Fuzzy Logic Programming based on Weak Unification: concepts, implementation and applications.

Pascual Julián-Iranzo[1]

[1]Department of Information Technologies and Systems,
University of Castilla-La Mancha, Spain.
(**Pascual.Julian@uclm.es**)

Overview of this talk

- There are two characteristics of programming languages that are going to be greatly appreciated in the future:
  1. their ability to answer flexibly to questions
  2. the possibility of modeling taxonomies of terms.
- This talk tries to show how both characteristics can be integrated in **fuzzy logic programming** languages through the unified concept of **proximity/similarity** relation.
- We present some of the benefits of this integration through a series of simple programs.

Outline

1. **Introduction**
   - Fuzzy Logic Programming
   - Bousi∼Prolog general features
2. **Bousi∼Prolog Fundamentals and its Implementation**
   - Proximity Relations and Similarity Relations
   - The Similarity-based Unification Algorithm
   - Pros and Cons of Proximity Relations
   - Proximity Blocks vs. Proximity Classes
   - A New Notion of Proximity Between Expressions
   - An Efficient Proximity-based Unification Algorithm
   - Weak SLD Resolution
3. **Some Bousi∼Prolog Applications**
   - Pattern Matching in Strings
   - Flexible Query Answering in Deductive Databases
   - Information Retrieval
   - Approximate Reasoning
4. **Conclusions**

Outline

1 Introduction
- Fuzzy Logic Programming
- Bousi~Prolog general features

2 Bousi~Prolog Fundamentals and its Implementation
- Proximity Relations and Similarity Relations
- The Similarity-based Unification Algorithm
- Pros and Cons of Proximity Relations
- Proximity Blocks vs. Proximity Classes
- A New Notion of Proximity Between Expressions
- An Efficient Proximity-based Unification Algorithm
- Weak SLD Resolution

3 Some Bousi~Prolog Applications
- Pattern Matching in Strings
- Flexible Query Answering in Deductive Databases
- Information Retrieval
- Approximate Reasoning

4 Conclusions

Fuzzy Logic Programming

- **Fuzzy Logic Programming** = Logic Prog. + Fuzzy Logic
- Born as early as the seventies (past century) *[Lee-72]*.
- There is no a standard language. Several approaches:
  1. SLD-resolution + **weak unification**

     Likelog *[Fontana & Formato-99]*; **SiLog** *[M. Sessa-01]*; **Bousi∼Prolog** *[Julián et al-08] [Julián & Sáenz-23]*
  2. **FUZZY inference** + syntactic unification

     Fril *[Baldwin et al-84]*; **f-Prolog** *[Vojtáš & Paulík-96] [Vojtáš -01]*; **MALP** *[Ojeda et al-01& -04]*; **Fuzzy Prolog** *[S. Muñoz et al-04]*
  3. **FUZZY inference** + **weak unification**

     FASILL *[Moreno & Julián-14]*

Fuzzy Logic Programming and Bousi Prolog

- SLD-resolution + **weak unification**
- **Bousi∼Prolog** (BPL) is a fuzzy logic programming language whose main objective is to make flexible the query answering process.
- BPL is a conservative extension of Prolog, introducing as many fuzzy features as possible while maintaining most of the Prolog syntax.

Bousi Prolog general features

- **One distinguished feature** of Bousi∼Prolog is that it makes a separate treatment of Vague Knowledge.

Algorithm = Logic + Vague Knowledge + Control.

- **Logic**: is specified by (possibly graded) facts and rules (most of which respect the Prolog syntax).
- **Vague Knowledge**: is specified by proximity equations (and/or directives defining fuzzy subsets).
- **Control**: is implemented by an operational semantics based on Weak SLD Resolution (= SLD Resolution + Weak Unification + Grade composition).

Bousi Prolog general features

Example

```
% FACTS
likes_teaching(john, physics).
likes_teaching(mary, chemistry).
has_degree(john, physics).
has_degree(mary, chemistry).

% RULE
can_teach(X,M):-has_degree(X, M), likes_teaching(X, M).
```

```
?- can_teach(X,maths).
```

No answers !!

Bousi Prolog general features

Example

% FACTS

```
likes_teaching(john, physics).
likes_teaching(mary, chemistry).
has_degree(john, physics).
has_degree(mary, chemistry).
```

% PROXIMITY EQUATIONS

```
physics ~ maths = 0.8.
physics ~ chemistry = 0.8.
chemistry ~ maths = 0.6.
```

% RULE

```
can_teach(X,M):-has_degree(X, M), likes_teaching(X, M).
```

```
?- can_teach(X,maths).
```

X = john With approximation degree: 0.8 ;

X = mary With approximation degree: 0.6.

Bousi Prolog general features

Example

```
% FACTS
likes_teaching(john, physics) with 0.75.
likes_teaching(mary, chemistry) with 0.5.
has_degree(john, physics).
has_degree(mary, chemistry).

% PROXIMITY EQUATIONS
physics ~ maths = 0.8.
physics ~ chemistry = 0.8.
chemistry ~ maths = 0.6.

% RULE
can_teach(X,M):-has_degree(X, M), likes_teaching(X, M) with 0.9.
```

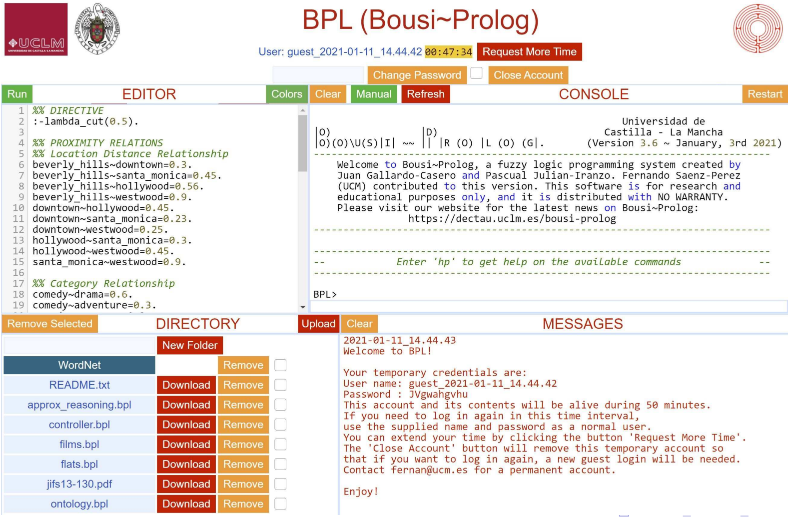
```
?- can_teach(X,maths).
```

X = john With approximation degree: 0.75 ;
X = mary With approximation degree: 0.5.

Bousi∼Prolog general features and implementation

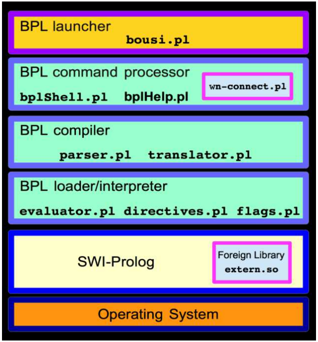
- **The BPL system** is an implementation of Bousi∼Prolog.
- It is a high level implementation system: compiles BPL programs into Prolog code which is executed by SWI-Prolog.
- It is publicly available at: `https://dectau.uclm.es/bousi-prolog/`
- Also available as an online interface: `https://dectau.uclm.es:8443`

A screenshot of the online interface



Architecture of the BPL system.

The Bousi∼Prolog system is composed of three subsystems which are integrated by a total of nine modules.



- The **bousi** module initializes the system.
- The **bplShell** module: command processing functionalities.
- The **parser** module: lexical, syntactic and semantic analysis of the BPL programs and queries.
- The **translator** module: translates the BPL source files and queries into TPL code. It relies on the **parser** module.
- The **evaluator** module: executes the TPL code. Implements the loader/interpreter of the BPL system.
- Modules with specific tasks: **bplHelp**, **directives**, **flags** and **foreign**.

Outline

1 Introduction
- Fuzzy Logic Programming
- Bousi∼Prolog general features

2 Bousi∼Prolog Fundamentals and its Implementation
- Proximity Relations and Similarity Relations
- The Similarity-based Unification Algorithm
- Pros and Cons of Proximity Relations
- Proximity Blocks vs. Proximity Classes
- A New Notion of Proximity Between Expressions
- An Efficient Proximity-based Unification Algorithm
- Weak SLD Resolution

3 Some Bousi∼Prolog Applications
- Pattern Matching in Strings
- Flexible Query Answering in Deductive Databases
- Information Retrieval
- Approximate Reasoning

4 Conclusions

Proximity Relations and Similarity Relations

- A binary fuzzy relation $\mathcal{R}$ on U is a mapping $\mathcal{R} : U \times U \to [0, 1]$.
- Some important properties fuzzy relations may have:
  1. **(Reflexive)** $\mathcal{R}(x, x) = 1$ for any $x \in U$;
  2. **(Symmetric)** $\mathcal{R}(x, y) = \mathcal{R}(y, x)$ for any $x, y \in U$;
  3. **(Transitive)** $\mathcal{R}(x, z) \geq \mathcal{R}(x, y) \triangle \mathcal{R}(y, z)$ for any $x, y, z \in U$;

  where $\triangle$ in any t-norm. When $\triangle \equiv \wedge$ (i.e., the minimum t-norm): **min-transitive**.
- **Proximity relations**: fuzzy binary relations fulfilling the reflexive and symmetric properties.
- **Similarity relations**: transitive proximity relations. Extension of the classical notion of equivalence relation.

Similarity relations on syntactic domains

- In classical Logic Programming different syntactic symbols represent distinct information.
- This restriction can be relaxed by introducing a similarity relation $\mathcal{R}$ defined on the alphabet of a first order language.
- Then, it can be extended to expressions (terms and atomic formulas) by structural induction:
  - $\hat{\mathcal{R}}(x, y) = 1$, if x and y are variables and $x \equiv y$.
  - $\hat{\mathcal{R}}(f(t_1, \ldots, t_n), g(s_1, \ldots, s_n)) = \mathcal{R}(f, g) \wedge (\bigwedge_{i=1}^{n} \hat{\mathcal{R}}(t_i, s_i))$, if f and g are function symbols and $t_1, \ldots, t_n$, $s_1, \ldots, s_n$ are terms.
  - $\hat{\mathcal{R}}(p(t_1, \ldots, t_n), q(s_1, \ldots, s_n)) = \mathcal{R}(p, q) \wedge (\bigwedge_{i=1}^{n} \hat{\mathcal{R}}(t_i, s_i))$, if p and q are predicate symbols and $t_1, \ldots, t_n$, $s_1, \ldots, s_n$ are terms.

Proximity/Similarity relations on syntactic domains

Example

- Given the fuzzy relation $\mathcal{R}$:

$$\mathcal{R}(p, q) = 0.6, \quad \mathcal{R}(a, b) = 0.5, \quad \mathcal{R}(b, c) = 0.4$$

- We can check the similarity of two terms using the extended relation $\hat{\mathcal{R}}$:

$$\hat{\mathcal{R}}(p(c), q(a)) = \mathcal{R}(p, q) \wedge \hat{\mathcal{R}}(c, a) = 0.6 \wedge \mathcal{R}(c, a) = 0.6 \wedge 0.4 = 0.4$$

Fuzzy Relations and Proximity Equations

- In Bousi∼Prolog fuzzy relations a syntactically represented by "proximity equations".

- **Proximity equation**:

  $< symbol > \sim < symbol > = < degree >$

  - Formally, is an entry defining a fuzzy binary relation $\mathcal{R}$
  - In practice, the built-in symbol "$\sim$" is a compressed notation for the symmetric closure of $\mathcal{R}$
  - "$a \sim b = \alpha$" means that a is close to b and b is close to a with degree α: $\mathcal{R}(a, b) = \alpha$ plus $\mathcal{R}(b, a) = \alpha$.

- Proximity Equations can express vague knowledge.

Fuzzy Relations and Proximity Equations

- A proximity or similarity relation $\mathcal{R}$ can be **partially specified**:

Example (3)

p ∼ q = 0.6.   a ∼ b = 0.5.   b ∼ c = 0.4.

- In fact, the above proximity equations are entries of a fuzzy relation which are internally represented, e.g., as `sim(p, q, 0.6)`.
- It will depend on the "transitivity" directive whether the fuzzy relation will become a proximity or a similarity relation.

Fuzzy Relations and Proximity Equations

- **The transitivity directive** has the following syntax:

```
:- transitivity([option]).
```

Option	Relation type	T-norm
yes	Similarity	Minim
no	Proximity	N/A
min	Similarity	Minim
luka	Similarity	Łukasiewicz
product	Similarity	Product
⋮	⋮	⋮

- **By default**: **:- transitivity(no).**

Fuzzy Relations and Proximity Equations

- Use "**:- transitivity(yes).**" If a similarity relation is needed.

Example (Computing a similarity relation)

For the partial specified fuzzy relation in Ex.3, the reflexive, symmetric, transitive closure is obtained.

	p	q	a	b	c
p	1	0.6	0	0	0
q	0.6	1	0	0	0
a	0	0	1	0.5	0.4
b	0	0	0.5	1	0.4
c	0	0	0.4	0.4	1

- We use an adaptation of the **Warshall algorithm**.

The Similarity-based Unification Algorithm

- For a similarity relation on a syntactic domain, $\mathcal{R}$, it is possible to define a fuzzy notion of a most general unifier (w.m.g.u.) of level λ (or λ-wmgu) of two expressions.
- For a Cut Value $\lambda > 0$, θ is a **λ-unifier** of t_1 and t_2 iff $\hat{\mathcal{R}}(t_1\theta, t_2\theta) > \lambda$.
- The weak unification algorithm [Sessa-02]:
  - $\{f(t_1, \ldots, t_n) \approx g(s_1, \ldots, s_n)\}$ weakly unifies (at a level λ) iff $\mathcal{R}(f, g) > \lambda$ and $\{t_1 \approx s_1, \ldots, t_n \approx s_n)\}$ weakly unifies (at a level λ).

  Output: a weak mgu of level λ, which is a substitution, plus an approximation degree.
- Note that it computes a representative of a class of wmgus.

The Similarity-based Unification Algorithm

Example (Find a 0.3-wmgu for $f(h(X)), k(Y))$ and $g(Z, j(Y)))$)

Assume $\mathcal{R}(f, g) = 0.8, \mathcal{R}(h, j) = 0.6, \mathcal{R}(h, k) = 0.3, \mathcal{R}(j, k) = 0.5$:

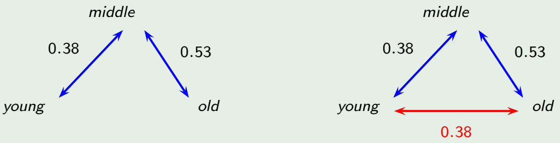
Unification problem	Weak Unifier	Degree
$\{f(h(X)), k(Y)) \approx g(Z, j(Y))\}$	$\{\}$	1
$\{h(X) \approx Z, k(Y) \approx j(X)\}$	$\{\}$	$1 \wedge 0.8$
$\{k(Y) \approx j(X)\}$	$\{Z/h(X)\}$	0.8
$\{Y \approx X\}$	$\{Z/h(X)\}$	$0.8 \wedge 0.5$
$\{\}$	$\{Z/h(X), Y/X\}$	0.5

Observe that it does not exists a w.m.g.u. of level 0.8.

Pros and Cons of Proximity Relations

- Bousi∼Prolog allows the use of proximity relations as a feature of its fuzzy unification algorithm.
- Several motivations for using proximity relations:
  - **1. The exclusive use of similarity relations may cause wrong modeling of vague information.**

Example

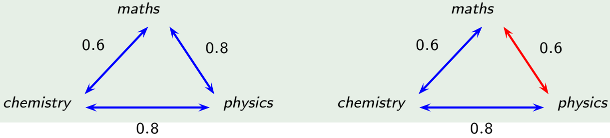


$\hat{\mathcal{R}}(young, old) \geq \hat{\mathcal{R}}(young, middle) \wedge \hat{\mathcal{R}}(middle, old) = 0.38 \wedge 0.53 = 0.38$

Pros and Cons of Proximity Relations

- Bousi∼Prolog allows the use of proximity relations as a feature of its fuzzy unification algorithm.
- Several motivations for using proximity relations:
  - **2. The transitivity constrains imposed by similarity relations may produce conflicts with user's specifications.**

Example



Pros and Cons of Proximity Relations

- Bousi∼Prolog allows the use of proximity relations as a feature of its fuzzy unification algorithm.
- Several motivations for using proximity relations:
- **3. Proximity relations are necessary to define "semantic unification" in terms of a weak unification algorithm.**

Example

Fuzzy subsets ⇒ **Standard matching functions** ⇒ **Proximity relation**

Pros and Cons of Proximity Relations

- The use of proximity relations increases the expressive power of the language and it is critical in order to give support to certain problems.
- However, a naïve treatment of proximity relations may cause unexpected severe problems.

  It is not suitable a direct combination of proximity relations with Sessa's unification algorithm.

  It may cause the incompleteness of Sessa's unification algorithm and the similarity-based SLD resolution procedure.

Pros and Cons of Proximity Relations

Example

Given $t_1 \equiv p(x, x)$ and $t_2 \equiv p(a, c)$ and the proximity $\mathcal{R} = \{\mathcal{R}(a, b) = 0.8, \mathcal{R}(b, c) = 0.75\}$,

- $\theta = \{x/b\}$ is a unifier of t_1 and t_2, with an approximation degree 0.75.
- However, Sessa's weak unification algorithm ends with failure:

$$\langle\{\underline{p(x,x) \approx p(a,c)}\}, id, 1\rangle \Rightarrow \langle\{\underline{x \approx a}, x \approx c\}, id, 1\rangle$$
$$\Rightarrow \langle\{\underline{a \approx c}\}, \{x/a\}, 1\rangle \Rightarrow \mathit{fail}$$

- Hence, Sessa's weak unification algorithm turns incomplete with proximity relations. This may lead to the incompleteness of the weak SLD resolution procedure.

Pros and Cons of Proximity Relations

- Moreover, also the cut rule

$$\Gamma \vdash \mathcal{A} \text{ and } \Gamma \cup \{\mathcal{A}\} \vdash \mathcal{B} \text{ imply } \Gamma \vdash \mathcal{B}$$

is not fulfilled.

Example

Given $\Pi = \{p(x, x).\}$ and the proximity $\mathcal{R} = \{\mathcal{R}(a, b) = 0.8, \mathcal{R}(b, c) = 0.75\}$. It is easy to check that:

- $\Pi, \mathcal{R} \vdash p(b, b)$, since $\leftarrow p(b, b) \overset{id,1}{\Rightarrow}_{\text{WSLD}} \Box$.
- $\Pi \cup \{p(b, b)\}, \mathcal{R} \vdash p(c, a)$, since $\leftarrow p(c, a) \overset{id,0.75}{\Rightarrow}_{\text{WSLD}} \Box$. Because $\langle\{c \approx b, a \approx b\}, id, 1\rangle \Rightarrow \langle\{a \approx b\}, id, 0.75\rangle \Rightarrow \langle\{\}, id, 0.75\rangle$

Pros and Cons of Proximity Relations

- Moreover, also the cut rule

$$\Gamma \vdash \mathcal{A} \text{ and } \Gamma \cup \{\mathcal{A}\} \vdash \mathcal{B} \text{ imply } \Gamma \vdash \mathcal{B}$$

  is not fulfilled.

Example

Given $\Pi = \{p(x,x).\}$ and the proximity $\mathcal{R} = \{\mathcal{R}(a,b) = 0.8, \mathcal{R}(b,c) = 0.75\}$. It is easy to check that:

- However, $\Pi, \mathcal{R} \not\vdash \leftarrow p(c,a)$, since the unification of $p(c,a)$ and $p(x_1,x_1)$ ends with failure:

$$\langle\{c \approx x_1, a \approx x_1\}, id, 1\rangle \Rightarrow \langle\{a \approx c\}, \{x_1/c\}, 1\rangle \Rightarrow \mathit{fail}$$

Pros and Cons of Proximity Relations

- Moreover, also the cut rule

$$\Gamma \vdash \mathcal{A} \text{ and } \Gamma \cup \{\mathcal{A}\} \vdash \mathcal{B} \text{ imply } \Gamma \vdash \mathcal{B}$$

is not fulfilled.

Example

Given $\Pi = \{p(x, x).\}$ and the proximity $\mathcal{R} = \{\mathcal{R}(a, b) = 0.8, \mathcal{R}(b, c) = 0.75\}$. It is easy to check that:

- The cut property, necessary for a reasonable logical consequence relation, is broken.

Pros and Cons of Proximity Relations

- To take advantage of proximity relations, but avoiding their problems, it is necessary:
  1. An accurate notion of proximity between terms and atoms of a first order language.
  2. An efficient implementation of the weak unification algorithm based on that notion of proximity.
- **To fulfill these goals we need more knowledge about proximity relations.**

Proximity Levels

- A proximity relation is characterized by a set $\Lambda = \{\lambda_1, ..., \lambda_n\}$ of approximation levels.

Example

Given $\{\mathcal{R}(a,a) = 1; \mathcal{R}(a,b) = 0.8; \mathcal{R}(b,b) = 1; \mathcal{R}(b,a) = 0.8\}$,
$\Longrightarrow \Lambda = \{0.8; \quad 1\}$.

- Given a proximity relation $\mathcal{R}$ on a set U, a λ-cut of $\mathcal{R}$

$$\mathcal{R}_\lambda = \{\langle x, y\rangle \mid \mathcal{R}(x,y) \geq \lambda\}$$

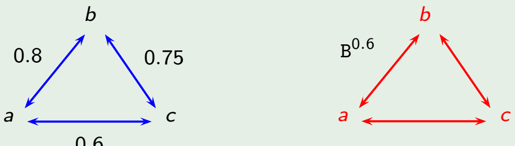
Example

$\mathcal{R}_1 = \{(a,a); (b,b)\}$ and $\mathcal{R}_{0.8} = \{(a,a); (a,b); (b,a); (b,b)\}$.

Proximity Blocks

- Proximity block of level λ (or λ-block):
  - Given a proximity relation $\mathcal{R}$ on a set U,
  - is a subset of U such that the restriction of $\mathcal{R}_\lambda$ to this subset is a maximal total relation.

Example (11)



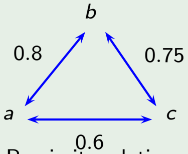
Proximity relation

$\mathrm{B}^{0.6} = \{a, b, c\}$

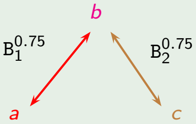
Proximity Blocks

- Proximity block of level λ (or λ-block):
  - Given a proximity relation $\mathcal{R}$ on a set U,
  - is a subset of U such that the restriction of $\mathcal{R}_\lambda$ to this subset is a maximal total relation.

Example (11)



Proximity relation

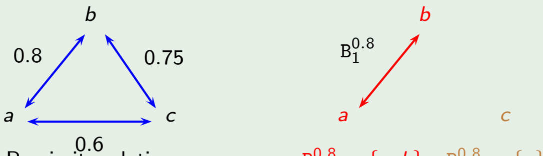


$B_1^{0.75} = \{a, b\}$ $B_2^{0.75} = \{b, c\}$

Proximity Blocks

- Proximity block of level λ (or λ-block):
  - Given a proximity relation $\mathcal{R}$ on a set U,
  - is a subset of U such that the restriction of $\mathcal{R}_\lambda$ to this subset is a maximal total relation.

Example (11)



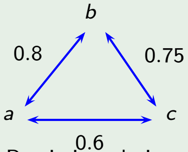
Proximity relation

$B_1^{0.8} = \{a, b\}$ $B_2^{0.8} = \{c\}$

Proximity Blocks

- Proximity block of level λ (or λ-block):
  - Given a proximity relation $\mathcal{R}$ on a set U,
  - is a subset of U such that the restriction of $\mathcal{R}_\lambda$ to this subset is a maximal total relation.

Example (11)



Proximity relation

$B_1^1 = \{a\}$ $B_2^1 = \{b\}$ $B_3^1 = \{c\}$

Proximity Classes

- Proximity class of level λ (λ-Class) of an element $x \in U$:

  $\mathcal{K}_\lambda(x) = \{y \in U \mid \mathcal{R}(x, y) \geq \lambda\}$

  The set of those elements of U that are λ-approximate to x.

Example

Given $\mathcal{R} = \{\mathcal{R}(a, b) = 0.8, \mathcal{R}(b, c) = 0.75, \mathcal{R}(a, c) = 0.6\}$

- $\mathcal{K}_{0.75}(a) = \{a, b\}$; $\mathcal{K}_{0.75}(b) = \{a, b, c\}$; $\mathcal{K}_{0.75}(c) = \{b, c\}$;
- $\mathcal{K}_{0.8}(a) = \mathcal{K}_{0.8}(b) = \{a, b\}$; $\mathcal{K}_{0.8}(c) = \{c\}$

- Blocks and Classes of a proximity relation on a set U form coverings of U, but not necessarily partitions.

Proximity Relations on Syntactic Domains

- Proximity relations can be defined on the alphabet of a first order language and extended to terms and atomic formulas.
- As was seen, for similarity relations the extension is made by a simple structural induction.
- For proximity relations this task is more complex: The key factor is to investigate the role of the notion of "indistinguishable" symbols.

Proximity Relations on Syntactic Domains

- There are **two options** because a symbol may be indistinguishable w.r.t. another:
  1. They belong to **the same proximity class** (of level λ) or
  2. They belong to **the same proximity block** (of level λ).
- The aforementioned problems arise because we were using the first option to decide if two expressions are approximate.
- We can define a new notion of proximity between expressions through the concept of λ-block.

Proximity Between Expressions

- Declarative notion of proximity: two expressions e_1 and e_2 of a first-order language $\mathcal{L}$ are λ-approximate

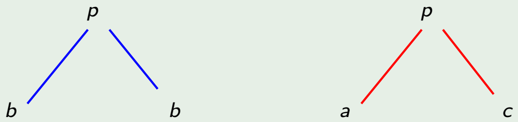
  1. When their symbols, at their corresponding positions, belong to the same λ-block and
  2. A certain symbol is always assigned to the same λ-block (i.e., it is playing the same role) along a computation.

- When two expressions e_1 and e_2 are λ-approximate, we denote this as $e_1 \approx_{\mathcal{R},\lambda} e_2$ and its *proximity degree* as $\widehat{\mathcal{R}}(e_1, e_2)$.

Proximity Between Expressions

Example (13: Proximity between $A_1 \equiv p(b,b)$ and $A_2 \equiv p(a,c)$)

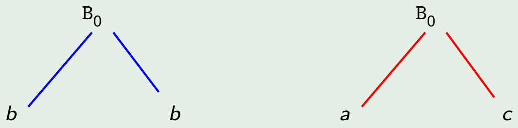
- Assume that $\mathcal{R} = \{\mathcal{R}(a,b) = 0.8, \mathcal{R}(b,c) = 0.75\}$,
- 0.75-blocks: $\mathtt{B}_0 = \{p\}$, $\mathtt{B}_1 = \{a,b\}$, $\mathtt{B}_2 = \{b,c\}$



Proximity Between Expressions

Example (13: Proximity between $A_1 \equiv p(b, b)$ and $A_2 \equiv p(a, c)$)

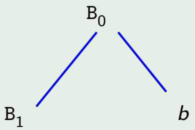
- Assume that $\mathcal{R} = \{\mathcal{R}(a, b) = 0.8, \mathcal{R}(b, c) = 0.75\}$,
- 0.75-blocks: $\mathtt{B}_0 = \{p\}$, $\mathtt{B}_1 = \{a, b\}$, $\mathtt{B}_2 = \{b, c\}$



Proximity Between Expressions

Example (13: Proximity between $A_1 \equiv p(b, b)$ and $A_2 \equiv p(a, c)$)

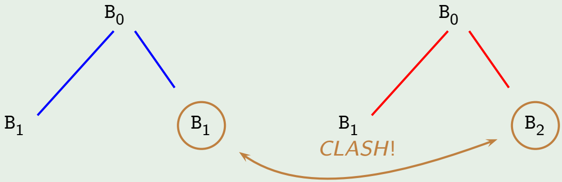
- Assume that $\mathcal{R} = \{\mathcal{R}(a, b) = 0.8, \mathcal{R}(b, c) = 0.75\}$,
- 0.75-blocks: $\mathtt{B}_0 = \{p\}$, $\mathtt{B}_1 = \{a, b\}$, $\mathtt{B}_2 = \{b, c\}$



Proximity Between Expressions

Example (13: Proximity between $A_1 \equiv p(b,b)$ and $A_2 \equiv p(a,c)$)

- Assume that $\mathcal{R} = \{\mathcal{R}(a,b) = 0.8, \mathcal{R}(b,c) = 0.75\}$,
- 0.75-blocks: $\mathtt{B}_0 = \{p\}$, $\mathtt{B}_1 = \{a,b\}$, $\mathtt{B}_2 = \{b,c\}$



- **The atoms A_1 and A_2 are not approximate.**

An Efficient Proximity-based Unification Algorithm

- Now, we are ready to define our weak unification algorithm.
- It relies on the notion of proximity just introduced.
- **The weak unification algorithm has three stages**.

An Efficient Proximity-based Unification Algorithm

- **Stage 1**: we analyze the proximity relation $\mathcal{R}$ extracting the set of proximity blocks.
  - This analysis is linked with the problem of finding all maximal cliques on an undirected graph G corresponding to $\mathcal{R}$.
  - The Bron-Kerbosch algorithm is a widely used efficient algorithm for this purpose. So, we adapt a variant of this algorithm with pivoting.
  - **Done at compile time !!**

An Efficient Proximity-based Unification Algorithm

- **Stage 2**: we extend the proximity relation $\mathcal{R}$ into a new relation $\mathcal{RB}$, enhancing $\mathcal{R}$ with specific λ-block information.

Example

If a and b belong to the λ-block $\texttt{B}$ and $\mathcal{R}(a, b) = \alpha$, we generate $\mathcal{RB}(a, b, \texttt{B}) = \alpha$.

- **Also done at compile time !!**

- These two previous steps are implemented by a foreign predicate coded in C (and connected to the system through the SWI-Prolog Foreign Language Interface).

An Efficient Proximity-based Unification Algorithm

- **Stage 3**: weak unification, formalized by a transition system (A notion of unification state + a proximity-based unification relation "$\Rightarrow$").
- A weak unification state is a tuple $\langle P, S, C, \alpha \rangle$ where:
  1. P is a (multi-)set of weak unification problems or failure;
  2. S is a set of equations in solved form;
  3. C is a set of block constraints of level λ: (<`symbol`>:< λ-`block_label`>);
  4. α is a unification degree.

An Efficient Proximity-based Unification Algorithm

- A block constraint is an ordered pair that links a symbol with a proximity λ-block label. We denote these constraints as bindings "< symbol>:< λ-block_label>" .
- Block constraints of level λ are used to detect inconsistencies in "block assignments" for an alphabet symbol.
- A **satisfaction function**, $\mathcal{S}at$, is used for block constraint satisfaction.
  - Implement as a Prolog predicate, `sat/3`, which essentially performs a membership test on an association list **and can be done efficiently at runtime!!**

An Efficient Proximity-based Unification Algorithm

- A weak unification process is formalized as a sequence of transition steps performed using "$\Rightarrow$".
- The proximity-based unification relation, "$\Rightarrow$", is defined by a set of transition rules:

  Term decomposition:

  (a) $\langle\{f(\overline{t_n})\approx f(\overline{s_n})\}\cup E, S, C, \alpha\rangle \Rightarrow \langle\{\overline{t_n \approx s_n}\}\cup E, S, C, \alpha\rangle$,

  (b) $\langle\{f(\overline{t_n})\approx g(\overline{s_n})\}\cup E, S, C, \alpha\rangle \Rightarrow$
  $\langle\{\overline{t_n \approx s_n}\}\cup E, S, \{(f{:}\mathrm{B}^{\lambda}_{\mathcal{R}}), (g{:}\mathrm{B}^{\lambda}_{\mathcal{R}})\}\cup C, \alpha\triangle\beta\rangle$,
  if $\mathcal{RB}(f, g, \mathrm{B}^{\lambda}_{\mathcal{R}}) = \beta \geq \lambda$ and $\mathcal{S}at(\{(f{:}\mathrm{B}^{\lambda}_{\mathcal{R}}), (g{:}\mathrm{B}^{\lambda}_{\mathcal{R}})\}, C) \neq \mathit{failure}$

  where $\mathcal{RB}$ is the extension of $\mathcal{R}$ with block information.

An Efficient Proximity-based Unification Algorithm

- A weak unification process is formalized as a sequence of transition steps performed using "$\Rightarrow$".
- The proximity-based unification relation, "$\Rightarrow$", is defined by a set of transition rules:

  Failure rule:

  $$\langle\{f(\overline{t_n}) \approx g(\overline{s_m})\} \cup E, S, C, \alpha\rangle \Rightarrow \langle fail, S, C, \alpha\rangle,$$

  if $n \neq m$, $\mathcal{RB}(f, g, \mathrm{B}^{\lambda}_{\mathcal{R}}) < \lambda$ or $\mathcal{S}at(\{(f{:}\mathrm{B}^{\lambda}_{\mathcal{R}}), (g{:}\mathrm{B}^{\lambda}_{\mathcal{R}})\}, C) = failure$

  where $\mathcal{RB}$ is the extension of $\mathcal{R}$ with block information.

The Proximity-based Unification Algorithm in Action

Example (15: $A_1 \equiv p(b, b)$ and $A_2 \equiv p(a, c)$)

- $\mathcal{R}(a, b)=0.8, \mathcal{R}(b, c)=0.75$
- **Stage 1**: $\mathtt{B}_1=\{a, b\}, \mathtt{B}_2=\{b, c\}$
- **Stage 2**: $\mathcal{RB}(a, b, \mathtt{B}_1) = 0.8, \mathcal{RB}(b, c, \mathtt{B}_2) = 0.75, \ldots$
- **Stage 3**: **The atoms A_1 and A_2 do not weakly unify**.

$$\langle \{\underline{p(b, b) \approx p(a, c)}\}, id, \emptyset, 1\rangle$$
$$\Rightarrow_{1a} \langle \{\underline{b \approx a}, b \approx c\}, id, \emptyset, 1\rangle$$
$$\Rightarrow_{1b} \langle \{\underline{b \approx c}\}, id, \{(b{:}\mathtt{B}_1), (a{:}\mathtt{B}_1)\}, 0.8 \wedge 1\rangle$$
$$\Rightarrow_{5} \langle \mathit{failure}, id, \{(b{:}\mathtt{B}_2), (c{:}\mathtt{B}_2), (b{:}\mathtt{B}_1), (a{:}\mathtt{B}_1)\}, 0.8\rangle$$

It is important to note that Sessa's weak unification algorithm wrongly succeeds in this example!!

Three Different Weak Unification Algorithms

- The BPL system implements three different weak unification algorithms:
  - **(A1)** The similarity-based unification algorithm proposed by Maria Sessa, which is only adequate for similarity relations (“`:- weak_unification(a1).`”).
  - **(A2)** The original proximity-based unification algorithm that was defined in our 2015 FSS paper, which uses *proximity constraints* (“`:- weak_unification(a2).`”).
  - **(A3)** The present reformulation of the proximity-based unification algorithm described in this paper, which uses *block constraints* (“`:- weak_unification(a3).`”).

Weak SLD Resolution (WSLD) (of level λ)

- Let Π be a program, $\mathcal{R}$ be a proximity relation, $\triangle$ a fixed t-norm and a λ cut value.
- Weak SLD (WSLD) resolution is defined as a transition system $\langle E, \Rightarrow_{\text{WSLD}} \rangle$ where:
  - E is a set of tuples $\langle \mathcal{G}, \theta, \alpha, C \rangle$ (the state of a computation)
  - $\Rightarrow_{\text{WSLD}} \subseteq (E \times E)$ is the transition relation, defined as:

$$\langle (\leftarrow A' \wedge \mathcal{Q}'), \theta, \alpha, C \rangle \Rightarrow_{\text{WSLD}} \langle \leftarrow (\mathcal{Q} \wedge \mathcal{Q}')\sigma, \theta\sigma, \beta \triangle \alpha \triangle \mu, C' \cup C \rangle$$

if
1. $R \equiv (A \leftarrow \mathcal{Q} \text{ with } \mu) \ll \Pi$,
2. $\text{wmgu}^{\lambda}_{\mathcal{R}}(A, A') = \langle \sigma, C', \beta \rangle$,
3. $\mathcal{S}at(C', C) \neq \mathit{failure}$,
4. $(\beta \triangle \alpha \triangle \mu) \geq \lambda$.

Where β and μ are truth degrees (in $[0, 1]$), $\mathcal{Q}$ and $\mathcal{Q}'$ are conjunctions of atoms.

Weak SLD Resolution (WSLD) (of level λ)

- A **WSLD derivation** (of level λ) for $\Pi \cup \{\mathcal{G}_0\}$ is a sequence of WSLD resolution steps

$$\langle \mathcal{G}_0, id, 1, \emptyset \rangle \Rightarrow_{\text{WSLD}} \langle \mathcal{G}_1, \theta_1, \alpha_1, C_1 \rangle \Rightarrow_{\text{WSLD}} \ldots \Rightarrow_{\text{WSLD}} \langle \mathcal{G}_n, \theta_n, \alpha_n, C_n \rangle$$

- **WSLD refutation** is a WSLD derivation (of level λ):

$$\langle \mathcal{G}, id, 1, \emptyset \rangle \Rightarrow_{\text{WSLD}}{}^{*} \langle \square, \theta, \alpha, C \rangle$$

- **Output of the computation**: $\langle \sigma, \alpha \rangle$
  - $\sigma = \theta \restriction \mathcal{V}ar(\mathcal{G}_0)$ is a computed answer and α is its computed approximation degree.
- Block constraints are used to guarantee the consistency of the final answer (although it is not part of it).

WSLD Resolution: Implementation details

- Bousi∼Prolog implements WSLD resolution by compiling (transpiling) BPL programs into a set of Prolog clauses that are able of emulating it.
- It uses a program translation that we call **BPL expansion**:
  1. Each BPL program rule is replaced by the set of rules which are approximate (w.r.t. $\mathcal{R}$) to the rule being transformed.
  2. The head of those approximate rules are linearised to facilitate the crisp unification of the defined predicate with a goal, while the weak unification of their arguments are carried out explicitly in the body of the transformed rules

WSLD Resolution: Implementation details

Definition (BPL expansion)

- Let $\mathcal{RB}$ be the extension of $\mathcal{R}$, $\triangle$ the fixed t-norm and $\lambda \in [0, 1]$ a cut value.
- Let $p(t1, \ldots, t_n) \leftarrow \mathcal{Q}$ with δ be a graded rule in Π.

Then, for each entry $\mathcal{RB}(p, q, \mathtt{B}^{\lambda}_{\mathcal{R}}) = \alpha \geq \lambda$ add to the transformed program Π' the e-clause:

$$\langle q(x_1, \ldots, x_n) \leftarrow x_1 \approx t_1 \wedge \cdots \wedge x_n \approx t_n \wedge \mathcal{Q};\ (\delta \triangle \alpha);\ [p : \mathtt{B}^{\lambda}_{\mathcal{R}}, q : \mathtt{B}^{\lambda}_{\mathcal{R}}] \rangle$$

where each x_i is a fresh variable and $x_i \approx t_i$ forces weak unification, i.e., the evaluation of $\text{wmgu}^{\lambda}_{\mathcal{R}}(x_i, t_i)$.

WSLD Resolution: Implementation details

Example (17)

% PROXIMITY EQUATIONS
p ∼ q = 0.9.
% FACTS & RULES
p(a).

% PROXIMITY RELATION
$\mathcal{RB}$(p,q,0) = 0.9.
$\mathcal{RB}$(q,p,0) = 0.9.
% E-CLAUSES
<p(X1) :- X1≈a; 1 ; []>
<q(X1) :- X1≈a; 0.9; [(p,0), (q,0)]>

WSLD Resolution: Implementation details

Example (18)

% PROXIMITY EQUATIONS
a ∼ b = 0.7.
b ∼ c = 0.8.
p ∼ q = 0.9.
% FACTS & RULES
p(X) :- r(X) with 0.75.
r(a).

% PROXIMITY RELATION
$\mathcal{RB}$(a,b,2)=0.7. $\mathcal{RB}$(c,b,1)=0.8.
$\mathcal{RB}$(b,a,2)=0.7. $\mathcal{RB}$(p,q,0)=0.9.
$\mathcal{RB}$(b,c,1)=0.8. $\mathcal{RB}$(q,p,0)=0.9.
% E-CLAUSES
<p(X1) :- X1≈X, r(X); 0.75 ; []>
<q(X1) :- X1≈X, r(X);
0.9 ∧ 0.75; [(p,0), (q,0)]>
<r(X1) :- X1≈a; 1; []>

WSLD Resolution: Implementation details

Definition (operational semantics for expanded programs)

Defined as a transition system $\langle E, \Rightarrow_{\text{EXP}} \rangle$ where

- E is a set of tuples $\langle \mathcal{G}, \alpha, C \mid \theta \rangle$ (goal, approximation degree, block constraints, substitution),
- $\Rightarrow_{\text{EXP}} \subseteq (E \times E)$ is a transition relation which satisfies:

Rule 1: if $\text{wmgu}^{\lambda}_{\mathcal{R}}(A, B) = \langle \sigma, \beta, C' \rangle$, $\mathcal{S}at(C \cup C') \neq \mathit{failure}$ and $(\beta \triangle \alpha) \geq \lambda$,

$$\langle (\leftarrow \underline{A \approx B} \wedge \mathcal{Q}), \alpha, C \mid \theta \rangle \; \Rightarrow_{\text{EXP}} \; \langle \leftarrow \mathcal{Q}\sigma, \beta \triangle \alpha, C \cup C' \mid \theta\sigma \rangle$$

WSLD Resolution: Implementation details

Definition (operational semantics for expanded programs)

Defined as a transition system $\langle E, \Rightarrow_{\text{EXP}} \rangle$ where

- E is a set of tuples $\langle \mathcal{G}, \alpha, C \mid \theta \rangle$ (goal, approximation degree, block constraints, substitution),
- $\Rightarrow_{\text{EXP}} \subseteq (E \times E)$ is a transition relation which satisfies:

Rule 2: if $\langle p(x_1, \ldots, x_n) \leftarrow x_1 \approx t_1 \wedge \cdots \wedge x_n \approx t_n \wedge \mathcal{Q}'; \beta; C' \rangle \ll \Pi'$ and $\mathcal{S}at(C \cup C') \neq failure$

$$\langle (\leftarrow \underline{p(s_1, \ldots, s_n)} \wedge \mathcal{Q}), \alpha, C \mid \theta \rangle \;\Rightarrow_{\text{EXP}} \\ \langle (\leftarrow s_1 \approx t_1 \wedge \cdots \wedge s_n \approx t_n \wedge \mathcal{Q}' \wedge \mathcal{Q}), \beta \triangle \alpha, C \cup C' \mid \theta \rangle$$

in Rule 2, we perform a syntactic unification of the selected atom of the e-goal and the head of the e-clause.

WSLD Resolution: Implementation details

Example (20: e-clauses for the program of Ex.18)

```
p(X1,C0,C2,D):- unify_arguments_a3([[X1,X,C0,C1,D1]]),
                r(X,C1,C2,D2),
                degree_composition([0.75,D1,D2],D),
                over_lambdacut(D).
q(X1,C0,C3,D):- over_lambdacut(0.9),
                sat_a3([q:0,p:0],C0,C1),
                unify_arguments_a3([[X1,X,C1,C2,D1]]),
                r(X,C2,C3,D2),
                degree_composition([0.9,0.75,D1,D2],D),
                over_lambdacut(D).
r(X1,C0,C1,D):-  unify_arguments_a3([[X1,a,C0,C1,D1]]),
                degree_composition([1,D1],D),
                over_lambdacut(D).
```

Outline

1 Introduction
- Fuzzy Logic Programming
- Bousi∼Prolog general features

2 Bousi∼Prolog Fundamentals and its Implementation
- Proximity Relations and Similarity Relations
- The Similarity-based Unification Algorithm
- Pros and Cons of Proximity Relations
- Proximity Blocks vs. Proximity Classes
- A New Notion of Proximity Between Expressions
- An Efficient Proximity-based Unification Algorithm
- Weak SLD Resolution

3 Some Bousi∼Prolog Applications
- Pattern Matching in Strings
- Flexible Query Answering in Deductive Databases
- Information Retrieval
- Approximate Reasoning

4 Conclusions

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[\boxed{\text{a,b}}, c, a, c, b, d, a, c, d, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, \boxed{\text{b,c}}, a, c, b, d, a, c, d, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, \boxed{\text{c,a}}, c, b, d, a, c, d, b, ...]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, \boxed{a,c}, b, d, a, c, d, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, a, \boxed{\mathrm{c,b}}, d, a, c, d, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, a, c, \boxed{\text{b,d}}, a, c, d, b, ...]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, a, c, b, \boxed{\text{d,a}}, c, d, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, a, c, b, d, \boxed{\text{a,c}}, d, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, a, c, b, d, a, \boxed{\text{c,d}}, b, \ldots]$$

Pattern Matching in Strings

- Program Pattern Matching in Strings:
  - Given a list of characters, find the occurrences of a pattern `[e1,e2]`, where `e1` must be `a` and `e2` may be `b` or `c`.
  - The program search the list exploring if each pair of characters match the pattern:

$$[a, b, c, a, c, b, d, a, c, \boxed{\text{d,b}}, \ldots]$$

Pattern Matching in Strings

```
% PROXIMITY EQUATIONS
b~c = 1.

% FACTS and RULES
match(_, [], 0).
match(P, S, N) :- search(P, S, N, P, S, 0).

% search(Pattern, String, Number, Pattern_acc, String_acc, Number_acc):
search(P, [], N, _, _, A) :- P = [] -> N is A+1 ; N= A.
search([], [_ | _], N, OP, OS, A) :- A1 is A+1, search_next(N, OP, OS, A1).
search([P | PP], [P | SS], N, OP, OS, A) :- !, search(PP, SS, N, OP, OS, A).
search([_ | _], [_ | _], N, OP, OS, A) :- search_next(N, OP, OS, A).
```

Pattern Matching in Strings

```
% FACTS and RULES (Cont.)

% search_next(Number, Pattern_acc, String_acc, Number_acc).
% Called after the pattern is found or the pattern fail to be found.
% If String_acc = [_ | SS] the search of the pattern continues
% starting from SS.
%
search_next(N, OP, [_ | SS], A) :- search(OP, SS, N, OP, SS, A).


 ?- goal(N):-match([a,b], [a,b,c,a,c,b,d,a,c,d,b,b,a,b,c,c,a,c,a,b], N).
N = 6.
```

Flexible Query Answering in Deductive Databases

- The first application examples come from the area of flexible databases.
- There are several approaches to fuzzy flexible database. We highlight two of them:
  1. The model of **Buckles-Petry and Shenoi-Melton** (similarity/proximity relations)
  2. The model of **Prade-Testemale** (fuzzy sets).
- We show how Bousi∼Prolog allows to simulate both fuzzy flexible database approaches effectively

Flexible Query Answering in Deductive Databases The model of Buckles-Petry and Shenoi-Melton

```
% DIRECTIVE
:-lambda_cut(0.5).
%% PROXIMITY EQUATIONS
%% Location Distance Relation
bervely_hills ~ downtown=0.3.
downtown ~ santa_monica=0.23.
bervely_hills ~ santa_monica=0.45.
downtown ~ westwood=0.25.
bervely_hills ~ hollywood=0.56.
hollywood ~ santa_monica=0.3.
bervely_hills ~ westwood=0.9.
```

```
hollywood ~ westwood=0.45.
downtown ~ hollywood=0.45.
santa_monica ~ westwood=0.9.

%% Category Relation
comedy ~ drama=0.6.
drama ~ adventure=0.6.
comedy ~ adventure=0.3.
drama ~ suspense=0.6.
comedy ~ suspense=0.3.
adventure ~ suspense=0.9.
```

Flexible Query Answering in Deductive Databases The model of Buckles-Petry and Shenoi-Melton

```
%% FACTS MODELING A DATABASE
%% Films Table:
%% film(Title, Director, Category)
film(four_feathers,korda,adventure).
film(modern_times,chaplin,comedy).
film(psycho, hitchcock,suspense).
film(rear_window,hitchcock,suspense).
film(robbery,yates,suspense).
film(star_wars,lucas,adventure).
film(surf_party,dexter,drama).
```

```
%% Theaters Table:
%% theater(Name,Owner,Location).
theater(chinese,mann,hollywood).
theater(egyptian,va,westwood).
theater(music_hall,lae,bervely_hills).
theater(odeon,cineplex,santa_monica).
theater(rialto,independent,downtown).
theater(village,mann,westwood).
```

Flexible Query Answering in Deductive Databases
The model of Buckles-Petry and Shenoi-Melton

```
%% FACTS MODELING A DATABASE
%% Engagements Table:
%% engagement(Film,Theater)
engagement(modern_times, rialto).
engagement(start_wars, rialto).
engagement(star_wars, chinese).
engagement(rear_window, egyptian).
engagement(surf_party, village).
engagement(robbery, odeon).
engagement(modern_times, odeon).
engagement(four_feathers,music_hall).
```

```
%% MAIN RULE:
%% search(in, in, out, out)
search(Category,Location,Film,Theater)
    :-  film(Film,_, Category),
        engagement(Film, Theater),
        theater(Theater, _, Location).
```

```
?- search(adventure, westwood, Film, Theater).
Film=four_feathers, Theater=music_hall, with 0.9;
Film=rear_window, Theater=egyptian, with 0.9;
Film=robbery, Theater=odeon, with 0.9;
Film=surf_party, Theater=village, with 0.6;
```

Flexible Query Answering in Deductive Databases The model of Prade-Testemale

```
%% DIRECTIVES declaring and defining linguistic variables
%% Linguistic variable: rental
:-domain(rental,0,600,euros).
:-fuzzy_set(rental,[cheap(100,100,250,500), normal(100,300,400,600),
                    expensive(300,450,600,600)]).

%% Linguistic variable: walk distance
:-domain(distance,0,50,minutes).
:-fuzzy_set(distance,[close(0,0,15,40), medial(15,25,30,35), far(20,35,50,50)]).

%% Linguistic variable: flat conditions
:-domain(condition,0,10,conditions)).
:-fuzzy_set(condition,[unfair(0,0,1,3), fair(1,3,6), good(4,6,8), excellent(7,9,10,10)]).
```

Flexible Query Answering in Deductive Databases The model of Prade-Testemale

```
%% FACTS
%% Flats table: flat(Code,Street,Rental,Condition).
flat(f1, libertad_street, rental#300, more_or_less#good).
flat(f2, ciruela_street, rental#450, somewhat#good).
flat(f3, granja_street, rental#200, unfair).

%% Streets table: street(Name,District)
street(libertad_street, ronda_la_mata).      street(ciruela_street, downtwon).
street(granja_street, ronda_santa_maria).

%% Distance (to campus) table: distance(District,District,Distance)
distance(downtwon,campus,medial).          distance(ronda_santa_maria,campus,far).
distance(ronda_la_mata, campus, somewhat#close).
```

Flexible Query Answering in Deductive Databases
The model of Prade-Testemale

```
%% RULES
flat_district(Flat,Flat_Dist) :- flat(Flat,Street,_,_),
                                 street(Street,Flat_Dist).

close_to(Flat, District):- flat_district(Flat, Flat_Dist),
                           distance(Flat_Dist, District, close).

select_flat(Flat,Street):- flat(Flat,Street,cheap,good), close_to(Flat,campus).

?- select_flat(Flat, Street).
Flat = f1, Street = libertad, with 0.8;    Flat = f2, Street = ciruela, with 0.14;
```

Information Retrieval

- Proximity equations can be used as a fuzzy model for information retrieval where textual information is selected or analyzed using an ontology of terms.
- Ontologies of terms can be represented by a set of proximity equations (The set of proximity equations used in this example has been obtained using **WordNet**).
- In this example, we want to extract information of terms analogous to "wheat" on a given text (borrowed from Reuters, a test collection for text categorization research).

Information Retrieval

- The text provided by Reuters:

  The U.S. Agriculture Department reported the farmer-owned reserve national five-day average price through April 8 as follows (Dlrs/Bu-Sorghum Cwt) - . . .

- The text after a linguistic preprocess (removing stop words, performing a stemming process and grouping meaningful couples of words – e.g.: crude_oil –):

  agriculture, department, report, farm, own, reserve, national, average, price, loan, release, price, reserves, matured, bean, grain, enter, corn, sorghum, rates, bean, potato

Information Retrieval

```
%% DIRECTIVES
:- transitivity(yes). %% builds a similarity starting from the proximity equations
:-transitivity(min).
:- weak_unification(a1).
:- wn_connect.
:- wn_gen_prox_equations(wup, [[wheat, agriculture, department, report, farm, own,
reserve, national, average, price, loan, release, price,
reserves, matured, bean, grain, enter, corn, sorghum,
rates, bean, potato]]).

%% FACTS and RULES
% searchTerm(T,L1,L2), true if T is a (constant) term, L1 is a list of (constant)
% terms (model a text); L2 is a list of triples t(X,N,D), where X is a
% term similar to T with degree D, which occurs N times in the text L1
searchTerm(T,[],[]).
searchTerm(T,[X|R],L):- T~X=AD,!,searchTerm(T,R,L1),insert(t(X,1,AD),L1,L).
searchTerm(T,[X|R],L):- searchTerm(T,R,L).
```

Information Retrieval

```
insert(t(T,N,D), [], [t(T,N,D)]).
insert(t(T1,N1,D), [t(T2,N2,_)|R],[t(T1,N,D)|R]) :- T1 == T2, N is N1+N2.
insert(t(T1,N1,D),[t(T2,N2,D2)|R2],[t(T2,N2,D2)|R]):-
                        T1\==T2,insert(t(T1,N1,D),R2,R).

%% GOAL
g(T,L):-searchTerm(T, [agriculture,department,report,farm,
                 own,reserve,national,average,price,loan,release,
                 price,reserves,matured,bean,grain,enter,corn,
                 sorghum,rates,bean,potato], L).
```

```
?- g(wheat,L).
L = [t(potato,1,0.43),t(bean,2,0.43),t(rates,1,0.43),t(sorghum,1,0.89),
t(corn,1,0.93),t(grain,1,0.42),t(reserves,1,0.35),t(price,2,0.375),
t(release,1,0.55),t(loan,1,0.43),t(average,1,0.35),t(national,1,0.61),
t(reserve,1,0.37),t(farm,1,0.44),t(report,1,0.37),t(department,1,0.35),
t(agriculture,1,0.35)]
```

Approximate Reasoning

- **Approximate reasoning** is basically the inference of an imprecise conclusion from imprecise premises.
- **Fuzzy inference** is a generalization of *modus ponens*. It can be stated as:

$$\frac{\begin{array}{l}\text{if } x \text{ is } F \text{ then } y \text{ is } G\\ x \text{ is } F'\end{array}}{y \text{ is } G'}$$

  - x and y in crisp sets U and W,
  - F and F' are fuzzy subsets on U,
  - G and G' are fuzzy subsets on W.

- Roughly speaking, and following Zadeh, $G' = F' \circ R$ where R is a fuzzy relation (the meaning of the conditional) such that $R(x, y) = min(\mu_F(x), \mu_G(y))$, for all $x \in U$ and $y \in W$.

Approximate Reasoning

- Bousi∼Prolog proceeds differently by constructing (at compile time) a fuzzy relation over the (declared) fuzzy domains, which is used by the weak SLD resolution procedure to infer an answer to a query.

```
:-domain(age,0,100,years).
:-fuzzy_set(age,[young(0,0,30,50), middle(20,40,60,80), old(50,80,100,100)]).

:-domain(speed,0,40,'km/h')).
:-fuzzy_set(speed,[slow(0,0,15,20), normal(15,20,25,40), fast(25,30,40,40)]).

speed(X, fast) :- age(X, young).    age(robert, middle).

?- speed(robert, somewhat#fast).           Yes with 0.375
```

- Last program models the fuzzy inference: "**if** x **is young then** x **is fast**" and "**Robert is middle**" therefore "**Robert is somewhat fast**" in a very natural way.

Real applications

- We have developed several real applications coded with Bousi∼Prolog:
  - Text categorization and cataloging [RJFG13JLRE] and [AJRS22]
    `https://dectau.uclm.es/bousi-prolog/applications/`
  - Abstract knowledge discovery [RJ15JIFS]
  - Linguistic feedback in computer games [RT16]
  - FuzzyDES: mapping Bousi∼Prolog to a deductive database. Application to a recommender system
    `http://des.sourceforge.net/fuzzy/recommender.dl`
  - Integration of WordNet into Bousi∼Prolog [JS19EUSFLAT] and [JS21TPLP]: The idea is to provide Bousi∼Prolog with linguistic resources
    `https://dectau.uclm.es/bousi-prolog/applications/`

Outline

1. Introduction
   - Fuzzy Logic Programming
   - Bousi∼Prolog general features
2. Bousi∼Prolog Fundamentals and its Implementation
   - Proximity Relations and Similarity Relations
   - The Similarity-based Unification Algorithm
   - Pros and Cons of Proximity Relations
   - Proximity Blocks vs. Proximity Classes
   - A New Notion of Proximity Between Expressions
   - An Efficient Proximity-based Unification Algorithm
   - Weak SLD Resolution
3. Some Bousi∼Prolog Applications
   - Pattern Matching in Strings
   - Flexible Query Answering in Deductive Databases
   - Information Retrieval
   - Approximate Reasoning
4. **Conclusions**

Conclusions

- Throughout this talk we have presented part of the work developed over almost two decades.
- Motivated by the objective to introduce weak unification within (fuzzy) logic languages,
  - We have designed Bousi∼Prolog: a Prolog programming language extension; and
  - Developed the BPL system: a **high level implementation** of Bousi∼Prolog.

    BPL programs are “compiled” into SWI-Prolog programs

Conclusions

- We have presented the main features and some implementation details of Bousi∼Prolog.

- Through a number of (small but meaningful) examples we have shown the potential power of Bousi∼Prolog and how it is useful for:
  - Pattern matching in strings;
  - Flexible query answering;
  - Dealing with approximate reasoning; and
  - Modeling vagueness.

Future Work

- We need to delve deeper into the formal properties of Bousi∼Prolog.
  - A new declarative semantics for programs in the context of proximity relations and arbitrary t-norms.
  - Soundness and completness properties in that context.
  - A study of (fuzzy) negation in Bousi∼Prolog.

Acknowledgments

- I would like to thank the efforts of the people who have contributed to the development of the BPL system in the past:
  - Clemente Rubio-Manzano (University of Castilla-La Mancha – now at the Bío-Bío University -Chile-)
  - Juan Gallardo-Casero (University of Castilla-La Mancha – now at INDRA Sistemas).
- Special thanks to Fernando Sáenz-Pérez, from Complutense University of Madrid, for its involvement during the last years that boosted the BPL system to another level.

Bibliography: Introduction

[Baldwin et al-84] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. "Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence". John Wiley & Sons, Inc., 1995.

[Fontana & Formato-99] F. A. Fontana and F. Formato. "Likelog: A logic programming language for flexible data retrieval". In Proc. of the ACM Symposium on Applied Computing (SAC'99), pages 260–267, 1999.

[Julián et al-08] Pascual Julián-Iranzo and Clemente Rubio-Manzano and J. Gallardo-Casero. "Bousi∼Prolog: a Prolog extension language for flexible query answering". VIII Jornadas sobre Programación y Lenguajes, PROLE'2008, Gijón, España, October, 7-10, Páginas: 41-55.

[Julián & Sáenz-23] Pascual Julián Iranzo, Fernando Sáenz-Pérez: "Bousi∼Prolog: Design and implementation of a proximity-based fuzzy logic programming language". Expert Syst. Appl. 213(Part): 118858 (2023)

[Lee-72] R.C.T. Lee, "Fuzzy Logic and the Resolution Principle", J. of the ACM, 19(1): 109-119, 1972.

[S. Muñoz et al-04] S. Guadarrama, S. Mu noz, and C. Vaucheret. "Fuzzy Prolog: A new approach using soft constraints propagation". Fuzzy Sets and Systems, Elsevier, 144(1):127–150, 2004.

[Ojeda et al-01] J. Medina and M. Ojeda-Aciego and P. Vojtáš, "A procedural semantics for multi-adjoint logic programming". Proc. of Progress in Artificial Intelligence, EPIA'01, Springer-Verlag, LNAI 2258, pages 290–297, 2001.

[Ojeda et al-04] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. "Similarity-based unification: a multi-adjoint approach". Fuzzy Sets and Systems, 146(1):43–62, 2004.

[M. Sessa-01] V. Loia, S. Senatore, and M. I. Sessa. "Similarity-based SLD resolution and its implementation in an extended prolog system". In FUZZ-IEEE, pages 650–653, 2001.

[Vojtas & Paulík-96] P. Vojtáš and L. Paulík, "Soundness and completeness of non-classical extended SLD-resolution". Proc. ELP'96 Leipzig, LNCS 1050, Springer Verlag, pages 289-301, 1996.

[Vojtas-01] P. Vojtáš. "Fuzzy Logic Programming". Fuzzy Sets and Systems, 124(1):361–370, 2001.

Bibliography: Real applications

[AJRS22] Al-Sayadi, Sami H. and Julian-Iranzo, Pascual and Romero, Francisco P. and Sáenz-Pérez, Fernando, "A Fuzzy Declarative Approach to Classify Unlabeled Short Texts Based on Automatically Constructed WordNet Ontologies", Computational Intelligence and Mathematics for Tackling Complex Problems 3, Springer International Publishing, pages 157–164, 2022. https://doi.org/10.1007/978-3-030-74970-5_18

[JS18IEEE-TFS] Pascual Julián Iranzo, Fernando Sáenz-Pérez: "A Fuzzy Datalog Deductive Database System". IEEE Trans. Fuzzy Syst. 26(5): 2634-2648 (2018)

[JS19EUSFLAT] Pascual Julián Iranzo, Fernando Sáenz-Pérez: "WordNet and Prolog: why not?". Proceedings of the 11th Conference of the European Society for Fuzzy Logic and Technology, EUSFLAT 2019, pages 1–8, 2019.

[JS21TPLP] Pascual Julián Iranzo, Fernando Sáenz-Pérez: "Implementing WordNet Measures of Lexical Semantic Similarity in a Fuzzy Logic Programming System". Theory Pract. Log. Program. 21(2): 264-282, 2021.

[RJ15JIFS] Rubio-Manzano, C., Julián-Iranzo, P. "Incorporation of abstraction capability in a logic-based framework by using proximity relations". Journal of Intelligent and Fuzzy Systems 29 (4), 1671–1683, 2015. http://dx.doi.org/10.3233/IFS-151645

[RJFG13JLRE] Romero, F. P., Julián-Iranzo, P., Soto, A., Ferreira, M., Gallardo-Casero, J. "Classifying unlabeled short texts using a fuzzy declarative approach". Language Resources and Evaluation 47 (1), 151–178, 2013. http://dx.doi.org/10.1007/s10579-012-9203-2

[RT16] C. Rubio-Manzano and G. Triviño. "Improving player experience in Computer Games by using players' behavior analysis and linguistic descriptions", International Journal of Human-Computer Studies, vol. 95, pages 27–38, 2016.