabet symbols to terms, we need to know which arguments of proximal symbols are related to each other (argument relations).

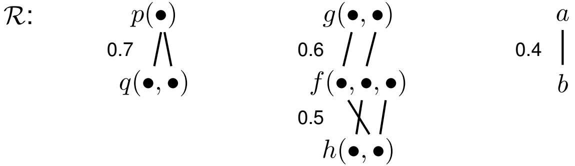
We assume that this information is provided.

If $\mathcal{R}(f, g) = \alpha$ and the argument relation between f and g is ρ, we write $f \sim^{\rho}_{\mathcal{R},\alpha} g$.

Argument relations should satisfy certain extra properties in order a similarity relation on the signature to be extendable to a similarity relation over terms.

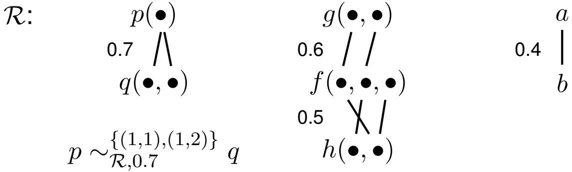
Quantitative term relations: fully fuzzy case

Example of a proximity relation on a fully fuzzy signature.



Quantitative term relations: fully fuzzy case

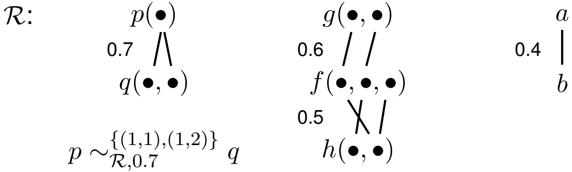
Example of a proximity relation on a fully fuzzy signature.



$$p \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},0.7} q$$

Quantitative term relations: fully fuzzy case

Example of a proximity relation on a fully fuzzy signature.



We have $f \sim_{\mathcal{R},1}^{Id} f$ for all f.

Quantitative term relations: fully fuzzy case

Extending $\mathcal{R}$ from the signature to terms:

- $\mathcal{R}(x, x) = 1$ for all variables x.
- $\mathcal{R}(f(t_1, \dots, t_n), g(s_1, \dots, s_m)) = \alpha \wedge \bigwedge_{(i,j)\in\rho} \mathcal{R}(t_i, s_j)$, where $f \sim^{\rho}_{\mathcal{R},\alpha} g$.
- $\mathcal{R}(t, s) = 0$ in all other cases.

Such an extension is a proximity relation on terms.

The extension of $\mathcal{R}$ on terms is similarity if $\mathcal{R}$ is similarity on the signature and the argument relations satisfy certain properties (Aït-Kaci and Pasi, 2020)

Proximity for basic signatures is a special case for proximity for fully fuzzy signatures, with ρ required to be a (left and right) total identity relation.

Relations $\preceq$ and $\precsim_{\mathcal{R},\lambda}$

$\preceq$ for terms:

t is syntactically more general than s, written $t \preceq s$,
if there exists a σ such that $t\sigma = s$.

$\preceq$ for substitutions:

ϑ is syntactically more general than φ, written $\vartheta \preceq \varphi$,
if there exists a σ such that $x\vartheta\sigma = x\varphi$ for all x.

Relations $\preceq$ and $\precsim_{\mathcal{R},\lambda}$

$\preceq$ for terms:

t is syntactically more general than s, written $t \preceq s$, if there exists a σ such that $t\sigma = s$.

$\preceq$ for substitutions:

ϑ is syntactically more general than φ, written $\vartheta \preceq \varphi$, if there exists a σ such that $x\vartheta\sigma = x\varphi$ for all x.

$\precsim_{\mathcal{R},\lambda}$ for terms:

t is $(\mathcal{R}, \lambda)$-more general than s, written $t \precsim_{\mathcal{R},\lambda} s$, if there exists σ such that $t\sigma \simeq_{\mathcal{R},\lambda} s$.

$\precsim_{\mathcal{R},\lambda}$ for substitutions:

ϑ is $(\mathcal{R}, \lambda)$-more general than φ, written $\vartheta \precsim_{\mathcal{R},\lambda} \varphi$, if there exists σ such that $x\vartheta\sigma \simeq_{\mathcal{R},\lambda} x\varphi$ for all x.

Properties of $\preceq$ and $\precsim_{\mathcal{R},\lambda}$

$\preceq$ is a transitive relation.

$\precsim_{\mathcal{R},\lambda}$ is not transitive, in general (but it is, if $\mathcal{R}$ is similarity).

If $a \simeq_{\mathcal{R},\lambda} b$, $b \simeq_{\mathcal{R},\lambda} c$, and $a \not\simeq_{\mathcal{R},\lambda} c$,
then $a \precsim_{\mathcal{R},\lambda} b$, $b \precsim_{\mathcal{R},\lambda} c$, and $a \not\precsim_{\mathcal{R},\lambda} c$.

Properties of $\preceq$ and $\precsim_{\mathcal{R},\lambda}$

$\preceq$ is a transitive relation.

$\precsim_{\mathcal{R},\lambda}$ is not transitive, in general (but it is, if $\mathcal{R}$ is similarity).

If $a \simeq_{\mathcal{R},\lambda} b$, $b \simeq_{\mathcal{R},\lambda} c$, and $a \not\simeq_{\mathcal{R},\lambda} c$,
then $a \precsim_{\mathcal{R},\lambda} b$, $b \precsim_{\mathcal{R},\lambda} c$, and $a \not\precsim_{\mathcal{R},\lambda} c$.

$\preceq \; \subseteq \; \precsim_{\mathcal{R},\lambda}$ for any $\mathcal{R}$ and λ.

Proximity-/similarity-based unification

Given: A proximity relation $\mathcal{R}$, a cut value λ, and term pairs (t_i, s_i), $1 \leq i \leq n$.

Find: σ such that $t_i\sigma \simeq_{\mathcal{R},\lambda} s_i\sigma$ for all $1 \leq i \leq n$.

Proximity-/similarity-based unification

Given: A proximity relation $\mathcal{R}$, a cut value λ, and term pairs (t_i, s_i), $1 \leq i \leq n$.

Find: σ such that $t_i\sigma \simeq_{\mathcal{R},\lambda} s_i\sigma$ for all $1 \leq i \leq n$.

$(\mathcal{R}, \lambda)$-unification problem: $P = \{t_1 \simeq^?_{\mathcal{R},\lambda} s_1, \ldots, t_n \simeq^?_{\mathcal{R},\lambda} s_n\}$.

We may skip $(\mathcal{R}, \lambda)$, when it does not cause confusion.

σ: $(\mathcal{R}, \lambda)$-unifier of P.

Interesting unifiers are most general ones.

Proximity-/similarity-based unification

Given: A proximity relation $\mathcal{R}$, a cut value λ, and term pairs (t_i, s_i), $1 \le i \le n$.

Find: σ such that $t_i\sigma \simeq_{\mathcal{R},\lambda} s_i\sigma$ for all $1 \le i \le n$.

$(\mathcal{R}, \lambda)$-unification problem: $P = \{t_1 \simeq^?_{\mathcal{R},\lambda} s_1, \dots, t_n \simeq^?_{\mathcal{R},\lambda} s_n\}$.

We may skip $(\mathcal{R}, \lambda)$, when it does not cause confusion.

σ: $(\mathcal{R}, \lambda)$-unifier of P.

Interesting unifiers are most general ones.

The signature can be basic or fully fuzzy.

Similarity-based unification: when $\mathcal{R}$ is similarity.

Proximity-/similarity-based matching

Given: A proximity relation $\mathcal{R}$, a cut value λ, and term pairs (t_i, s_i), $1 \leq i \leq n$.

Find: σ such that $t_i\sigma \simeq_{\mathcal{R},\lambda} s_i$ for all $1 \leq i \leq n$.

Proximity-/similarity-based matching

Given: A proximity relation $\mathcal{R}$, a cut value λ, and term pairs (t_i, s_i), $1 \le i \le n$.

Find: σ such that $t_i\sigma \simeq_{\mathcal{R},\lambda} s_i$ for all $1 \le i \le n$.

$(\mathcal{R}, \lambda)$-matching problem: $P = \{t_1 \precsim^?_{\mathcal{R},\lambda} s_1, \dots, t_n \precsim^?_{\mathcal{R},\lambda} s_n\}$.

We may skip $(\mathcal{R}, \lambda)$, when it does not cause confusion.

σ: $(\mathcal{R}, \lambda)$-matcher of P.

Can be treated as a special case of unification.

Better: use a simpler dedicated algorithm.

Proximity-/similarity-based matching

Given: A proximity relation $\mathcal{R}$, a cut value λ, and term pairs (t_i, s_i), $1 \leq i \leq n$.

Find: σ such that $t_i\sigma \simeq_{\mathcal{R},\lambda} s_i$ for all $1 \leq i \leq n$.

$(\mathcal{R}, \lambda)$-matching problem: $P = \{t_1 \precsim^?_{\mathcal{R},\lambda} s_1, \dots, t_n \precsim^?_{\mathcal{R},\lambda} s_n\}$.

We may skip $(\mathcal{R}, \lambda)$, when it does not cause confusion.

σ: $(\mathcal{R}, \lambda)$-matcher of P.

Can be treated as a special case of unification.

Better: use a simpler dedicated algorithm.

The signature can be basic or fully fuzzy.

Similarity-based matching: when $\mathcal{R}$ is similarity.

Proximity-/similarity-based generalization

Given: A proximity relation $\mathcal{R}$, a cut value λ, and two terms t and s.

Find: A term r such that $r \precsim_{\mathcal{R},\lambda} t$ and $r \precsim_{\mathcal{R},\lambda} s$.

Proximity-/similarity-based generalization

Given: A proximity relation $\mathcal{R}$, a cut value λ, and two terms t and s.

Find: A term r such that $r \precsim_{\mathcal{R},\lambda} t$ and $r \precsim_{\mathcal{R},\lambda} s$.

$t \triangleq_{\mathcal{R},\lambda} s$: the notation for t and s to be generalized.

We may skip $(\mathcal{R}, \lambda)$, when it does not cause confusion.

r: $(\mathcal{R}, \lambda)$-generalization of s and t.

Interesting generalizations are the least general ones.

Proximity-/similarity-based generalization

Given: A proximity relation $\mathcal{R}$, a cut value λ, and two terms t and s.

Find: A term r such that $r \precsim_{\mathcal{R},\lambda} t$ and $r \precsim_{\mathcal{R},\lambda} s$.

$t \triangleq_{\mathcal{R},\lambda} s$: the notation for t and s to be generalized.

We may skip $(\mathcal{R}, \lambda)$, when it does not cause confusion.

r: $(\mathcal{R}, \lambda)$-generalization of s and t.

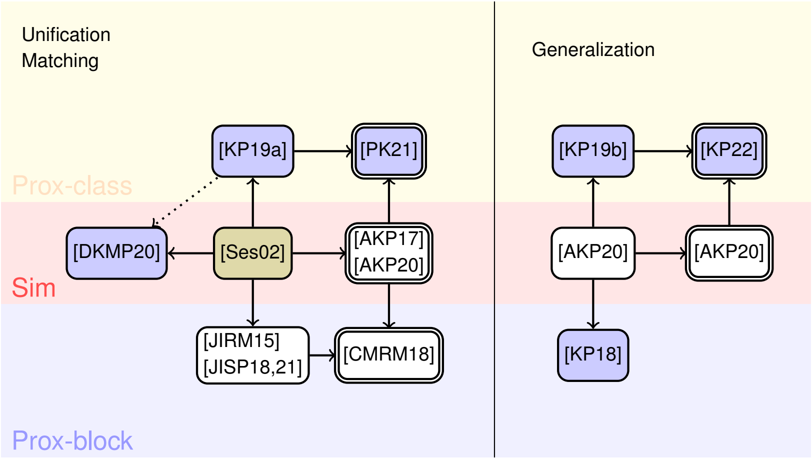
Interesting generalizations are the least general ones.

The signature can be basic or fully fuzzy.

Similarity-based generalization: when $\mathcal{R}$ is similarity.

Outline

Overview



Entries with double borders consider fully fuzzy signatures.

Unification: similarity-based, basic signature

The “weak unification” algorithm by Sessa.

Computes a mgu together with its unification degree.

Mgus have the highest unification degree.

Unification: similarity-based, basic signature

The "weak unification" algorithm by Sessa.

Computes a mgu together with its unification degree.

Mgus have the highest unification degree.

If $\mathcal{R}(f, g) = 0.7$ and $\mathcal{R}(a, b) = 0.5$, then $f(x) \simeq^?_{\mathcal{R},\lambda} g(a)$ has two solutions:

- $\{x \mapsto a\}$ with degree 0.7,
- $\{x \mapsto b\}$ with degree 0.5.

Remember: our T-norm is $\min$.

Unification: similarity-based, basic signature

The "weak unification" algorithm by Sessa.

Computes a mgu together with its unification degree.

Mgus have the highest unification degree.

If $\mathcal{R}(f, g) = 0.7$ and $\mathcal{R}(a, b) = 0.5$, then $f(x) \simeq^?_{\mathcal{R},\lambda} g(a)$ has two solutions:

- $\{x \mapsto a\}$ with degree 0.7,
- $\{x \mapsto b\}$ with degree 0.5.

Remember: our T-norm is $\min$.

For strict similarity relations, unifiers with degree 1 coincide with syntactic unifiers.

Unification: similarity-based, basic signature

The "weak unification" algorithm by Sessa.

These rules are an adaptation of those for the syntactic unification:

DECOMPOSITION:

$$\{f(s_1,\dots,s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1,\dots,t_n)\} \uplus P;\ \alpha;\ \sigma \Longrightarrow$$
$$\{s_1 \simeq^?_{\mathcal{R},\lambda} t_1,\dots,s_n \simeq^?_{\mathcal{R},\lambda} t_n\} \cup P;\ \alpha \wedge \mathcal{R}(f,g);\ \sigma,$$

if $\mathcal{R}(f,g) \geq \lambda$.

CLASH:

$$\{f(s_1,\dots,s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1,\dots,t_m)\} \uplus P;\ \alpha;\ \sigma \Longrightarrow \bot,$$

if $\mathcal{R}(f,g) < \lambda$.

The other rules are the same as in syntactic unification.

Unification: similarity-based, basic signature

The "weak unification" algorithm by Sessa.

These rules are an adaptation of those for the syntactic unification:

DECOMPOSITION:

$$\{f(s_1,\dots,s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1,\dots,t_n)\} \uplus P;\ \alpha;\ \sigma \Longrightarrow$$
$$\{s_1 \simeq^?_{\mathcal{R},\lambda} t_1,\dots,s_n \simeq^?_{\mathcal{R},\lambda} t_n\} \cup P;\ \alpha \wedge \mathcal{R}(f,g);\ \sigma,$$

if $\mathcal{R}(f,g) \geq \lambda$.

CLASH:

$$\{f(s_1,\dots,s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1,\dots,t_m)\} \uplus P;\ \alpha;\ \sigma \Longrightarrow \perp,$$

if $\mathcal{R}(f,g) < \lambda$.

The other rules are the same as in syntactic unification.

The algorithm computes an mgu with the maximal unification degree.

Sessa's algorithm in action

Similarity relation $\mathcal{R}$:

$$\begin{tikzcd}[arrows=dash] & a \arrow[dl, "0.3"'] \arrow[dr, "0.3"] & \\ b \arrow[rr, "0.4"] & & c \end{tikzcd} \qquad f \xrightarrow{0.6} g$$

Sessa's algorithm in action

Similarity relation $\mathcal{R}$:

$$\begin{tikzcd}[arrows=-] & a \arrow[dl, "0.3"'] \arrow[dr, "0.3"] & \\ b \arrow[rr, "0.4"] & & c \end{tikzcd} \qquad \begin{tikzcd}[arrows=-] f \arrow[r, "0.6"] & g \end{tikzcd}$$

Take $\lambda = 0.2$ and unify $f(x, c)$ and $g(b, x)$.

Sessa's algorithm in action

Similarity relation $\mathcal{R}$:

$$\begin{tikzcd} & a \arrow[dl, dash, "0.3"'] \arrow[dr, dash, "0.3"] & \\ b \arrow[rr, dash, "0.4"] & & c \end{tikzcd} \qquad f \overset{0.6}{\text{———}} g$$

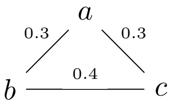
Take $\lambda = 0.2$ and unify $f(x, c)$ and $g(b, x)$.

$$\{f(x,c) \simeq^{?}_{\mathcal{R},0.2} g(b,x)\};\ 1;\ Id \Longrightarrow$$
$$\{x \simeq^{?}_{\mathcal{R},0.2} b,\ \ c \simeq^{?}_{\mathcal{R},0.2} x\};\ 0.6;\ Id \Longrightarrow$$
$$\{c \simeq^{?}_{\mathcal{R},0.2} b\};\ 0.6;\ \{x \mapsto b\} \Longrightarrow$$
$$\emptyset;\ 0.4;\ \{x \mapsto b\}.$$

$\{x \mapsto b\}$ is an mgu, with degree 0.4.

Sessa's algorithm in action

Similarity relation $\mathcal{R}$:



$f \overset{0.6}{\text{———}} g$

Take $\lambda = 0.2$ and unify $f(x, c)$ and $g(b, x)$.

$$\{f(x,c) \simeq^?_{\mathcal{R},0.2} g(b,x)\};\ 1;\ Id \Longrightarrow$$
$$\{x \simeq^?_{\mathcal{R},0.2} b,\ \ c \simeq^?_{\mathcal{R},0.2} x\};\ 0.6;\ Id \Longrightarrow$$
$$\{c \simeq^?_{\mathcal{R},0.2} b\};\ 0.6;\ \{x \mapsto b\} \Longrightarrow$$
$$\emptyset;\ 0.4;\ \{x \mapsto b\}.$$

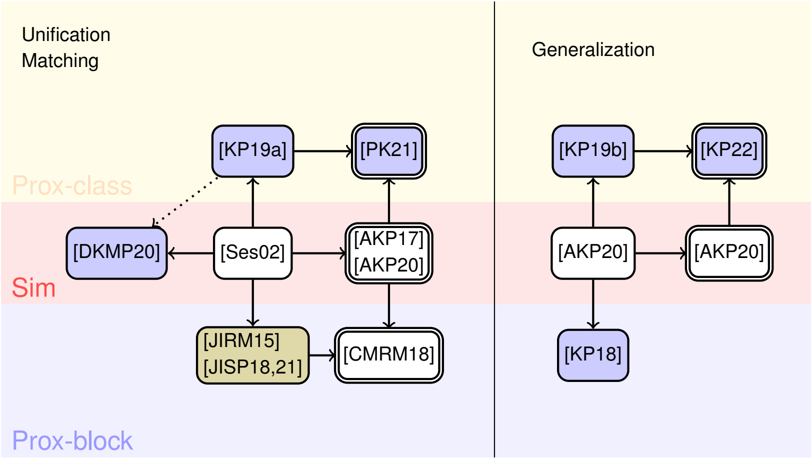
$\{x \mapsto b\}$ is an mgu, with degree 0.4.

Other mgu would be $\{x \mapsto c\}$ (with the same degree 0.4).

$\{x \mapsto a\}$ is a solution (not mgu): its degree is smaller, 0.3.

Outline

Overview

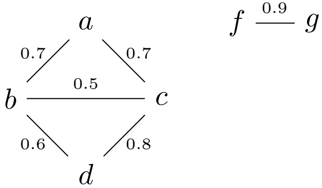


Entries with double borders consider fully fuzzy signatures.

Proximity blocks

Proximity blocks are cliques (complete subgraphs) in the graph that corresponds to the proximity relation.

$\mathcal{R}$:

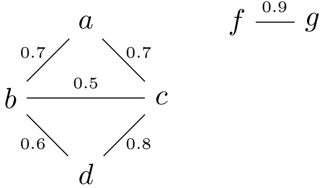


Three $(\mathcal{R}, 0.5)$-blocks: $\{a, b, c\}$, $\{b, c, d\}$ and $\{f, g\}$.

In block-based proximity unification, one symbol cannot belong at the same time to two different blocks.

Proximity blocks

$\mathcal{R}$:



Three $(\mathcal{R}, 0.5)$-blocks: $\{a, b, c\}$, $\{b, c, d\}$ and $\{f, g\}$.

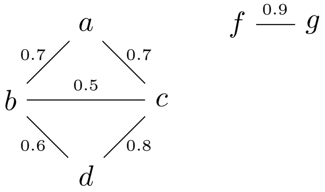
$f(x, x) \simeq^{?}_{\mathcal{R},\lambda} g(b, c)$ has four solutions:

$$\{x \mapsto b\}; 0.5, \quad \{x \mapsto c\}; 0.5, \quad \{x \mapsto a\}; 0.7, \quad \{x \mapsto d\}; 0.6.$$

The algorithm by Julian-Iranzo et al. computes $\{x \mapsto b\}; 0.5$ (or $\{x \mapsto c\}; 0.5$, depending on the choice of an equation).

Proximity blocks

$\mathcal{R}$:



Three $(\mathcal{R}, 0.5)$-blocks: $\{a, b, c\}$, $\{b, c, d\}$ and $\{f, g\}$.

$f(x, x) \simeq^?_{\mathcal{R},\lambda} g(b, c)$ has four solutions:

$$\{x \mapsto b\}; 0.5, \quad \{x \mapsto c\}; 0.5, \quad \{x \mapsto a\}; 0.7, \quad \{x \mapsto d\}; 0.6.$$

The algorithm by Julian-Iranzo et al. computes $\{x \mapsto b\}; 0.5$ (or $\{x \mapsto c\}; 0.5$, depending on the choice of an equation).

$f(x, x) \simeq^?_{\mathcal{R},\lambda} g(a, d)$ has no solutions.

Outline

Proximity classes

$\mathcal{R}_\lambda$:

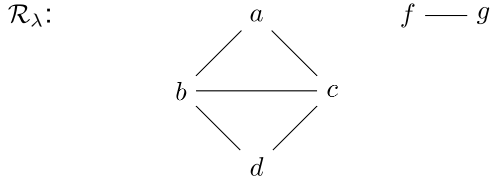
$$\begin{tikzcd}[arrows=dash] & a \arrow[dl] \arrow[dr] & \\ b \arrow[rr] \arrow[dr] & & c \arrow[dl] \\ & d & \end{tikzcd} \qquad \begin{tikzcd}[arrows=dash] f \arrow[r] & g \end{tikzcd}$$

We think that the terms $f(x, x)$ and $g(a, d)$ should be unifiable.

Reason: a and d have common neighbors, b and c.

It would be natural to have $\{x \mapsto b\}$ and $\{x \mapsto c\}$ as unifiers of $f(x, x)$ and $g(a, d)$.

Proximity classes



We think that the terms $f(x,x)$ and $g(a,d)$ should be unifiable.

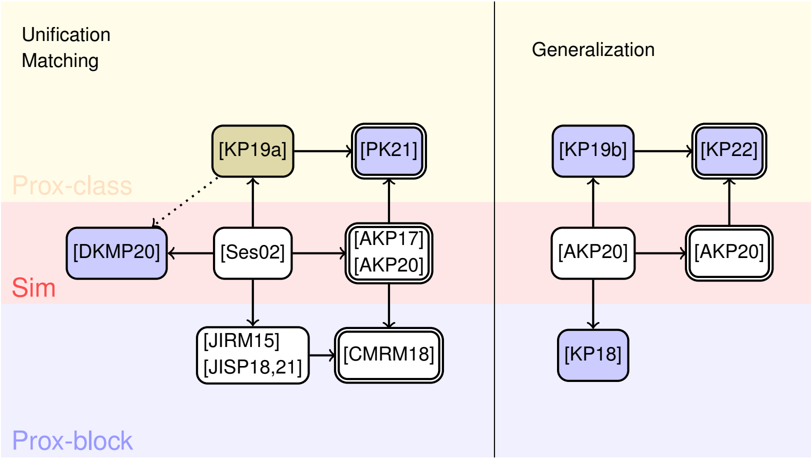
Reason: a and d have common neighbors, b and c.

It would be natural to have $\{x \mapsto b\}$ and $\{x \mapsto c\}$ as unifiers of $f(x,x)$ and $g(a,d)$.

Proximity class of a symbol: its neighborhood in the graph.

$\mathsf{class}(a, \mathcal{R}, \lambda) = \{a, b, c\}$. $\quad$ $\mathsf{class}(d, \mathcal{R}, \lambda) = \{d, b, c\}$.

Overview



Entries with double borders consider fully fuzzy signatures.

Outline

From equalities to tolerances

Overview

Quantitative relations over terms

Similarity-based unification

Proximity-based unification using blocks, basic signatures

Proximity constraints using classes

- Unification and matching in basic signatures
- Generalization in basic signatures
- Unification and matching in fully fuzzy signatures
- Generalization in fully fuzzy signatures

Future research directions

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Take $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$ and the problems

$$P_1 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(y,b)\},\ P_2 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(b,b)\}.$$

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Take $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$ and the problems

$$P_1 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(y,b)\},\ P_2 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(b,b)\}.$$

Let $\sigma = \{x \mapsto d, y \mapsto c\}$ and $\vartheta = \{x \mapsto a, y \mapsto c\}$.

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Take $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$ and the problems

$$P_1 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(y,b)\},\ P_2 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(b,b)\}.$$

Let $\sigma = \{x \mapsto d, y \mapsto c\}$ and $\vartheta = \{x \mapsto a, y \mapsto c\}$.

σ is a unifier of P_1: $f(d,c) \simeq_{\mathcal{R},\lambda} f(c,b)$.

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Take $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$ and the problems

$$P_1 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(y,b)\},\ P_2 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(b,b)\}.$$

Let $\sigma = \{x \mapsto d, y \mapsto c\}$ and $\vartheta = \{x \mapsto a, y \mapsto c\}$.

σ is a unifier of P_1: $f(d,c) \simeq_{\mathcal{R},\lambda} f(c,b)$.

But σ is not a unifier of P_2: $f(d,c) \not\simeq_{\mathcal{R},\lambda} f(b,b)$.

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Take $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$ and the problems

$$P_1 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(y,b)\},\ P_2 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(b,b)\}.$$

Let $\sigma = \{x \mapsto d, y \mapsto c\}$ and $\vartheta = \{x \mapsto a, y \mapsto c\}$.

ϑ is not a unifier of P_1: $f(a,c) \not\simeq_{\mathcal{R},\lambda} f(c,b)$.

Proximity-based unification using classes

Some peculiarities.

Syntactic unification problems

$$\{f(x,y) \doteq^? f(y,b)\} \text{ and } \{f(x,y) \doteq^? f(b,b)\}$$

have the same set of unifiers.

In proximity-based unification with classes this is not the case.

Take $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$ and the problems

$$P_1 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(y,b)\},\ P_2 = \{f(x,y) \simeq^?_{\mathcal{R},\lambda} f(b,b)\}.$$

Let $\sigma = \{x \mapsto d, y \mapsto c\}$ and $\vartheta = \{x \mapsto a, y \mapsto c\}$.

ϑ is not a unifier of P_1: $f(a,c) \not\simeq_{\mathcal{R},\lambda} f(c,b)$.

But ϑ is a unifier of P_2: $f(a,c) \simeq_{\mathcal{R},\lambda} f(b,b)$.

Proximity-based unification using classes

Some more peculiarities.

If σ is an $(\mathcal{R}, \lambda)$-unifier of a unification problem P, then any syntactic instance of σ is also an $(\mathcal{R}, \lambda)$-unifier of P.

It is not the case with $(\mathcal{R}, \lambda)$-instances, in general.

Proximity-based unification using classes

Some more peculiarities.

If σ is an $(\mathcal{R}, \lambda)$-unifier of a unification problem P, then any syntactic instance of σ is also an $(\mathcal{R}, \lambda)$-unifier of P.

It is not the case with $(\mathcal{R}, \lambda)$-instances, in general.

Let $\mathcal{R}_\lambda = \{(a, b), (b, c)\}$ and $P = \{x \simeq^?_{\mathcal{R},\lambda} f(y)\}$.

Proximity-based unification using classes

Some more peculiarities.

If σ is an $(\mathcal{R}, \lambda)$-unifier of a unification problem P, then any syntactic instance of σ is also an $(\mathcal{R}, \lambda)$-unifier of P.

It is not the case with $(\mathcal{R}, \lambda)$-instances, in general.

Let $\mathcal{R}_\lambda = \{(a, b), (b, c)\}$ and $P = \{x \simeq^?_{\mathcal{R},\lambda} f(y)\}$.

Take $\sigma = \{x \mapsto f(y)\}$ and $\varphi = \{x \mapsto f(a), y \mapsto c\}$.

Proximity-based unification using classes

Some more peculiarities.

If σ is an $(\mathcal{R}, \lambda)$-unifier of a unification problem P, then any syntactic instance of σ is also an $(\mathcal{R}, \lambda)$-unifier of P.

It is not the case with $(\mathcal{R}, \lambda)$-instances, in general.

Let $\mathcal{R}_\lambda = \{(a, b), (b, c)\}$ and $P = \{x \simeq^?_{\mathcal{R},\lambda} f(y)\}$.

Take $\sigma = \{x \mapsto f(y)\}$ and $\varphi = \{x \mapsto f(a), y \mapsto c\}$.

$\sigma \precsim_{\mathcal{R},\lambda} \varphi$ because

$$\begin{aligned} \sigma \{y \mapsto b\} = \{x \mapsto f(b), y \mapsto b\} &\simeq_{\mathcal{R},\lambda} \\ \{x \mapsto f(a), y \mapsto c\} &= \varphi. \end{aligned}$$

Proximity-based unification using classes

Some more peculiarities.

If σ is an $(\mathcal{R}, \lambda)$-unifier of a unification problem P, then any syntactic instance of σ is also an $(\mathcal{R}, \lambda)$-unifier of P.

It is not the case with $(\mathcal{R}, \lambda)$-instances, in general.

Let $\mathcal{R}_\lambda = \{(a, b), (b, c)\}$ and $P = \{x \simeq^?_{\mathcal{R},\lambda} f(y)\}$.

Take $\sigma = \{x \mapsto f(y)\}$ and $\varphi = \{x \mapsto f(a), y \mapsto c\}$.

$\sigma \precsim_{\mathcal{R},\lambda} \varphi$ because

$$\begin{aligned}\sigma\{y \mapsto b\} = \{x \mapsto f(b), y \mapsto b\} \simeq_{\mathcal{R},\lambda} \\ \{x \mapsto f(a), y \mapsto c\} = \varphi.\end{aligned}$$

σ is an $(\mathcal{R}, \lambda)$-unifier of P: $f(y) \simeq_{\mathcal{R},\lambda} f(y)$.

Proximity-based unification using classes

Some more peculiarities.

If σ is an $(\mathcal{R}, \lambda)$-unifier of a unification problem P, then any syntactic instance of σ is also an $(\mathcal{R}, \lambda)$-unifier of P.

It is not the case with $(\mathcal{R}, \lambda)$-instances, in general.

Let $\mathcal{R}_\lambda = \{(a, b), (b, c)\}$ and $P = \{x \simeq^?_{\mathcal{R},\lambda} f(y)\}$.

Take $\sigma = \{x \mapsto f(y)\}$ and $\varphi = \{x \mapsto f(a), y \mapsto c\}$.

$\sigma \precsim_{\mathcal{R},\lambda} \varphi$ because

$$\begin{aligned} \sigma\,\{y \mapsto b\} = \{x \mapsto f(b), y \mapsto b\} &\simeq_{\mathcal{R},\lambda} \\ \{x \mapsto f(a), y \mapsto c\} &= \varphi. \end{aligned}$$

σ is an $(\mathcal{R}, \lambda)$-unifier of P: $f(y) \simeq_{\mathcal{R},\lambda} f(y)$.

φ is not: $f(a) \not\simeq_{\mathcal{R},\lambda} f(c)$.

Minimal complete set of approximate unifiers

A complete set of $(\mathcal{R}, \lambda)$-unifiers of a unification problem P:
a set of substitutions Σ satisfying the properties:

Soundness:

Every $\sigma \in \Sigma$ is an $(\mathcal{R}, \lambda)$-unifier of P;

Completeness:

For any $(\mathcal{R}, \lambda)$-unifier ϑ of P, there exists $\sigma \in \Sigma$ with $\sigma \preceq \vartheta$.

Minimal complete set of approximate unifiers

A complete set of $(\mathcal{R}, \lambda)$-unifiers of a unification problem P:
a set of substitutions Σ satisfying the properties:

Soundness:

Every $\sigma \in \Sigma$ is an $(\mathcal{R}, \lambda)$-unifier of P;

Completeness:

For any $(\mathcal{R}, \lambda)$-unifier ϑ of P, there exists $\sigma \in \Sigma$ with $\sigma \preceq \vartheta$.

Σ is a minimal complete set of $(\mathcal{R}, \lambda)$-unifiers of P if in addition, the minimality condition holds:

No two elements in Σ are comparable with respect to $\preceq$:
For all $\sigma, \vartheta \in \Sigma$, if $\sigma \neq \vartheta$, then $\sigma \not\preceq \vartheta$.

Proximity-based unification using classes

Some more peculiarities.

Let $\mathcal{R}_\lambda = \{(a,b),(b,c)\}$.

$$\text{MCSU}(\{x \simeq^?_{\mathcal{R},\lambda} b\}) = \{\{x \mapsto a\},\{x \mapsto b\},\{x \mapsto c\}\}.$$

Contains $\precsim_{\mathcal{R},\lambda}$-comparable substitutions:

$$\{x \mapsto a\} \precsim_{\mathcal{R},\lambda} \{x \mapsto b\},$$
$$\{x \mapsto b\} \precsim_{\mathcal{R},\lambda} \{x \mapsto c\}.$$

By they are not $\preceq$-comparable.

Proximity-based unification using classes

Some more peculiarities.

Let $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$.

Take a variable-only unification problem:

$$P = \{x \simeq^?_{\mathcal{R},\lambda} y,\ y \simeq^?_{\mathcal{R},\lambda} z,\ z \simeq^?_{\mathcal{R},\lambda} u\}.$$

Proximity-based unification using classes

Some more peculiarities.

Let $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$.

Take a variable-only unification problem:

$$P = \{x \simeq^?_{\mathcal{R},\lambda} y,\ y \simeq^?_{\mathcal{R},\lambda} z,\ z \simeq^?_{\mathcal{R},\lambda} u\}.$$

Somewhat unexpectedly, $\{\{x \mapsto u,\ y \mapsto u,\ z \mapsto u\}\} \neq \mathsf{MCSU}(P)$.

Completeness does not hold:

- $\{x \mapsto u, y \mapsto u, z \mapsto u\} \not\precsim \{x \mapsto a, y \mapsto b, z \mapsto c, u \mapsto d\}$,
- but $\{x \mapsto a, y \mapsto b, z \mapsto c, u \mapsto d\}$ is an $(\mathcal{R},\lambda)$-unifier of P.

Proximity-based unification using classes

Some more peculiarities.

Let $\mathcal{R}_\lambda = \{(a,b),(b,c),(c,d)\}$.

Take a variable-only unification problem:

$$P = \{x \simeq^?_{\mathcal{R},\lambda} y,\ y \simeq^?_{\mathcal{R},\lambda} z,\ z \simeq^?_{\mathcal{R},\lambda} u\}.$$

Somewhat unexpectedly, $\{\{x \mapsto u,\ y \mapsto u,\ z \mapsto u\}\} \neq \text{MCSU}(P)$.

Completeness does not hold:

- $\{x \mapsto u, y \mapsto u, z \mapsto u\} \not\preceq \{x \mapsto a, y \mapsto b, z \mapsto c, u \mapsto d\}$,
- but $\{x \mapsto a, y \mapsto b, z \mapsto c, u \mapsto d\}$ is an $(\mathcal{R},\lambda)$-unifier of P.

The same would happen if MCSU were defined using $\precsim_{\mathcal{R},\lambda}$:

- $\{x \mapsto u, y \mapsto u, z \mapsto u\} \not\precsim_{\mathcal{R},\lambda} \{x \mapsto a, y \mapsto b, z \mapsto c, u \mapsto d\}$.

Proximity-based unification using classes

Our algorithm works in two steps:

In the first step (pre-unification), it tries to solve the unification problem.

The result of this step is either failure (in this case the problem is unsolvable), or a triple:

- substitution (pre-unifier) that gives an idea how variable instantiations would look if the problem eventually is solvable,
- a constraint between variables (always solvable, but having potentially infinitely many solutions), and
- a constraint between functions symbols and so called neighborhood variables (that stand for sets of symbols).

Proximity-based unification using classes

In the pre-unification step, failure happens for one of two possible reasons:

- arity clash between terms to be unified:
  $f(s_1, \ldots, s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1, \ldots, t_m)$, $n \neq m$, or
- the unification problem contains an occurrence cycle.

Proximity-based unification using classes

In the pre-unification step, failure happens for one of two possible reasons:

- arity clash between terms to be unified: $f(s_1, \dots, s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1, \dots, t_m)$, $n \neq m$, or
- the unification problem contains an occurrence cycle.

Occurrence cycle of length 0:

$$\begin{tikzcd} x_0 \arrow[r, phantom, "\simeq^?_{\mathcal{R},\lambda}"] \arrow[r, bend left, dashed] & s_0 \end{tikzcd}$$

Proximity-based unification using classes

In the pre-unification step, failure happens for one of two possible reasons:

- arity clash between terms to be unified:
  $f(s_1, \ldots, s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1, \ldots, t_m)$, $n \neq m$, or
- the unification problem contains an occurrence cycle.

Occurrence cycle of length 1:

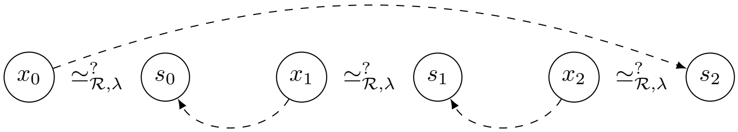
$$\begin{tikzcd}
x_0 \arrow[r, phantom, "\simeq^?_{\mathcal{R},\lambda}"] \arrow[rrr, dashed, bend left] & s_0 & x_1 \arrow[r, phantom, "\simeq^?_{\mathcal{R},\lambda}"] \arrow[l, dashed, bend left] & s_1
\end{tikzcd}$$

Proximity-based unification using classes

In the pre-unification step, failure happens for one of two possible reasons:

- arity clash between terms to be unified: $f(s_1, \ldots, s_n) \simeq^?_{\mathcal{R},\lambda} g(t_1, \ldots, t_m)$, $n \neq m$, or
- the unification problem contains an occurrence cycle.

Occurrence cycle of length 2:



Proximity-based unification using classes

In pre-unification, variable elimination is be done in a special way to take into account possible future proximities.

Proximity-based unification using classes

In pre-unification, variable elimination is be done in a special way to take into account possible future proximities.

For instance, in $\{x \simeq^?_{\mathcal{R},\lambda} f(y),\ x \simeq^?_{\mathcal{R},\lambda} g(h(z))\}$, we do not replace x by $f(y)$ and try to solve $f(y) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Proximity-based unification using classes

In pre-unification, variable elimination is be done in a special way to take into account possible future proximities.

For instance, in $\{x \simeq^?_{\mathcal{R},\lambda} f(y),\ x \simeq^?_{\mathcal{R},\lambda} g(h(z))\}$, we do not replace x by $f(y)$ and try to solve $f(y) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Instead, we "approximate" the structure of the instance of x, replacing x by $\mathbf{X}_f(y_1)$ and then try to solve $\mathbf{X}_f(y_1) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Remember: in basic signatures proximal terms have exactly the same structure (same set of positions).

Proximity-based unification using classes

In pre-unification, variable elimination is be done in a special way to take into account possible future proximities.

For instance, in $\{x \simeq^?_{\mathcal{R},\lambda} f(y),\ x \simeq^?_{\mathcal{R},\lambda} g(h(z))\}$, we do not replace x by $f(y)$ and try to solve $f(y) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Instead, we “approximate” the structure of the instance of x, replacing x by $\mathbf{X}_f(y_1)$ and then try to solve $\mathbf{X}_f(y_1) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Remember: in basic signatures proximal terms have exactly the same structure (same set of positions).

$\mathbf{X}_f$ is a new variable that is supposed to be instantiated by a set of function symbols (that are proximal to f), and y_1 is a new variable.

To make the relation between $\mathbf{X}_f$ and f clear, we add a new neighborhood constraint $\mathbf{X}_f \approx^?_{\mathcal{R},\lambda} f$.

Proximity-based unification using classes

In pre-unification, variable elimination is be done in a special way to take into account possible future proximities.

For instance, in $\{x \simeq^?_{\mathcal{R},\lambda} f(y),\ x \simeq^?_{\mathcal{R},\lambda} g(h(z))\}$, we do not replace x by $f(y)$ and try to solve $f(y) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Instead, we "approximate" the structure of the instance of x, replacing x by $\mathbf{X}_f(y_1)$ and then try to solve $\mathbf{X}_f(y_1) \simeq^?_{\mathcal{R},\lambda} g(h(z))$.

Remember: in basic signatures proximal terms have exactly the same structure (same set of positions).

$\mathbf{X}_f$ is a new variable that is supposed to be instantiated by a set of function symbols (that are proximal to f), and y_1 is a new variable.

To make the relation between $\mathbf{X}_f$ and f clear, we add a new neighborhood constraint $\mathbf{X}_f \approx^?_{\mathcal{R},\lambda} f$.

Also, the value of the new variable y_1 should be close to that of the old y: $y_1 \simeq^?_{\mathcal{R},\lambda} y$ is a new variable constraint.

Proximity-based unification using classes

In the second step, we try to solve neighborhood constraints.

They may have zero, one, or more (finitely many) solutions.

Each solution maps neighborhood variables to sets of symbols.

Proximity-based unification using classes

In the second step, we try to solve neighborhood constraints.

They may have zero, one, or more (finitely many) solutions.

Each solution maps neighborhood variables to sets of symbols.

Solutions of neighborhood constraints, combined with the pre-unifier, give the solutions of the original problem.

It gives a compact representation of the minimal complete set of approximate unifiers.

Proximity-based unification using classes

The pre-unification step, the proximity relation plays no role.

For $p(x,y,x)$ and $q(f(a),g(d),y)$, pre-unification returns:

- the pre-unifier $\{x \mapsto \mathbf{X}_f(\mathbf{X}_a), y \mapsto \mathbf{X}_g(\mathbf{X}_d)\}$
  applying to the initial problem, it gives

  $$p(\mathbf{X}_f(\mathbf{X}_a),\ \mathbf{X}_g(\mathbf{X}_d),\ \mathbf{X}_f(\mathbf{X}_a)) \simeq^?_{\mathcal{R},\lambda} q(f(a),\ g(d),\ \mathbf{X}_g(\mathbf{X}_d)).$$

- the empty variable constraint,
- the neighborhood constraint:

$$\begin{aligned}
&\{p \approx^?_{\mathcal{R},\lambda} q,\\
&\ \mathbf{X}_f \approx^?_{\mathcal{R},\lambda} f,\ \ \mathbf{X}_a \approx^?_{\mathcal{R},\lambda} a,\\
&\ \mathbf{X}_g \approx^?_{\mathcal{R},\lambda} g,\ \ \mathbf{X}_d \approx^?_{\mathcal{R},\lambda} d,\\
&\ \mathbf{X}_f \approx^?_{\mathcal{R},\lambda} \mathbf{X}_g,\ \ \mathbf{X}_a \approx^?_{\mathcal{R},\lambda} \mathbf{X}_d\}.
\end{aligned}$$

Proximity-based unification using classes

The proximity relation is needed in solving the neighborhood constraints.

Assume $\mathcal{R}_\lambda$: a —— b —— c —— d $\quad f$ —— g $\quad p$ —— q

Then the neighborhood constraint

$$
\begin{aligned}
&\{p \approx^?_{\mathcal{R},\lambda} q,\\
&\ \mathbf{X}_f \approx^?_{\mathcal{R},\lambda} f,\ \ \mathbf{X}_a \approx^?_{\mathcal{R},\lambda} a,\\
&\ \mathbf{X}_g \approx^?_{\mathcal{R},\lambda} g,\ \ \mathbf{X}_d \approx^?_{\mathcal{R},\lambda} d,\\
&\ \mathbf{X}_f \approx^?_{\mathcal{R},\lambda} \mathbf{X}_g,\ \ \mathbf{X}_a \approx^?_{\mathcal{R},\lambda} \mathbf{X}_d\}.
\end{aligned}
$$

is solved by the mapping:

$$\{\mathbf{X}_f \mapsto \{f, g\},\ \mathbf{X}_a \mapsto \{b\},\ \mathbf{X}_g \mapsto \{f, g\},\ \mathbf{X}_d \mapsto \{c\}\}.$$

Proximity-based unification using classes

Hence, the $(\mathcal{R}, \lambda)$-unification problem

$$p(x, y, x) \simeq^?_{\mathcal{R},\lambda} q(f(a), g(d), y)$$

is solved by the pair of an extended substitution and a variable constraint

$$\{x \mapsto \{f, g\}(\{b\}), y \mapsto \{f, g\}(\{c\})\} \parallel \emptyset$$

Proximity-based unification using classes

Hence, the $(\mathcal{R}, \lambda)$-unification problem

$$p(x, y, x) \simeq^?_{\mathcal{R},\lambda} q(f(a), g(d), y)$$

is solved by the pair of an extended substitution and a variable constraint

$$\{x \mapsto \{f, g\}(\{b\}), y \mapsto \{f, g\}(\{c\})\} \parallel \emptyset$$

$\{f, g\}(\{b\})$ an extended term, representing the set of terms

$$\mathsf{terms}(\{f, g\}(\{b\})) = \{f(b), g(b)\}.$$

Since the extended substitution represents a set of solutions, we cannot return a single unification degree.

Instead, it is possible to compute its upper and lower bounds.

Proximity-based unification using classes

For $(\mathcal{R}, \lambda)$-unification problem

$$p(x, x) \simeq^?_{\mathcal{R},\lambda} q(f(y), f(h(z)))$$

pre-unification gives

- the pre-unifier $\{x \mapsto \mathbf{X}_f(\mathbf{X}_h(z_1)),\ y \mapsto \mathbf{Y}(z_2)\}$,
  applying to the initial problem, it gives
  $$p(\mathbf{X}_f(\mathbf{X}_h(z_1)),\ \mathbf{X}_f(\mathbf{X}_h(z_1))) \simeq^?_{\mathcal{R},\lambda} q(f(\mathbf{Y}(z_2)),\ f(h(z))).$$
- the variable constraint $\{z_1 \simeq^?_{\mathcal{R},\lambda} z_2,\ z_1 \simeq^?_{\mathcal{R},\lambda} z\}$,
- the neighborhood constraint
  $\{p \approx^?_{\mathcal{R},\lambda} q,\ \mathbf{X}_f \approx^?_{\mathcal{R},\lambda} f,\ \mathbf{X}_h \approx^?_{\mathcal{R},\lambda} \mathbf{Y},\ \mathbf{X}_h \approx^?_{\mathcal{R},\lambda} h\}$.

Proximity-based unification using classes

Assume $\mathcal{R}_\lambda$:

$$\begin{tikzcd}[arrows=dash] & g_1 \arrow[dr] & \\ f \arrow[ur] \arrow[dr] & & h \\ & g_2 \arrow[ur] & \end{tikzcd} \qquad\qquad \begin{tikzcd}[arrows=dash] p \arrow[r] & q \end{tikzcd}$$

Then the neighborhood constraint

$$\{p \approx^?_{\mathcal{R},\lambda} q,\ \mathbf{X}_f \approx^?_{\mathcal{R},\lambda} f,\ \mathbf{X}_h \approx^?_{\mathcal{R},\lambda} \mathbf{Y},\ \mathbf{X}_h \approx^?_{\mathcal{R},\lambda} h\}$$

has three solutions:

$$\{\mathbf{X}_f \mapsto \{f, g_1, g_2\},\ \ \mathbf{X}_h \mapsto \{h\},\ \ \mathbf{Y} \mapsto \{h, g_1, g_2\}\}$$
$$\{\mathbf{X}_f \mapsto \{f, g_1, g_2\},\ \ \mathbf{X}_h \mapsto \{g_1\},\ \ \mathbf{Y} \mapsto \{g_1, f, h\}\}$$
$$\{\mathbf{X}_f \mapsto \{f, g_1, g_2\},\ \ \mathbf{X}_h \mapsto \{g_2\},\ \ \mathbf{Y} \mapsto \{g_2, f, h\}\}$$

Proximity-based unification using classes

Assume $\mathcal{R}_\lambda$:

$$\begin{tikzcd}[column sep=small, row sep=small] & g_1 \arrow[dr, dash] & \\ f \arrow[ur, dash] \arrow[dr, dash] & & h \\ & g_2 \arrow[ur, dash] & \end{tikzcd} \qquad \begin{tikzcd} p \arrow[r, dash] & q \end{tikzcd}$$

Consequently, $p(x,x) \simeq^?_{\mathcal{R},\lambda} q(f(y), f(h(z)))$ has three (compact) solutions:

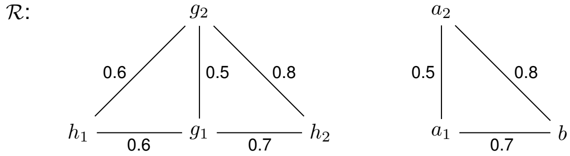
$$\{x \mapsto \{f, g_1, g_2\}(\{h\}(z_1)),\ y \mapsto \{h, g_1, g_2\}(z_2)\}$$
$$\quad \| \ \{z_1 \simeq^?_{\mathcal{R},\lambda} z_2,\ z_1 \simeq^?_{\mathcal{R},\lambda} z\}$$

$$\{x \mapsto \{f, g_1, g_2\}(\{g_1\}(z_1)),\ y \mapsto \{g_1, f, h\}(z_2)\}$$
$$\quad \| \ \{z_1 \simeq^?_{\mathcal{R},\lambda} z_2,\ z_1 \simeq^?_{\mathcal{R},\lambda} z\}$$

$$\{x \mapsto \{f, g_1, g_2\}(\{g_2\}(z_1)),\ y \mapsto \{g_2, f, h\}(z_2)\}$$
$$\quad \| \ \{z_1 \simeq^?_{\mathcal{R},\lambda} z_2,\ z_1 \simeq^?_{\mathcal{R},\lambda} z\}$$

Proximity-based matching using classes

The set-based compact representation (extended terms) is a convenient notation for formulating a matching algorithm.



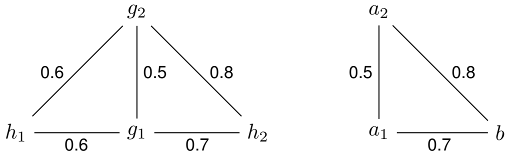
$\lambda = 0.6$:

$$
\begin{aligned}
&\{f(x,x) \precsim^{?}_{\mathcal{R},\lambda} f(g_1(a_1), g_2(a_2))\};\ \emptyset \Longrightarrow \\
&\{x \precsim^{?}_{\mathcal{R},\lambda} g_1(a_1),\ x \precsim^{?}_{\mathcal{R},\lambda} g_2(a_2)\};\ \emptyset \Longrightarrow \\
&\{x \precsim^{?}_{\mathcal{R},\lambda} g_2(a_2)\};\ \{x \approx \{g_1, h_1, h_2\}(\{a_1, b\})\} \Longrightarrow \\
&\emptyset;\ \{x \approx \{g_1, h_1, h_2\}(\{a_1, b\}),\ x \approx \{g_2, h_1, h_2\}(\{a_2, b\})\} \Longrightarrow \\
&\emptyset;\ \{x \approx \{h_1, h_2\}(\{b\})\}.
\end{aligned}
$$

Proximity-based matching using classes

The set-based compact representation (extended terms) is a convenient notation for formulating a matching algorithm.

$\mathcal{R}$:

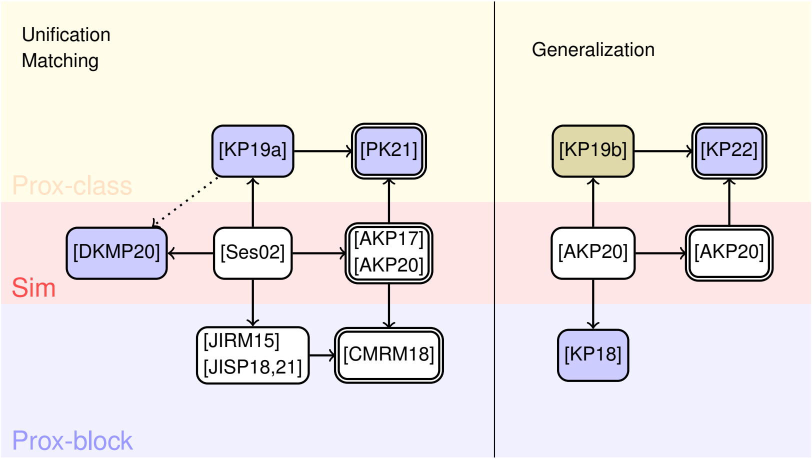


$\lambda = 0.8$:

$$
\begin{aligned}
&\{f(x,x) \precsim^{?}_{\mathcal{R},\lambda} f(g_1(a_1), g_2(a_2))\};\ \emptyset \Longrightarrow \\
&\{x \precsim^{?}_{\mathcal{R},\lambda} g_1(a_1),\, x \precsim^{?}_{\mathcal{R},\lambda} g_2(a_2)\};\ \emptyset \Longrightarrow \\
&\{x \precsim^{?}_{\mathcal{R},\lambda} g_2(a_2)\};\ \{x \approx \{g_1\}(\{a_1\})\};\ \Longrightarrow \\
&\emptyset;\ \{x \approx \{g_1\}(\{a_1\}),\ x \approx \{g_2, h_2\}(\{a_2, b\})\} \Longrightarrow \\
&\perp.
\end{aligned}
$$

Outline

Overview



Entries with double borders consider fully fuzzy signatures.

The problem statement, reformulated

Slight reformulation of the problem statement based on the solution representation form.

Given:

$\mathcal{R}$, λ, and two terms t and s.

Find:

An extended term $\mathbf{r}$ such that each $r \in \mathsf{terms}(\mathbf{r})$ is an $(\mathcal{R}, \lambda)$-least general generalization of t and s.

The problem statement, reformulated

Slight reformulation of the problem statement based on the solution representation form.

Given:

$\mathcal{R}$, λ, and two terms t and s.

Find:

An extended term $\mathbf{r}$ such that each $r \in \mathsf{terms}(\mathbf{r})$ is an $(\mathcal{R}, \lambda)$-least general generalization of t and s.

To compute $(\mathcal{R}, \lambda)$-lgg of t and s, take

$$\{x : \mathbf{ext}(t, \mathcal{R}, \lambda) \triangleq \mathbf{ext}(s, \mathcal{R}, \lambda)\}; x$$

and apply the anti-unification rules.

Example

$$\mathcal{R}: \quad f \overset{0.7}{\text{——}} g \qquad a_1 \overset{0.5}{\text{———}} a \overset{0.5}{\text{———}} a_2 \overset{0.6}{\text{———}} a' \overset{0.6}{\text{———}} a_3$$

$$b_1 \overset{0.5}{\text{———}} b \overset{0.5}{\text{———}} b_2 \overset{0.6}{\text{———}} b' \overset{0.6}{\text{———}} b_3$$

$t = f(a_1, a_2, a_3)$ and $s = g(b_1, b_2, b_3)$. Assume $\lambda = 0.5$.

The $(\mathcal{R}, \lambda)$-extended term versions of t and s are:

$$\mathbf{ext}(t, \mathcal{R}, 0.5) = \{f, g\}(\{a_1, a\}, \{a, a_2, a'\}, \{a', a_3\})$$
$$\mathbf{ext}(s, \mathcal{R}, 0.5) = \{f, g\}(\{b_1, b\},\ \{b, b_2, b'\},\ \{b', b_3\})$$

Two solutions:

1. $\{f, g\}(x, x, y),$ with $\{x : \{a\} \triangleq \{b\},\ y : \{a', a_3\} \triangleq \{b', b_3\}\}$.
2. $\{f, g\}(x, y, y),$ with $\{x : \{a_1, a\} \triangleq \{b_1, b\},\ y : \{a'\} \triangleq \{b'\}\}$.

Example

$$\mathcal{R}: \quad f \overset{0.7}{\text{———}} g \qquad a_1 \overset{0.5}{\text{———}} a \overset{0.5}{\text{———}} a_2 \overset{0.6}{\text{———}} a' \overset{0.6}{\text{———}} a_3$$

$$b_1 \overset{0.5}{\text{———}} b \overset{0.5}{\text{———}} b_2 \overset{0.6}{\text{———}} b' \overset{0.6}{\text{———}} b_3$$

$t = f(a_1, a_2, a_3)$ and $s = g(b_1, b_2, b_3)$. Assume $\lambda = 0.6$.

The $(\mathcal{R}, \lambda)$-extended term versions of t and s are:

$$\mathbf{ext}(t, \mathcal{R}, 0.6) = \{f, g\}(\{a_1\}, \{a_2, a'\}, \{a', a_3\})$$
$$\mathbf{ext}(s, \mathcal{R}, 0.6) = \{f, g\}(\{b_1\}, \{b_2, b'\}, \{b', b_3\})$$

One solution:

$\{f, g\}(x, y, y),$ with $\{x : \{a_1\} \triangleq \{b_1\},\ y : \{a'\} \triangleq \{b'\}\}$.

Example

$\mathcal{R}$: $f \overset{0.7}{\text{———}} g$ $\quad a_1 \overset{0.5}{\text{———}} a \overset{0.5}{\text{———}} a_2 \overset{0.6}{\text{———}} a' \overset{0.6}{\text{———}} a_3$

$b_1 \overset{0.5}{\text{———}} b \overset{0.5}{\text{———}} b_2 \overset{0.6}{\text{———}} b' \overset{0.6}{\text{———}} b_3$

$t = f(a_1, a_2, a_3)$ and $s = g(b_1, b_2, b_3)$. Assume $\lambda = 0.7$.

The $(\mathcal{R}, \lambda)$-extended term versions of t and s are:

$$\mathbf{ext}(t, \mathcal{R}, 0.7) = \{f, g\}(\{a_1\}, \{a_2\}, \{a_3\})$$
$$\mathbf{ext}(s, \mathcal{R}, 0.7) = \{f, g\}(\{b_1\},\ \{b_2\},\ \{b_3\})$$

One solution:

$\{f, g\}(x, y, z),$

with $\{x : \{a_1\} \triangleq \{b_1\},\ y : \{a_1\} \triangleq \{b_2\},\ z : \{a_3\} \triangleq \{b_3\}\}$.

Example

$\mathcal{R}$: $\quad f \overset{0.7}{\text{——}} g \qquad a_1 \overset{0.5}{\text{——}} a \overset{0.5}{\text{——}} a_2 \overset{0.6}{\text{——}} a' \overset{0.6}{\text{——}} a_3$

$\qquad\qquad\qquad\qquad b_1 \overset{0.5}{\text{——}} b \overset{0.5}{\text{——}} b_2 \overset{0.6}{\text{——}} b' \overset{0.6}{\text{——}} b_3$

$t = f(a_1, a_2, a_3)$ and $s = g(b_1, b_2, b_3)$. Assume $\lambda = 0.8$.

The $(\mathcal{R}, \lambda)$-extended term versions of t and s are:

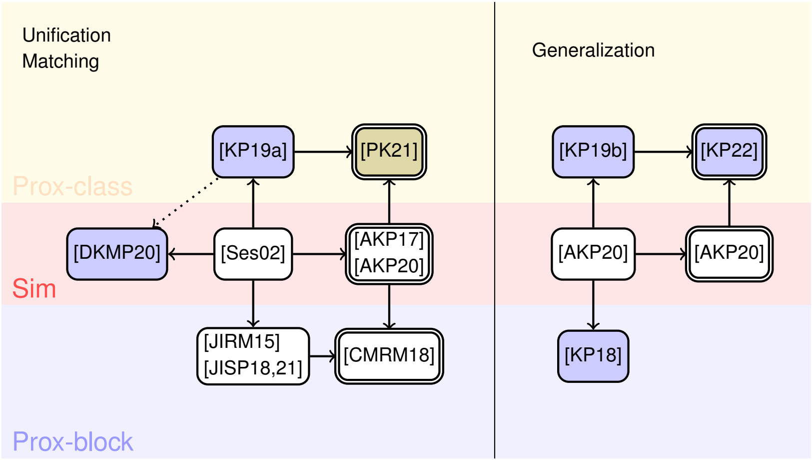
$$\mathbf{ext}(t, \mathcal{R}, 0.8) = \{f\}(\{a_1\}, \{a_2\}, \{a_3\})$$
$$\mathbf{ext}(s, \mathcal{R}, 0.8) = \{g\}(\{b_1\}, \{b_2\}, \{b_3\})$$

One solution:

x, with $\{x : \{f\}(\{a_1\}, \{a_2\}, \{a_3\}) \triangleq \{g\}(\{b_1\}, \{b_2\}, \{b_3\})\}$.

Outline

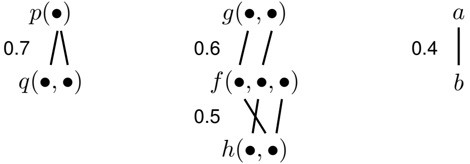
Overview



Entries with double borders consider fully fuzzy signatures.

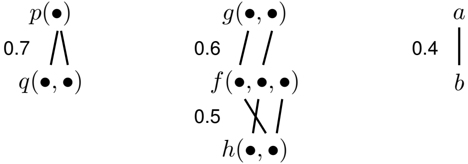
Unification using classes, fully fuzzy

An example to remind fully fuzzy signatures:



Unification using classes, fully fuzzy

An example to remind fully fuzzy signatures:



Unlike for basic signatures, in the fully fuzzy case proximal terms may have different structures.

The trick that worked with variable elimination in unification in basic signatures does not work here anymore.

Consequently, we can not represent solutions in a compact form.

Should revert to the explicit representation.

Unification using classes, fully fuzzy

Decomposition should take into account the argument relation.

DECOMPOSITION:

$$\{f(t_1,\dots,t_n) \simeq^{?}_{\mathcal{R},\lambda} g(s_1,\dots,s_m)\} \uplus P;\ \sigma;\ \alpha \Longrightarrow$$
$$P \cup \{t_i \simeq^{?}_{\mathcal{R},\lambda} s_j \mid (i,j) \in \rho\};\ \sigma;\ \alpha \wedge \beta,$$

where $n, m \geq 0$, $f \sim^{\rho}_{\mathcal{R},\beta} g$, and $\beta \geq \lambda$.

Unification using classes, fully fuzzy

For $x \simeq^?_{\mathcal{R},\lambda} t$, variable elimination replaces x with a term whose head is close to the head of t and whose arguments are fresh variables.

A lazy way of choosing a right term in the neighborhood of t.

This step is nondeterministic, since there might be more than one such right terms.

Unification using classes, fully fuzzy

VARIABLE ELIMINATION:

$$\{x \simeq^?_{\mathcal{R},\lambda} g(s_1, \ldots, s_n)\} \uplus P;\ \sigma;\ \alpha \Longrightarrow$$
$$P\vartheta \cup \{y_i \simeq^?_{\mathcal{R},\lambda} s_j \mid (i,j) \in \rho\};\ \sigma\vartheta;\ \alpha \wedge \beta,$$

where

- $\{x \simeq^?_{\mathcal{R},\lambda} g(s_1, \ldots, s_n)\} \uplus P$ contains no occurrence cycle for x,
- $\vartheta = \{x \mapsto f(y_1, \ldots, y_m)\}$ with fresh variables $y_1, \ldots, y_m$,
- $f \sim^{\rho}_{\mathcal{R},\beta} g$ with $\beta \geq \lambda$,
- $n, m \geq 0$.

Unification using classes, fully fuzzy

Other rules: TRIVIAL, ORIENT, CLASH, OCCURRENCE CHECK

Unification using classes, fully fuzzy

The rules work on triples $P; \sigma; \alpha$, called unification configurations, where

- P is a unification problem,
- σ is the substitution computed so far,
- α is the approximation degree, also computed so far.

The rules transform configurations into configurations.

We stop either with failure or once we reach a variables-only configuration:

$$\{x_1 \simeq^?_{\mathcal{R},\lambda} y_1, \ldots, x_n \simeq^?_{\mathcal{R},\lambda} y_n\};\ \sigma;\ \alpha, \quad n \geq 0$$

Unification using classes, fully fuzzy

The algorithm works for argument relations $\rho \subseteq N \times M$ that are correspondence relations, i.e. they are:

- left-total
  for all $i \in N$ there exists $j \in M$ such that $(i, j) \in \rho$;
- right-total
  for all $j \in M$ there exists $i \in N$ such that $(i, j) \in \rho$.

Unification using classes, fully fuzzy

The algorithm works for argument relations $\rho \subseteq N \times M$ that are correspondence relations, i.e. they are:

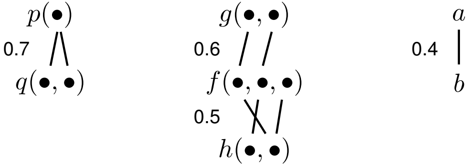
- left-total
  for all $i \in N$ there exists $j \in M$ such that $(i, j) \in \rho$;
- right-total
  for all $j \in M$ there exists $i \in N$ such that $(i, j) \in \rho$.

This is to make sure that failing with occurrence cycles does not lead to losing a solution.

Correspondence relations guarantee that proximal terms have the same set of variables and no term is close to its proper subterm.

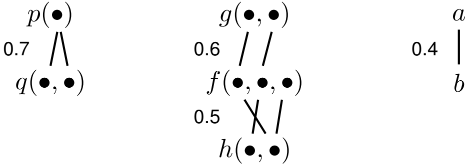
Unification using classes, fully fuzzy

The argument relation in this example is not correspondence:

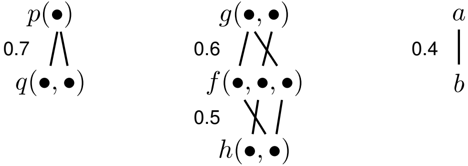


Unification using classes, fully fuzzy

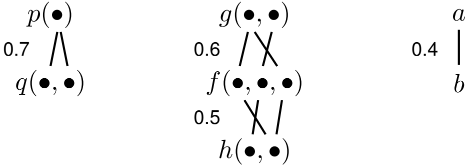
The argument relation in this example is not correspondence:



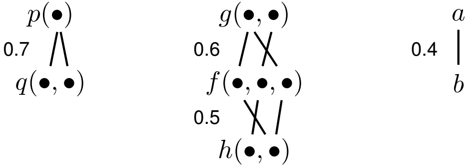
Here it is:



Unification using classes, fully fuzzy



Unification using classes, fully fuzzy

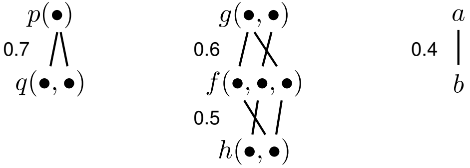


Unification problem: $P = \{p(x) \simeq^?_{\mathcal{R},0.4} q(g(u,y),h(z,u))\}$.

For P, the algorithm stops with the configuration

$$\{v_1 \simeq^?_{\mathcal{R},0.4} u,\ v_2 \simeq^?_{\mathcal{R},0.4} y,\ v_2 \simeq^?_{\mathcal{R},0.4} z,\ v_3 \simeq^?_{\mathcal{R},0.4} u\};$$
$$\{x \mapsto f(v_1,v_2,v_3)\};\ 0.5$$

Unification using classes, fully fuzzy



Unification problem: $P = \{p(x) \simeq^?_{\mathcal{R},0.4} q(g(u,a), h(z,u))\}$.

For P, the algorithm produces four final configurations:

$$\{v_1 \simeq^?_{\mathcal{R},0.4} u,\ \ v_3 \simeq^?_{\mathcal{R},0.4} u\}; \qquad \{v_1 \simeq^?_{\mathcal{R},0.4} u,\ \ v_3 \simeq^?_{\mathcal{R},0.4} u\};$$
$$\{x \mapsto f(v_1, a, v_3), z \mapsto a\};\ 0.5 \qquad \{x \mapsto f(v_1, b, v_3), z \mapsto a\};\ 0.4$$

$$\{v_1 \simeq^?_{\mathcal{R},0.4} u,\ \ v_3 \simeq^?_{\mathcal{R},0.4} u\}; \qquad \{v_1 \simeq^?_{\mathcal{R},0.4} u,\ \ v_3 \simeq^?_{\mathcal{R},0.4} u\};$$
$$\{x \mapsto f(v_1, a, v_3), z \mapsto b\};\ 0.4 \qquad \{x \mapsto f(v_1, b, v_3), z \mapsto b\};\ 0.5$$

Unifiability

The decision problem of class-based approximate unifiability with in fully fuzzy signatures is NP-hard.

It can be shown by a reduction from positive 1-in-3-SAT problem.

Unifiability

The decision problem of class-based approximate unifiability with in fully fuzzy signatures is NP-hard.

It can be shown by a reduction from positive 1-in-3-SAT problem.

In fact, the reduction shows that already a special case of unifiability (well-moded) is NP-hard.

Unification algorithm: properties

Theorem (Soundness)

Let $P; \varepsilon; 1 \Longrightarrow^* S; \sigma; \alpha$ *be a derivation performed by the unification algorithm where* $S; \sigma; \alpha$ *is a variables-only configuration.*

Let φ *be a unifier of* S *with the approximation degree* β*.*

Then $\sigma\varphi$ *is a unifier of* P *with the approximation degree* $\alpha \wedge \beta$*.*

Unification algorithm: properties

Theorem (Completeness)

Let P be a $(\mathcal{R}, \lambda)$-unification problem and ϑ be its unifier with the approximation degree β.

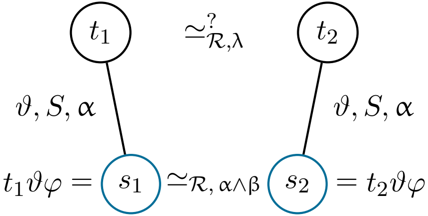
Then there exists a derivation $P; \varepsilon; 1 \Longrightarrow^* S; \sigma; \alpha$ by the unification algorithm, where

- $S; \sigma; \alpha$ *is a variables-only configuration with* $\alpha \geq \beta$ *and*
- *there is a unifier* φ *of* S *such that* $(\sigma\varphi)|_{var(P)} = \vartheta|_{var(P)}$.

Unification using classes, fully fuzzy

$$\boxed{t_1} \quad \simeq^{?}_{\mathcal{R},\lambda} \quad \boxed{t_2}$$

Unification using classes, fully fuzzy



If φ solves the variable-only constraint S with degree β then $\vartheta\varphi$ solves the unification problem $t_1 \simeq^?_{\mathcal{R},\lambda} t_2$ with degree $\alpha \wedge \beta$

Matching using classes, fully fuzzy

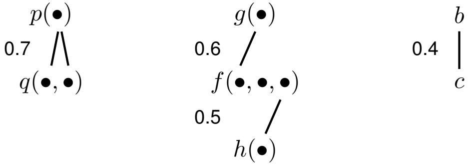
Unlike unification, we do not have to restrict argument relations for matching.

It may cause matchers to contain fresh variables.

Matching using classes, fully fuzzy

Unlike unification, we do not have to restrict argument relations for matching.

It may cause matchers to contain fresh variables.



Consider the matching problem $p(x) \precsim^?_{\mathcal{R},0.4} q(g(a), h(c))$.

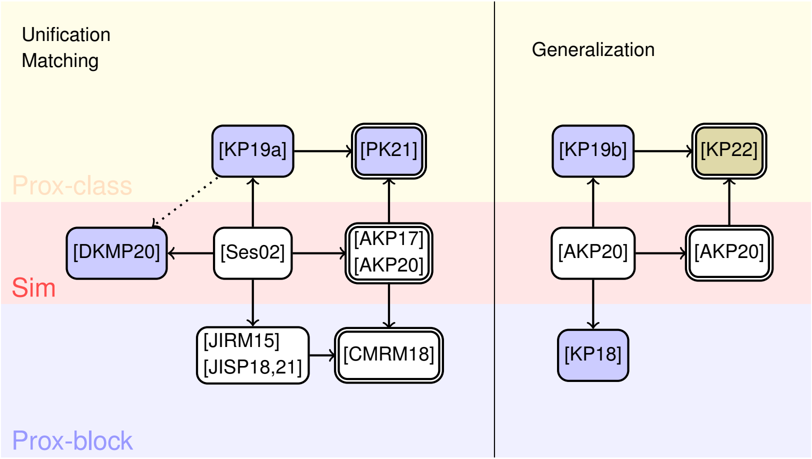
The matching algorithm returns two solutions:

$$\{x \mapsto f(a, v, c)\}; 0.5 \qquad\qquad \{x \mapsto f(a, v, b)\}; 0.4$$

where v is a fresh variable.

Outline

Overview



Entries with double borders consider fully fuzzy signatures.

Generalization using classes, fully fuzzy

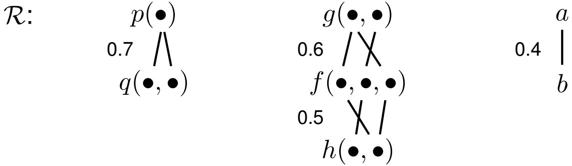
Again, no compact terms in fully fuzzy signatures: proximal terms may have different structures.

We compute t, α_1, α_2, and a representation from which σ_1 and σ_2 can be read.

t: a least general generalization
$X = t$ solves the anti-unification problem $X : t_1 \triangleq_{\mathcal{R},\lambda} t_2$
with degrees α_1 and α_2

$$\begin{tikzcd}
 & & t \arrow[dl, dash, "{\sigma_1, \alpha_1}"'] \arrow[dr, dash, "{\sigma_2, \alpha_2}"] & & \\
t\sigma_1 \simeq_{\mathcal{R},\,\alpha_1} & t_1 & \triangleq_{\mathcal{R},\lambda} & t_2 & \simeq_{\mathcal{R},\,\alpha_2} t\sigma_2
\end{tikzcd}$$

Generalization using classes, fully fuzzy

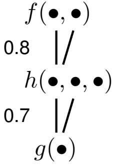


Given $\mathcal{R}$ and $\lambda = 0.4$, anti-unify $g(a, b)$ and $h(c, b)$.

One of the solutions: $f(a, x, a)$, where $x : b \triangleq c$, with the approximation degrees 0.6 for $g(a, b)$ and 0.4 for $h(c, b)$.

Generalization using classes, fully fuzzy

- $f \sim_{\mathcal{R},0.8}^{\{(1,1),(2,1)\}} h$.
- $h \sim_{\mathcal{R},0.7}^{\{(1,1),(2,1)\}} g$.
- $a \sim_{\mathcal{R},0.6}^{\emptyset} b$, $b \sim_{\mathcal{R},0.5}^{\emptyset} c$



Generalization using classes, fully fuzzy

- $f \sim_{\mathcal{R},0.8}^{\{(1,1),(2,1)\}} h$.
- $h \sim_{\mathcal{R},0.7}^{\{(1,1),(2,1)\}} g$.
- $a \sim_{\mathcal{R},0.6}^{\emptyset} b$, $\quad b \sim_{\mathcal{R},0.5}^{\emptyset} c$

$$\begin{array}{rc} & f(\bullet,\bullet) \\ 0.8 & |\!/ \\ & h(\bullet,\bullet,\bullet) \\ 0.7 & |\!/ \\ & g(\bullet) \end{array}$$

$(\mathcal{R}, 0.5)$-lggs of $f(a, c)$ and $g(b)$:
$h(b, a, \_)$ and $h(b, b, \_)$.

- lgg's can be comparable wrt $\precsim_{\mathcal{R},\lambda}$ (but not wrt $\preceq$),
- the irrelevant generalization argument is expressed by the anonymous variable _.

$$\begin{array}{rc} & f(a,c) \\ 0.5 & |\!/ \\ & h(b,a,\_) \\ 0.6 & |\!/ \\ & g(b) \end{array} \qquad \begin{array}{rc} & f(a,c) \\ 0.5 & |\!/ \\ & h(b,b,\_) \\ 0.7 & |\!/ \\ & g(b) \end{array}$$

Generalization using classes, fully fuzzy

- $f \sim_{\mathcal{R},0.8}^{\{(1,1),(2,1)\}} h$.
- $h \sim_{\mathcal{R},0.7}^{\{(1,1),(2,1)\}} g$.
- $a \sim_{\mathcal{R},0.6}^{\emptyset} b, \quad b \sim_{\mathcal{R},0.5}^{\emptyset} c$

$$f(\bullet,\bullet)$$
0.8
$$h(\bullet,\bullet,\bullet)$$
0.7
$$g(\bullet)$$

$(\mathcal{R}, 0.6)$-lgg of $f(a,c)$ and $g(b)$: x.

- It can not be $h(y, b, \_)$, because y can not be instantiated by a term that is $(\mathcal{R}, 0.6)$-close to both a and c.
- The set $\{a, c\}$ is $(\mathcal{R}, 0.6)$-inconsistent

$$f(a,c)$$
1
$$x$$
1
$$g(b)$$

Generalization using classes, fully fuzzy

Peculiarities of proximity-based fully fuzzy anti-unification using classes:

- nonstandard variable merging (also in basic signatures)
- irrelevant position abstraction
- look-ahead consistency check of arguments

The rules of our algorithm deal with them.

Generalization using classes, fully fuzzy

Rules:

- TRIVIAL: abstracts irrelevant positions by anonymous variables.
- DECOMPOSITION: adds a new symbol to the generalization, performs consistency check.
- SOLVE: keeps a variable in the generalization when there is no other way.
- MERGE: merges the generalization variables if they generalize proximal terms.

Decomposition rule

$$\begin{aligned}
&\{x : T_1 \triangleq T_2\} \uplus A; S; r; \alpha_1; \alpha_2 \Longrightarrow \\
&\quad \{y_i : Q_{i1} \triangleq Q_{i2} \mid 1 \le i \le n\} \cup A; S; \\
&\quad r\{x \mapsto h(y_1, \ldots, y_n)\}; \\
&\quad \min\{\alpha_1, \beta_1\};\ \min\{\alpha_2, \beta_2\}
\end{aligned}$$

where $T_1 \cup T_2 \neq \emptyset$; h is n-ary with $n \geq 0$; $y_1, \ldots, y_n$ are fresh; and for $j = 1, 2$, if $T_j = \{t^j_1, \ldots, t^j_{m_j}\}$, then

- $h \sim^{\rho^j_k}_{\mathcal{R},\gamma^j_k} head(t^j_k)$ with $\gamma^j_k \geq \lambda$ for all $1 \leq k \leq m_j$ and $\beta_j = \min\{\gamma^j_1, \ldots, \gamma^j_{m_j}\}$ (note that $\beta_j = 1$ if $m_j = 0$),
- for all $1 \leq i \leq n$, $Q_{ij} = \cup_{k=1}^{m_j}\{t^j_k|_q \mid (i, q) \in \rho^j_k\}$ and is $(\mathcal{R}, \lambda)$-consistent.

Generalization using classes, fully fuzzy

Example

$$h \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},0.8} f, \quad h \sim^{\{(1,1),(2,1)\}}_{\mathcal{R},0.7} g, \quad a \sim^{\emptyset}_{\mathcal{R},0.6} b, \quad b \sim^{\emptyset}_{\mathcal{R},0.5} c.$$

Computing an $(\mathcal{R}, 0.5)$-lgg $h(b, a, \_)$ of $f(a, c)$ and $g(a)$.

$$\{x : \{f(a,c)\} \triangleq \{g(a)\}\}; \emptyset; x; 1; 1$$

Generalization using classes, fully fuzzy

Example

$$h \sim_{\mathcal{R},0.8}^{\{(1,1),(1,2)\}} f, \quad h \sim_{\mathcal{R},0.7}^{\{(1,1),(2,1)\}} g, \quad a \sim_{\mathcal{R},0.6}^{\emptyset} b, \quad b \sim_{\mathcal{R},0.5}^{\emptyset} c.$$

Computing an $(\mathcal{R}, 0.5)$-lgg $h(b, a, \_)$ of $f(a, c)$ and $g(a)$.

$$\{x : \{f(a,c)\} \triangleq \{g(a)\}\}; \emptyset; x; 1; 1$$

$$\big\downarrow \text{Dec}$$

$$\{y_1 : \{a, c\} \triangleq \{a\}, y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(y_1, y_2, y_3); 0.8; 0.7$$

Generalization using classes, fully fuzzy

Example

$$h \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},0.8} f, \quad h \sim^{\{(1,1),(2,1)\}}_{\mathcal{R},0.7} g, \quad a \sim^{\emptyset}_{\mathcal{R},0.6} b, \quad b \sim^{\emptyset}_{\mathcal{R},0.5} c.$$

Computing an $(\mathcal{R}, 0.5)$-lgg $h(b, a, \_)$ of $f(a, c)$ and $g(a)$.

$$\{x : \{f(a,c)\} \triangleq \{g(a)\}\}; \emptyset; x; 1; 1$$

$$\Big\downarrow \text{Dec}$$

$$\{y_1 : \{a,c\} \triangleq \{a\}, y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(y_1, y_2, y_3); 0.8; 0.7$$

$$\Big\downarrow \text{Dec}$$

$$\{y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(b, y_2, y_3); 0.5; 0.6$$

Generalization using classes, fully fuzzy

Example

$$h \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},0.8} f, \quad h \sim^{\{(1,1),(2,1)\}}_{\mathcal{R},0.7} g, \quad a \sim^{\emptyset}_{\mathcal{R},0.6} b, \quad b \sim^{\emptyset}_{\mathcal{R},0.5} c.$$

Computing an $(\mathcal{R}, 0.5)$-lgg $h(b, a, \_)$ of $f(a, c)$ and $g(a)$.

$$\begin{array}{c}
\{x : \{f(a,c)\} \triangleq \{g(a)\}\}; \emptyset; x; 1; 1 \\
\Big\downarrow \text{Dec} \\
\{y_1 : \{a,c\} \triangleq \{a\}, y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(y_1, y_2, y_3); 0.8; 0.7 \\
\Big\downarrow \text{Dec} \\
\{y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(b, y_2, y_3); 0.5; 0.6 \\
\Big\downarrow \text{Dec} \\
\{y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(b, a, y_3); 0.5; 0.6
\end{array}$$

Generalization using classes, fully fuzzy

Example

$$h \sim_{\mathcal{R},0.8}^{\{(1,1),(1,2)\}} f, \quad h \sim_{\mathcal{R},0.7}^{\{(1,1),(2,1)\}} g, \quad a \sim_{\mathcal{R},0.6}^{\emptyset} b, \quad b \sim_{\mathcal{R},0.5}^{\emptyset} c.$$

Computing an $(\mathcal{R}, 0.5)$-lgg $h(b, a, \_)$ of $f(a, c)$ and $g(a)$.

$$\{x : \{f(a,c)\} \triangleq \{g(a)\}\}; \emptyset; x; 1; 1$$
$$\Big\downarrow \text{Dec}$$
$$\{y_1 : \{a, c\} \triangleq \{a\}, y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(y_1, y_2, y_3); 0.8; 0.7$$
$$\Big\downarrow \text{Dec}$$
$$\{y_2 : \emptyset \triangleq \{a\}, y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(b, y_2, y_3); 0.5; 0.6$$
$$\Big\downarrow \text{Dec}$$
$$\{y_3 : \emptyset \triangleq \emptyset\}; \emptyset; h(b, a, y_3); 0.5; 0.6$$
$$\Big\downarrow \text{Tri}$$
$$\emptyset; \emptyset; h(b, a, \_); 0.5; 0.6$$

Family of algorithms

Some features of proximity-based fully fuzzy anti-unification:

- nonstandard variable merging (also in basic signatures)
- irrelevant position abstraction
- look-ahead consistency check of arguments

Family of algorithms

Some features of proximity-based fully fuzzy anti-unification:

- nonstandard variable merging (also in basic signatures)
  Not needed for linear generalizations
- irrelevant position abstraction
- look-ahead consistency check of arguments

Family of algorithms

Some features of proximity-based fully fuzzy anti-unification:

- nonstandard variable merging (also in basic signatures)
  Not needed for linear generalizations
- irrelevant position abstraction
  Not needed if argument relations are left- and right-total
- look-ahead consistency check of arguments

Family of algorithms

Some features of proximity-based fully fuzzy anti-unification:

- nonstandard variable merging (also in basic signatures)
  Not needed for linear generalizations
- irrelevant position abstraction
  Not needed if argument relations are left- and right-total
- look-ahead consistency check of arguments
  Not needed if argument relations are (partial) injective functions

Family of algorithms

Some features of proximity-based fully fuzzy anti-unification:

- nonstandard variable merging (also in basic signatures)
  Not needed for linear generalizations
- irrelevant position abstraction
  Not needed if argument relations are left- and right-total
- look-ahead consistency check of arguments
  Not needed if argument relations are (partial) injective functions

Combinations lead to eight different algorithms, obtained from the general set of rules in a modular way.

They differ from each other by the decomposition rule.

Each of them computes the respective minimal complete sets of generalizations, together with their approximation degree upper bounds.

Outline

Directions for future research

Not in particular order:

- Generic treatment of T-norms.
- Approximate unification and anti-unification modulo background theories (similar to crisp equational unification / anti-unification).
- Relating to a recently introduced framework of quantitative and metric rewriting (Gavazzo & del Florio, POPL'23).
- In the proximity setting, computing a best solution (by some criterion), instead of all solutions or some arbitrarily chosen ones ($\longrightarrow$ optimization?).
- Investigating the applicability of proximity-based anti-unification for approximate clone detection, chatbot development, or program repair.