ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/262358878

A Fuzzy linguistic prolog and its applications

Article *in* Journal of Intelligent and Fuzzy Systems · May 2014

DOI: 10.3233/IFS-130834

CITATIONS
26

READS
173

2 authors:

Clemente Rubio Manzano
Universidad de Cádiz
46 PUBLICATIONS **172** CITATIONS
SEE PROFILE

Pascual Julián Iranzo
University of Castilla-La Mancha
62 PUBLICATIONS **450** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

HVAC layout optimization through artificial intelligence algorithms View project

Fuzzy Linguistic Prolog and Fuzzy Logic Programming View project

All content following this page was uploaded by Clemente Rubio Manzano on 08 November 2016.

The user has requested enhancement of the downloaded file.

A Fuzzy linguistic prolog and its applications

Clemente Rubio-Manzano[2] and Pascual Julián-Iranzo[1]

[1] Dep. Information Technologies and Systems, University of Castilla-La Mancha, Spain.
Pascual.Julian@uclm.es
[2] Dept. de Sistemas de Información, Universidad del Bío-Bío, Chile.
clrubio@ubiobio.cl

Abstract. In this work a fuzzy linguistic prolog language is presented and its design, implementation and applications are detailed. A fuzzy linguistic prolog is a fuzzy prolog which allows to work with both fuzzy linguistic and linguistic tools in order to gear the prolog systems towards the computing with words paradigm in which the linguistic resources can be very useful.

Keyword: Fuzzy Logic Programming, Fuzzy Linguistic Prolog, Fuzzy Unification, Semantic Unification, Weak SLD Resolution

1 Introduction

The Prolog programming language has been used widely in the area of Artificial Intelligence for knowledge representation, expert system development or natural language processing. However, classical Prolog is not capable of representing and handling in an explicit, natural way the vagueness and/or the imprecision existing in the real world (as it will be shown in a number of examples along this paper). Certainly, dealing with vagueness and/or imprecision is essential in most Artificial Intelligence application areas, such as expert systems, fuzzy control, robotics, computer vision, machine learning or information retrieval. So, it is crucial to enhance these languages with new facilities.

Since the seminal work of Lee [21], in the early seventies of the last century, most of the proposed and implemented fuzzy prolog systems modified the classical resolution procedure, replacing it by a fuzzy one, while the unification mechanism was left intact [4, 11, 13, 14, 22]. However, there exists an alternative line of work which has produced fuzzy prolog systems that extends the classical SLD resolution principle by only changing its classical unification algorithm and substituting it by a fuzzy one, while the other features of the SLD resolution principle remain untouched [10, 18, 23].

One distinguished feature of the first line of work is that it has produced programming languages with explicitly annotated rules. Informally, these annotations or weights are expressing the confidence which the user of the system has in the truth of the rules integrating a program. This way of expressing vagueness and/or imprecision may be inappropriate, or at least inconvenient, in some contexts (where knowledge is characterized by means of linguistic concepts). Also it seems unnatural, from a declarative point of view, to specify vagueness and/or imprecision by means of a precise number. If we want to be more accurate to the spirit of declarative programming, we should find

mechanisms that allow us to express the vagueness in a way independent from the program. That is, surpass the well known assertion stating that an "Algorithm = Logic + Control" [20] and to move to a new assertion: an "Algorithm = Logic + Vague Knowledge + Control". In consonance with this argumentation, recently in a line of work proposing a 'natural' fuzzy linguistic Prolog [35], it has been suggested that a 'natural' fuzzy linguistic Prolog system should be able to work with linguistic resources (i.e. electronic linguistic tools as WordNet or ConceptNet [24]) and the resolution rule and unification mechanism should be modified in order to treat with linguistic relations (such as synonymy or antonym).

For these reasons, if our objective is to design a true fuzzy linguistic prolog, the second line of work seems to be more appropriate because: (i) it leads to languages that allow a clean separation between Logic, Vague Knowledge and Control; (ii) thanks to its fuzzy unification mechanism, it is easy to work with linguistic concepts and even to incorporate both fuzzy linguistic and linguistic tools.

Bousi∼Prolog (BPL for short) [16–18] is a fuzzy logic programming language, that may be classified among those programming languages pertaining to the second approach mentioned above. Moreover, it can be seen as a fuzzy linguistic logic system since its design has been conceived to make a clear separation between Logic, Vague Knowledge and Control. Bousi∼Prolog allows that a linguistic term may be used as a regular symbol of a first order alphabet, that is, as a constant, a function, or even a predicate symbol. Also it is able to work with electronic linguistic tools in a natural way since it allows to represent and manipulate linguistic relationships as the synonymy. For example, Bousi∼Prolog allows to work with WordNet::Similarity [28]. WordNet::Similarity [3] is a freely available software package that offers an implementation of six measures of semantic resemblance and three measures of relatedness between pairs of concepts (or word senses), all of which are based on the WordNet lexical database [9]. Wordnet is a thesaurus but is also an ontology. It groups English words into sets of synonyms called synsets. It also provides short, general definitions, and records the various semantic relations between these synonym sets. Thus, it is possible to know the meaning of a word and, at the same time, to associate it with other words using ontological relations like synonymy, antonymy, etc. The semantic relationships and the synsets can be used to obtain a degree of closeness between two words and thus, the set of words related to another word through the WordNet::Similarity API. The following example shows this feature.

Assume a fragment of a deductive database that stores information about people and their preferences.

```
%% mary loves chess
loves(mary, tracking).

%% john likes football
likes(john,football).

%% peter plays basketball
plays(peter,basketball).
```

[3] http://wn-similarity.sourceforge.net

```
%% if a person practises sports, he/she is
%% healthy
healthy(X):- practises(X,sport).
```

In a standard Prolog system, if we ask about what persons are healthy "`?-healthy(X).`" the system fails. However "`mary`", "`john`" and "`peter`" also are reasonable candidates to be a healthy person. Hence, if we are looking for a flexible query answering procedure, more accurate to the real world behavior, "`mary`", "`john`" and "`peter`" should be appear as answers. As Bousi∼Prolog allows to work with linguistic dictionaries, a list of similar concepts for the source concepts "`practises`" and "`sport`" could be obtained by using WordNet::Similarity. This list can be represented in Bousi∼Prolog by means of a set of proximity equations (see Section 2.4).

```
practises~loves=0.9.  sport~basketball=1.0.
practises~likes=0.7.  sport~football=1.0.
practises~plays=1.0.  sport~tracking =0.8.
```

Now, the BPL system allows us to get the answers: "`X=mary with 0.8 `", "`X=john with 0.7 `", and "`X=peter with 1.0 `". To obtain the first answer, the BPL system operates as follows: since we have specified that "`sport`" is close to "`tracking`", with degree 0.8, and that "`practises`" is close to "`loves`" with 0.9. The terms "`practises(X,sport)`" and "`loves(mary, tracking)`" may "weakly" unify with approximation degree `0.8` $\wedge$ `0.9 = 0.8`, producing the binding `X=mary`; therefore, the assertion `healthy(mary)` is stated with approximation degree `0.8`. Similarly for the remainder answers.

The above example, at the same time that confirms that Bousi∼Prolog can work with linguistic relations (as synonymy), also serves to illustrate some aspects of the syntax and the operational semantics of this programming language.

The aim of this paper is to present the main features of Bousi∼Prolog that make it a good candidate to be considered a fuzzy linguistic prolog system. To this end, along this work we precise the BPL syntax and operational semantics as well as we clarify other important features that contribute to make this goal possible. Particularly, we describe how to include linguistic variables into the core of this system. Also we want to show the way Bousi∼Prolog may contribute to solve several problems extracted from different application areas, where it is mandatory to deal with linguistic knowledge, such as: deductive databases, knowledge-based systems, data retrieval or approximate reasoning. To accomplish this last goal, through the paper, we discuss and implement several (small but meaningful) examples[4], showing the great potential of this programming language.

2 The Bousi∼Prolog Programming Language

Bousi∼Prolog is an extension of the standard Prolog language. Its operational semantics is an adaptation of the **S**election-function driven **L**inear resolution for **D**efinite clauses

[4] `http://dectau.uclm.es/bousi`

(SLD resolution) principle where classical unification has been replaced by a fuzzy unification algorithm based on proximity relations defined on a syntactic domain. Hence, the operational mechanism is a generalization of the similarity-based SLD resolution principle [32], we name *weak* SLD resolution.

Currently, Bousi∼Prolog is delivered in two implementation formats: a high level and a low level implementation. The high level implementation and some of the main features of this language are described in [18]. One step further, in [16] we presented the structure and main features of a low level implementation for Bousi∼Prolog. It consists in a compiler and an enlargement of the **W**arren **A**bstract **M**achine (WAM) able to incorporate fuzzy unification and to execute BPL programs efficiently. The key to this adaptation, without forcing the structure of the WAM, consists of transforming the BPL source program into an intermediate level code compiling the information provided by a proximity relation. It is important to note that, to the best of our knowledge, this is the first WAM implementation that supports weak SLD resolution. Also, to allow the development of BPL programs, a programming environment for this language has been implemented. In this section we recall the foundations and syntax of Bousi∼Prolog.

2.1 Fuzzy Relations and Weak Unification

A *binary fuzzy relation* on a set U [38] is a fuzzy subset on $U \times U$ (that is, a mapping $U \times U \longrightarrow [0,1]$). A binary fuzzy relation $\mathcal{R}$ is said to be a *proximity relation* if it fulfills the *reflexive* property (i.e. $\mathcal{R}(x,x) = 1$ for any $x \in U$) and the *symmetric* property (i.e. $\mathcal{R}(x,y) = \mathcal{R}(y,x)$ for any $x, y \in U$). A proximity relation which in addition fulfills the *transitive* relation (i.e., $\mathcal{R}(x,z) \geq \mathcal{R}(x,y) \triangle \mathcal{R}(y,z)$, for any $x, y, z \in U$) is said to be a *similarity relation*. The operator '$\triangle$' is an arbitrary t-norm. The notion of transitivity above is $\triangle$-transitive. If the operator $\triangle = \wedge$ (that is, it is the minimum of two elements), we speak of *mim*-transitive or $\wedge$-transitive. It is noteworthy that, when the transitivity flag is enabled Bousi∼Prolog constructs a similarity using *mim*-transitivity.

Bousi∼Prolog implements a *weak unification algorithm* which is an extension of the one appeared in [32], with proximity relations on syntactic domains.

It is formalized as a transition system supported on a proximity-based unification relation "$\Rightarrow$". The unification of two expressions $\mathcal{E}_1 = f(t_1, \ldots, t_n)$ and $\mathcal{E}_2 = g(s_1, \ldots, s_n)$ is obtained by a state transformation sequence starting from an initial state $\langle G, id, \alpha_0 \rangle$, where $G = \{t_1 \approx s_1, \ldots, t_n \approx s_n\}$ is a set of unification problems [5], id is the identity substitution and $\alpha_0 = 1$ is the initial proximity degree:

$$\langle G, id, \alpha_0 \rangle \Rightarrow \langle G1, \theta_1, \alpha_1 \rangle \Rightarrow \ldots \Rightarrow \langle G_n, \theta_n, \alpha_n \rangle.$$

When the final state $\langle G_n, \theta_n, \alpha_n \rangle$, with $G_n = \emptyset$, is reached (i.e., the equations in the initial state have been solved), the expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ are unifiable by proximity with weak most general unifier (w.m.g.u) θ_n and unification degree α_n. Therefore, the final state $\langle \emptyset, \theta_n, \alpha_n \rangle$ signals out the unification success. On the other hand, when expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ are not unifiable, the state transformation sequence ends with failure (i.e., $G_n = Fail$).

[5] Here, the symbol "$\approx$" means that the arguments in $\mathcal{E}_1$ and $\mathcal{E}_2$ are capable to be "equals" by proximity, that is, a substitution σ can be computed such that $\mathcal{R}(\mathcal{E}_1\sigma, \mathcal{E}_2\sigma) > 0$.

The *proximity-based unification relation*,"$\Rightarrow$", is defined as the smallest relation derived by a set of transition rules that behave as in the classical unification algorithm, except for the rules in the following table:

• *Term decomposition rule*:

$$\frac{\langle \{f(t_1,\ldots,t_n) \approx g(s_1,\ldots,s_n)\} \cup E, \theta, \alpha\rangle \ ,\ R(f,g) = \beta > 0}{\langle \{t_1 \approx s_1, \ldots, t_n \approx s_n\} \cup E, \theta, (\alpha \wedge \beta)\rangle}$$

• *Failure rule*:

$$\frac{\langle \{f(t_1,\ldots,t_n) \approx g(s_1,\ldots,s_n)\} \cup E, \theta, \alpha\rangle \ ,\ R(f,g) = 0}{\langle Fail, \theta, \alpha\rangle}$$

In the rules above, E denotes a set of (remaining) equational goals in the preceding state. Note that, when the proximity relation $\mathcal{R}$ is the diagonal relation[6], this algorithm conforms with the classical unification algorithm.

Certainly, the weak unification algorithm we have just presented can be weakened (even more) allowing fuzzy relations that do not fulfill the symmetric property. This last kind of fuzzy relations are useful for introducing the linguistic terms into the BPL system [19].

2.2 Proximity-Based SLD Resolution

Let Π be a set of Horn clauses and $\mathcal{R}$ a proximity relation on the alphabet of a first order language $\mathcal{L}$. Let $\Lambda = \{\lambda_1, ..., \lambda_n\}$ be the set of approximation levels of $\mathcal{R}$. We define *Weak SLD* (WSLD) *resolution* as a transition system $\langle E, \Longrightarrow_{\text{WSLD}}\rangle$ where E is a set of triples $\langle \mathcal{G}, \theta, \alpha\rangle$ (goal, substitution, approximation degree), that we call the *state* of a computation, and whose transition relation $\Longrightarrow_{\text{WSLD}} \subseteq (E \times E)$ is the smallest relation that satisfies:

$$\frac{\begin{array}{c}\mathcal{C} = (\mathcal{A} \leftarrow \mathcal{Q}) \ll \Pi, \\ \sigma = wmgu(\mathcal{A},\mathcal{A}') \neq fail,\ \beta = \mathcal{R}(\mathcal{A}\sigma,\mathcal{A}'\sigma)) \geq \lambda\end{array}}{\langle (\leftarrow \mathcal{A}', \mathcal{Q}'), \theta, \alpha\rangle \ \Longrightarrow_{\text{WSLD}}\ \langle \leftarrow (\mathcal{Q}, \mathcal{Q}')\sigma, \theta\sigma, \alpha \wedge \beta\rangle}$$

where $\mathcal{Q}$, $\mathcal{Q}'$ are conjunctions of atoms, the notation "$\mathcal{C} \ll \Pi$" is representing that $\mathcal{C}$ is a standardized apart clause in Π, and that the value λ is a cut value in Λ, which imposes a limit to the expansion of the search space in a computation. We say that the performed step is a *step of level* λ because the computed approximation degree is greater than or equal to λ.

A WSLD *derivation of level* λ for $\Pi \cup \{\mathcal{G}_0\}$ and $\mathcal{R}$ is a sequence of steps of level λ: $\langle \mathcal{G}_0, id, 1\rangle \ \Longrightarrow_{\text{WSLD}}\ \ldots\ \Longrightarrow_{\text{WSLD}}\ \langle \mathcal{G}_n, \theta_n, \beta_n\rangle$. That is, each $\beta_i \geq \lambda$. And a WSLD *refutation of level* λ for $\Pi \cup \{\mathcal{G}_0\}$ and $\mathcal{R}$ is a WSLD derivation of level λ for $\Pi \cup \{\mathcal{G}_0\}$ and $\mathcal{R}$: $\langle \mathcal{G}_0, id, 1\rangle \Longrightarrow_{\text{WSLD}}{}^{*} \langle \Box, \sigma, \beta\rangle$, where the symbol "$\Box$" stands for the empty clause, σ is the *computed substitution* and β is its *approximation degree*. The output of a WSLD refutation is the pair $\langle \sigma, \beta\rangle$, which is said to be the *computed answer*. Certainly, a WSLD refutation computes a family of answers, in the sense that, if $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ then, by definition, whatever substitution

[6] That is, $\mathcal{R}(a,a) = 1$ and $\mathcal{R}(a,b) = 0$ (when a and b are distinct symbols in the alphabet).

$\theta' = \{x_1/s_1, \ldots, x_n/s_n\}$, holding that $\mathcal{R}(s_i, t_i) \geq \lambda$, for any $1 \leq i \leq n$, is also a computed substitution with approximation degree $\beta \wedge (\bigwedge_1^n \mathcal{R}(s_i, t_i))$.

Observe that our definition of proximity based SLD resolution is parameterized by a cut level $\lambda \in \Lambda$. This introduces an important conceptual distinction between our approach an the similarity based SLD resolution presented in [32].

2.3 Semantic Unification

The weak unification algorithm described in Subsection 2.1 is syntactical in nature, since the symbols involved in the unification process are treated syntactically. However, when we want to unify linguistic terms, with fuzzy sets associated as their meaning, the unification process relies on semantics. In this section we recall an efficient generic approach to include semantic unification in the framework of fuzzy logic programming that first appear in [19]. Our approach differs from others because it compiles the information provided by a linguistic variable into a set of fuzzy relations.

Formally, our approach relies on the concepts of a linguistic variable [39] and a fuzzy relation. A *linguistic variable* is a quintuple $\langle X, T(X), U, G, M\rangle$ where: X is the variable name, $T(X)$ is the set of linguistic terms of X (i.e., the set of names of linguistic values of X), U is the domain or universe of discourse, G is a grammar that allows to generate $T(X)$ and M is a semantic rule which assigns to each linguistic term x in $T(X)$ its meaning (i.e., a fuzzy subset of U —characterized by its membership function μ_x—).

In this work we are interested in the definition of a fuzzy relation on the set of terms, $T(X)$, associated to a linguistic variable, X, in a way such that the linguistic variable (including its semantic component) could be treated at a purely syntactic level. To this end, we proceed as follows:

> Suppose that $T(X) = \{x_i \mid i \in I\}$, where I is a set of indexes. For each x_i and x_j, with $i, j \in I$, we generate the entry of a fuzzy relation on $T(X)$: $\mathcal{R}(x_i, x_j) = \alpha$. The relationship degree α can be calculated as the relation between the fuzzy subsets $M(x_i)$ and $M(x_j)$ associated to these terms as meaning.

For this purpose, we can make use of *fuzzy matching* techniques such as the ones developed in [6, 7], which have been successfully used in the system *FuzzyClips* [8]. Once the fuzzy relation, $\mathcal{R}$, has been generated, the operational mechanism of the language manipulates the linguistic variable X and, more precisely, the terms in $T(X)$ in a totally standard way. That is, as symbols of a first order language which are capable of participating in a weak unification process at the same level as the rest of symbols of language alphabet. Therefore, we are able to manipulate a semantic unification process by means of a weak unification algorithm which is syntactical in nature.

Ending this subsection, it is important to note that, if we want to cope with the distinction between general and specific knowledge introduced in [31], the fuzzy relation constructed during this process must fulfill the reflexive property but not necessarily the symmetric and/or transitive properties. In [31], and lately in [1], it is shown the importance of distinguish between general and specific knowledge. For instance: let

Age be a linguistic variable and `young` and `between_19_22` two terms in $T(Age)$; it is clear that the meaning of the linguistic term `between_19_22` is included into the meaning of the linguistic term `young` and not viceversa; therefore, the fuzzy relation between `young` and `between_19_22` should not be symmetric, if we want to represent this knowledge properly. Taken into account this observation, our proposal not only allows us to incorporate fuzzy sets in an easy, efficient way, solving some implementation problems mentioned in [37], but it also takes into account the distinction between general and specific knowledge made in [31].

2.4 The Bousi∼Prolog Syntax

The BPL syntax is mainly the Prolog syntax but enriched with a built-in symbol "∼" used for describing proximity relations[7] by means of what we call a "proximity equation". *Proximity equations* are expressions of the form: "`<symbol> ~ <symbol> = <degree>.`". Although, a proximity equation represents an arbitrary fuzzy binary relation, its intuitive reading is that two constants, n-ary function symbols or n-ary predicate symbols are approximate or similar with a certain degree. That is, a proximity equation $a \sim b = \alpha$ can be understood in both directions: a is approximate/similar to b and b is approximate/similar to a with degree α. Therefore, a BPL program is a sequence of Prolog facts and rules followed by a sequence of proximity equations.

In order to obtain a similarity relation, Bousi∼Prolog generates automatically a reflexive, symmetric and transitive closure, starting from an arbitrary fuzzy binary relation. This feature gives a substantial support for the programmer, since it is not easy to define a similarity relation due to the transitivity constrains, which may contradict the initial proximity values. The formal aspects of this problem and an algorithm for the construction of a transitive closure can be found in [15]. From the syntactical point of view, the construction of the transitive closure is enabled or disabled by means of the directive `transitivity`. The syntax of this directive is as follows: "`:-transitivity(B).`", where B is "`yes`" or "`no`". If B equals to "`yes`" then a similarity relation is generated starting from the proximity equations provided by the user. Otherwise, if B is "`no`" a proximity relation is generated (which is the default option).

We can impose a limit to the expansion of the search space in a computation by what we called a "lambda cut". When the LambdaCut flag is set to a value different than zero, the weak unification process fails if the computed approximation degree goes below the stored LambdaCut value. Therefore, the computation also fails and all possible branches starting from that choice point are discarded. By default the LambdaCut value is zero. However, the lambda cut flag can be set to a different value by means of a lambdacut directive: "`:-lambdaCut(N).`", where N is a number between `0` and `1`. Of course, Bousi∼Prolog also implements the cut operator, "`!`", a non declarative built-in operator which allows us to prune the search space during the program execution. In this case branches are cut unconditionally, without checking the computed approximation degree.

[7] Actually, fuzzy binary relations which are automatically converted into proximity or similarity relations.

The BPL language makes use of two directives to define and declare the structure of a linguistic variable [39]: `domain` and `fuzzy_set`. The `domain` directive allows to declare and define the universe of discourse or domain associated to a linguistic variable. The concrete syntax of this directive is:

```
:-domain(Dom_Name(n,m,Magnitude)).
```

where, `Dom_Name` is the name of the domain, n and m (with $n < m$) are the lower and upper bounds of the real subinterval $[n, m]$, and `Magnitude` is the name of the unit wherein the domain elements are measured. For example, the directive "`:-domain(age(0,100, years)).`" defines a domain with name `age`, whose values are numbers (between 0 and 100) measured in `years`. The `fuzzy_set` directive allows to declare and define a list of fuzzy subsets (which are associated to the primary terms of a linguistic variable) on a predefined domain. The concrete syntax of this directive is:

```
:-fuzzy_set(Dom_Name,[FSS_1(a1,b1,c1[,d1]),
                      ...,
                      FSS_n(an,bn,cn[,dn])]).
```

Fuzzy subsets are defined by indicating their name, `FSS_1`, and membership function type. At this time, it is possible to define two types of membership functions: either a *trapezoidal function*, if four arguments are used for defining the fuzzy subset or a *triangular function*, if three arguments are used. It is important to note that, in general, these kind of functions are well adapted to the definition of any concept, with the advantage of their simplicity (what contributes to their efficient implementation).

Once a domain and the fuzzy sets associated to the primary terms have been declared, composite terms may be generated through the following grammar:

```
<Term> ::= <Atomic_term> | <Composite_term>
<Composite_term> ::= <TModif>#<Atomic_term>
<TModif> ::= very | somewhat |
             more_or_less | extremely
```

Additionally, a BPL program may include what we call "domain points" and "domain ranges". A *domain point* is our practical artifice to represent a precise crisp value in the universe of discourse, aiming to compare it with other linguistic terms.

Domain points have a different behavior (meaning) depending on weather they represent specific or general knowledge[8]. In the first case, the meaning assigned to a domain point will be its membership function. In the second case, the domain point will be fuzzified into a fuzzy singleton set. Domain points are generated by means the following grammar.

```
<Dom_Point> ::= <Dom_Name>#<Dom_Value> |
                <Composite_DP>
<Composite_DP> ::= <PModif>#<Dom_Point>
<PModif> ::= about
```

[8] See [31] for a deeper explanation on the general/specific knowledge problem.

where <Dom_Value> is the set of symbols representing the elements in the universe of discourse. Note that a domain point can be transformed in a linguistic label associated to a fuzzy subset when it is preceded by the modifier "about".

A *domain range* is a label for an interval of elements in the universe of discourse. Domain range labels are generated by means of the following grammar:

```
<Dom_Range> ::= <Composite_DR>
   | <Dom_Name>#<Dom_Value>#<Dom_Value>
<Composite_DR> ::= <RModif>#<Dom_Range>
<RModif> ::= about
```

As in the previous case, a domain range can be fuzzified adding the modifier about. The use of linguistic terms is shown at a detailed level in Section 4.

3 Design and Implementation of Bousi∼Prolog.

The architecture of the BPL low level implementation is a multi-layer architecture with three layers: the programming environment, the compiler and the similarity abstract machine (SWAM). It consists of over 6500 code lines, divided into 27 classes. It has been implemented in Java, since this is an object oriented language that possesses facilities to deploy the BPL system on the web.

3.1 Similarity-based WAM.

The Similarity-based WAM (SWAM) modifies the two main parts of a standard WAM: the memory layout and the instruction set [16].

- *Extension of the memory layout.* The main changes are related to the incorporation of data structures that allow us to manage proximity or similarity relationship. We add the so called *Proximity Matrix Area*, which stores an adjacency matrix representation of a proximity or similarity relation. Two new specific registers: the *Approximation Degree register* (AD), which stores the current computed approximation degree of a derivation; and the *Lambda-Cut register* (LC) which stores the lower bound for the approximation degree in a derivation. Also, we need to modify the standard choice point frame structure by adding a new field, D, to save the value stored in the AD register, prior to the creation of a choice point.
- *Extension of the instruction set.* As the choice point frame structure has been modified, the machine instructions that work in combination with it need also to be modified. That is, we have to modify the instructions: try_me_else, retry_me_else, trust_me and backtrack. The first three machine instructions essentially update the value of the AD register and backtrack helps to recover the old approximation degree value after a failure. On the other hand, also it is necessary to modify some machine instructions involved in the process of argument unification, such as: get_structure, get_constant and unify.

The SWAM is executed as a thread inherited from the class Thread. The idea is to allow a programmer to use interactive predicates or I/O predicates like, for instance, the predicate read, for which it is necessary that the SWAM keeps waiting while the user writes the input data.

3.2 The Bousi∼Prolog compiler.

One of the most important features of the BPL system is its ability to compile all the information regarding the linguistic variables defined in a program. In this subsection the different compilation phases of the data type fuzzy set are detailed.

The first phase is the *Syntactical Analysis*. During this phase it is verified whether there exist syntactic errors in the source program. At the same time, the syntactic tree, which is the basis for later code generation, is built. Additionally, in this phase, the directives "`domain`" and "`fuzzy_set`" are read and the domains and associated fuzzy subsets are built. The directive `domain` triggers a procedure which creates an object of type domain. Such an object is composed of a name, a range and a magnitude. On the other hand, the directive `fuzzy_set` triggers a procedure that creates an object of type fuzzy set. An object of type fuzzy set is composed by a domain (a reference to the domain created previously), and a list of fuzzy subsets. In turn, each fuzzy subset is formed by a linguistic label working as an identifier, which may be used as a regular symbol of a first order alphabet[9] and a membership function determined by the parameters which are passed as the arguments of either a trapezoidal function $f(x, a, b, c, d) = max(min((x - a)/(b - a), 1, (d - x)/(d - c)), 0)$ or a triangular function $f(x, a, b, c) = max(min((x - a)/(b - a), (c - x)/(c - b)), 0)$.

Also, during the syntactical analysis phase the occurrence of composite terms, domain ranges and domain points in the source program is detected. These terms are included into the table of linguistic terms. The meaning assigned to composite terms is a membership function which depends on the modifier applied to their primary terms ($very(y) = y^2$, $somewhat(y) = y^{0.333}$, $more\_or\_less(y) = \sqrt{y}$ and $extremely(y) = y^3$). On the other hand, the meaning associated to a domain range `domain_name#a#b` is a membership function μ defined by the trapezoidal function $f(x, a, a, b, b)$. That is, $\mu(x) = 1$ if $a \leq x \leq b$ and $\mu(x) = 0$ otherwise. In other words, a domain range is a crisp subset. As it has been indicated in the Section 2.4, a domain range admits the modifier `about`. We associate the fuzzy subset defined by the trapezoidal function $f(x, max(a-F, n), a, b, min(b+F, m))$ to the linguistic label `about#domain_name#a#b`, where $F = (m - n) \times 2.5/100$ is a fuzzification factor for the crisp set we start from. We recall that n and m are, respectively, the lower and upper bound of the interval $[n, m]$ that forms the universe of discourse.

A domain point has no meaning assigned, since it is not properly considered as a fuzzy subset in our approach. This design decision will require us a special treatment of such kinds of terms in the next compilation phase. A domain point also admits the modifier `about`. For the label `about#domain_name#a`, we associate a fuzzy subset defined by the triangular function $f(x, max(a - F, n), a, min(a + F, m))$, where F (defined as in the previous case) is a fuzzification factor which converts the value a into a fuzzy subset.

We note that we shall only store information of those linguistic terms that appear in the source code of a BPL program. That is, it is not necessary to generate, for instance, all the composite terms derived from primary ones.

The *Generation of fuzzy relations* phase is the next step of the compilation process. It is focused on generating fuzzy relations between linguistic terms in order to

[9] That is, as a constant, a function or, even, a predicate symbol.

compile all the semantic information associated with them. It is implemented by means of the algorithm 1, which takes into account the distinction between general and specific knowledge. For the sake of simplicity, in the description of this algorithm, we focus our attention on linguistic terms associated to one single linguistic variable.

Algorithm 1

Input*: A Subset* $S = \{\langle x_i, \mu_{x_i}\rangle \mid i \in I\}$ *(where* I *is a set of indexes) of terms/meanings of a linguistic variable* X.

Output*: A set* $\mathcal{R}$ *of entries which defines a fuzzy relation on* S.

Initialization*:* $\mathcal{R} := \emptyset$

For each $\langle x_i, \mu_{x_i}\rangle$ **and** $\langle x_j, \mu_{x_j}\rangle$, **with** $i, j \in I$, **do**

Case of

1. $\mu_{x_i} \neq \bot$ **and** $\mu_{x_j} \neq \bot$*:*
   $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(x_i, x_j) = match(\mu_{x_j}, \mu_{x_i})\}$*;*
2. $\mu_{x_i} \neq \bot$ **and** $\mu_{x_j} = \bot$*:*
   Let $x_j = dom\#u_j$ **in** $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(x_i, x_j) = \mu_{x_i}(u_j)\}$*;*
3. $\mu_{x_i} = \bot$ **and** $\mu_{x_j} \neq \bot$*:*
   Let $x_i = dom\#u_i$ **in** $\mu_{x_i} := singleton(u_i)$*;*
   $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(x_i, x_j) = match(\mu_{x_j}, \mu_{x_i})\}$*;*

endCase

endFor

Return $\mathcal{R}$

The last algorithm deserves some comments. First, it is noteworthy that, the subset S only contains the primary terms in $T(X)$ an those composite terms occurring in the source program. This is a good design option in order to keep controlled the size of the fuzzy relation.

If we want to deal with the distinction between general and specific knowledge, as it is done in [31], patterns passed to the arguments of a relation must be classified as either general or specific. In a logic programming language the arguments of goals and atomic formulas in the body of program clauses contain the general knowledge, whilst the arguments of facts an the head of program clauses contain the specific knowledge. On the other hand, due to the operational features of the weak SLD resolution principle, for an entry $\mathcal{R}(x, y)$ of the fuzzy relation constructed by Algorithm 1, the argument x is classified as general information and the argument y as specific information.

The matching function calculates the degree of relation between two fuzzy subsets $\mathcal{F}$ and $\mathcal{F}'$ by using a resemblance measure. Following [6, 7], this measure is supported by the concepts of possibility P and necessity N. More precisely, this function has been defined as follows: $match(\mathcal{F}, \mathcal{F}') =$

$$\begin{cases} P(\mathcal{F}, \mathcal{F}') & \texttt{if } N(\mathcal{F}, \mathcal{F}') > 0.5; \\ (N(\mathcal{F}, \mathcal{F}') + 0.5) * P(\mathcal{F}, \mathcal{F}') & \texttt{otherwise.} \end{cases}$$

where, $P(\mathcal{F}, \mathcal{F}') = max\{min(\mu_{\mathcal{F}}(u), \mu_{\mathcal{F}'}(u)) \mid u \in U\}$ and $N(\mathcal{F}, \mathcal{F}')$=$1 - P(\overline{\mathcal{F}}, \mathcal{F}')$, being $\overline{\mathcal{F}}$ the complement of $\mathcal{F}$ described by the membership function: $\mu_{\overline{\mathcal{F}}}(u) = 1 - \mu_{\mathcal{F}}(u), \forall u \in U$.

The $matching$ function has an asymmetric behavior. Therefore, it is suitable to deal with the distinction between general and specific knowledge when building a fuzzy relation starting from two linguistic terms with associated fuzzy sets as meaning. However, when the fuzzy unification process involves domain points some subtleties arise and we need to perform a special treatment.

Once the compilation phase is concluded, all the information represented by fuzzy subsets has been stored in the fuzzy relation defined on the set of their linguistic labels. Thanks to this artifice, the execution of a program proceeds on a standard way, following the weak resolution mechanism. After that, *the Similarity Generator* calculates a proximity relation or a similarity relation (if the transitivity option is enabled) starting from the proximity equations provided by the programmer; the proximity/similarity relation is stored into the Proximity Matrix memory area and its information is used at compilation time, by the Adapter, and at execution time, when it is necessary during the unification process. Next, *the Adapter* takes the syntactic tree and the proximity or similarity relation and constructs an intermediate representation which is used by the Code Generator to obtain the object code. Finally, *the Code Generator* produces the machine code associated to the source program; once the machine code is generated, it is stored in the Code Area, an addressable array of memory words.

4 Applications of Bousi∼Prolog

In this section we describe how Bousi∼Prolog may contribute to resolve several problems extracted from different application areas, where it is mandatory to deal with vagueness and imprecision, such as: flexible deductive databases, fuzzy control, fuzzy experts systems, data retrieval or approximate reasoning. Note that Bousi∼Prolog allows the specification of a problem in a linguistic and declarative way.

4.1 Flexible Query Answering in Deductive Databases

Databases are software components that serve to collect and store information (that users can retrieve, add, update or remove). As the real world information is often permeated by vagueness and imprecision, database systems should deal with this topic. Also, they should permit to retrieve information in a flexible way. These capabilities (among others, like integrating semi-structured —XML— and unstructured —textual— data) are highly appreciated in the field of information systems and have led up to the notion of flexible databases.

There are several approaches to fuzzy flexible databases. We highlight two of them: i) the model of Buckles-Petry [5] and Shenoi-Melton [34], where a relational database is fuzzified by means of a similarity/proximity relation defined on an universal domain; ii) the model of Prade-Testemale [29] in which attribute values may be expressed in (vague) linguistic terms (roughly speaking, using fuzzy sets). In this section we show how Bousi∼Prolog allows to simulate both fuzzy flexible database approaches effectively. We begin with an example illustrating how to implement a fragment of a flexible database in the the Buckles-Petry and Shenoi-Melton style. This example is adapted from a work of Motro [26], although the techniques used there differs from ours. In

Motro's work the relational data model is extended using *data metrics*, and the query language is extended with a single feature, a *similar-to comparator*. On the other hand, we concentrate on deductive databases, i.e., databases consisting of facts and rules that have a deductive capability.

Assume a database storing information on films which are shown at some cinema of a specific neighborhood of Los Angeles city. The database consists of three tables (relations, represented by BPL facts) with a total of eight attributes. The `film` table has three attributes: title, director and category of the film. The `theater` table is characterized by the theater name, owner and location of the theater. The `engagement` table is used to link the information stored in the first two tables and it has two attributes: the title of the film and the name of the theater. The fuzzy component is defined by two proximity relations. The first one states the similarity between the different film categories (i.e., it is defined on the syntactic domain of film categories) and the second one states the closeness of two theater locations (i.e., it is defined on the syntactic domain of theater locations). In this example, both fuzzy relations are implemented explicitly by means of a set of proximity equations.

```
%% DIRECTIVE
:-lambdaCut(0.5).

%% PROXIMITY RELATIONS
%% Location Distance Relationship
bervely_hills~downtown=0.3.
bervely_hills~santa_monica=0.45.
bervely_hills~hollywood=0.56.
bervely_hills~westwood=0.9.
downtown~hollywood=0.45.
downtown~santa_monica=0.23.
downtown~westwood=0.25.
hollywood~santa_monica=0.3.
hollywood~westwood=0.45.
santa_monica~westwood=0.9.

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

%% 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).

%% 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(input, input, output, output)
search(Category, Location, Film, Theater) :-
              film(Film, _, Category),
              engagement(Film, Theater),
              theater(Theater, _, Location).
```

The predicate `search/4` allows us to know the cinema which is showing a film closest to our location and category of preference. If we launch the goal "`search(adventure, westwood, Film, Theater).`", the system answers: "`Film = rear_window, Theater = egyptian, with 0.9`" (a suspense film located at `west- wood`), "`Film = surf_party, Theater = vi- llage, with 0.6`" (a drama film located at `westwood`), "`Film = robbery, Theater = odeon, with 0.9`" (a suspense film located at `santa_monica`), "`Film = four_feathers, Theater = music_hall, with 0.9`" (an adventure film located at `bervely_hills`).

Next, we present a BPL program implementing a fragment of a flexible deductive database in the style of Prade and Testemale. That is, databases that incorporate the notion of fuzziness by means of fuzzy sets that may be used as attributes of a table.

In this example we model a database fragment for a real state company with information about flats to be hired. The company wants to help clients to select flats in stock, according their preferences.

```
%% DIRECTIVES
%% 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: 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(conditions, [unfair(0,0,1,3),
   fair(1,3,6), good(4,6,8),
   excellent(7,9,10,10)]).

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

%% RULES
%% close_to(Flat, District)
close_to(Flat, District):-
flat(Flat, Street, _, _),
street(Street, Flat_Dist),
distance(Flat_Dist, District, close).

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

Now `select_flat/2` selects those flats which may be considered `cheap`, `good` and `close` to the `campus` with a certain degree. More precisely, if we launch the goal "`?- select_flat(Flat, Street).`", we obtain: "`Flat = f1, Street = libertad, with 0.8`" and "`Flat = f2, Street = ciruela, with 0.2`", being flat `f3` at `granja` street disregarded.

Finally, it is noteworthy that in many cases it will be necessary, or preferable, to combine both approaches, in order to model a problem properly. That is, to declare and define linguistic variables in conjunction with proximity relations. This last option is completely feasible in Bousi∼Prolog.

4.2 Knowledge-Based Systems

Knowledge-Based Systems focuses on systems that use knowledge-based techniques to support decision-making, learning and action [25]. As an example of Knowledge Based System, we implement a fuzzy logic controller which is an adaptation of the one described in [33]. In the general model, the device being controlled has a set of activators that take the input and use this to affect its settings and a set of sensors that get information from the device. The fuzzy logic controller takes the 'crisp' information from the sensors as input and fuzzifies this into some linguistic variables, propagates membership values using linguistic rules and, finally, defuzzifies the output and returns these 'crisp' values to the activators. The controller we shall implement will use a set of fuzzy rules to adjust the throttle of a steam turbine according to its current temperature and pressure to keep it running smoothly. First we need to decide upon the linguistic variables that are going to be used in the BPL program. We do this by looking at the descriptors that will be used by the rules, in this case 'temperature', 'pressure' and 'throttle':

```
:-domain(temperature(0,500,Celsius)).
:-fuzzy_set(temperature,
    [cold(0,0,110,165), cool(110,165,220),
     normal(165,220,275), warm(220,275,330),
     hot(275,330,500,500)]).

:-domain(pressure(0,300,kpa)).
:-fuzzy_set(pressure,
    [weak(0,0,10,70), low(10,70,130),
     ok(70,130,190), strong(130,190,250),
     high(190,250,300,300)]).

:-domain(throttle(-60,60,rpm)).
:-fuzzy_set(throttle,
    [neg_large(-60,-60,-45,-30),
     neg_medium(-45,-30,-15),
     neg_small(-30,-15,0), zero(-15,0,15),
     pos_small(0,15,30), pos_medium(15,30,45),
     pos_large(30,45,60,60)]).
```

For the 'throttle' variable, a negative value indicates that the throttle should be moved back and a positive value that it should be moved forward. The problem results in moving the throttle by large, medium or small amounts in the negative and positive directions. After defining the linguistic variables used by our program, we can proceed to define the rules. In this example the rules will cover all the possible combinations of temperature and pressure and give a resultant throttle change for each. An example of a rule specified by an expert is: ***If the temperature is cold and the pressure is weak then increase the throttle by a large amount***. This rule has a direct translation to BPL code:

```
throttle(positive_large):-
        temperature(cold), pressure(weak).
```

In particular, the remainder rules for a situation of `cold` temperature and for a situation of `cool` temperature are:

```
%% Cold
throttle(positive_medium):-
        temperature(cold), pressure(low).
throttle(positive_small):-
        temperature(cold), pressure(ok).
throttle(negative_small):-
        temperature(cold), pressure(strong).
throttle(negative_medium):-
        temperature(cold), pressure(high).
%% Cool
throttle(positive_large):-
        temperature(cool), pressure(weak).
throttle(positive_medium):-
        temperature(cool), pressure(low).
throttle(zero):-
        temperature(cool), pressure(ok).
throttle(negative_medium):-
        temperature(cool), pressure(strong).
throttle(negative_medium):-
        temperature(cool), pressure(high).
```

We can define the rest of rules (for `normal`, `warm`, and `hot` temperature), in a similar *linguistic* declarative way.

The inputs to the logical system should be taken from the sensors of the real device, but in this simple example they are modeled by facts:

```
%% Facts
temperature(temperature#300).
pressure(pressure#150).
```

Running this BPL program, it is possible to return a 'crisp' value for the throttle by means of the BPL built-in predicate `defuzzify` [10]. If we launch the goal "`?-defuzzify(throttle(_),Y).`" we obtain the answer "`Y = throttle#-14.09`".

It is noteworthy that this BPL program employs very few extensions to the Prolog syntax, obtaining a natural codification of this problem that can be understood easily by a Prolog programmer [11].

4.3 Information Retrieval and the Semantic Web.

Bousi∼Prolog allows us to implement declarative approaches to text cataloging using the ability of Bousi∼Prolog for flexible matching and knowledge representation by

[10] As a reasonable starting point we are using the standard method of "*average of the maximum*" as defuzzifying method.

[11] We suggest to compare our solution with the one appeared in [33], in order to observe how natural is our codification of this problem.

means of an ontology of terms modeled by a set of proximity equations. The following example shows how proximity equations can be used as a fuzzy model for information retrieval where textual information is selected or analyzed using an ontology [12], that is, a structured collection of terms that formally defines the relations among them. Hence, inside a semantic context instead of a purely syntactic context. This is an extreme useful application in a world dominated by the Semantic Web [3], where people are exposed to great amounts of (textual) information. A practical textual inference task is finding concepts which are structurally analogous. Proximity equations can help in this task. The set of proximity equations shown below has been obtained using ConceptNet [24], a freely available commonsense knowledge-base and natural-language-processing toolkit[12]. ConceptNet is structured as a network of semi-structured natural language fragments. It has a `GetAnalogousConcepts()` function that returns a list of structurally analogous concepts given a source concept. In our case the source concept is "`wheat`". The degree of structural analogy between terms is also provided by ConceptNet. The set of proximity equations is a partial view of the original output, which was hand-made adapted by ourselves.

We want to extract information on terms structurally analogous to "`wheat`" on a given text. The fragment text of our example is one classified with the label (topic) "`wheat`" and processed by erasing stop words and the endings of a word stem.

```
% DIRECTIVE
:- transitivity(yes).

% PROXIMITY EQUATIONS
wheat~bean=0.315.    bean~crop=0.315.
wheat~corn=0.315.    bean~child=0.33.
wheat~grass=0.315.   bean~grass=0.315.
wheat~horse=0.315.   bean~flower=0.315.
wheat~human=0.205.   bean~horse=0.335.
bean~animal=0.35.    bean~potato=0.5.
bean~corn=0.48.      bean~table=0.35.

% FACTS and RULES
%% searchTerm(T,L1,L2), true (with approxima-
%% tion degree 1) if T is a (constant) term,
%% L1 is a list of (constant) terms (represen-
%% ting a text) and 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).

insert(t(T,N,D),[],[t(T,N,D)]).
```

[12] Available at `http://conceptnet.media.mit.edu/`.

```
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).
```

A simple session with the BPL system, launching the goal "`?- g(corn,L).`" provides the answer: "`L = [t(potato, 1, 0.48), t(bean, 2, 0.48), t(corn, 1, 1.0)] with 1.0`". The information returned by the goal can be used lately to analyze the input text or to obtain a degree of preference in a retrieval process.

It is noteworthy, that using similar techniques to the above described here, in [30] we have presented a declarative approach to text categorization using the ability of Bousi∼Prolog for flexible matching and knowledge representation by means of an ontology of terms modeled by a set of proximity equations.

4.4 Approximate Reasoning

Approximate reasoning is basically the inference of an imprecise conclusion from imprecise premises. All the examples discussed along this paper concerns with approximate reasoning in some degree. However, in this section we want to make a reflexion on fuzzy inference, such as it was formalized by Zadeh [39, 40], and how Bousi∼Prolog can deal with this kind of reasoning.

In this approach, each granule of knowledge is represented by a fuzzy set or a fuzzy relation on the appropriate universe. The premises of an argument are expressed as fuzzy rules and a *fuzzy inference* is a generalization of *modus ponens* that can be formalized as: "`if` x `is` F `then` y `is` G" and "x `is` F'" then "y `is` G'". Roughly speaking, x and y are variables that takes values on ordinary sets U and W, F and F' are fuzzy subsets on U whilst G and G' are fuzzy subsets on W. Several methods have been proposed to compute G', though there is not consensus as to which is the best. The method proposed by Zadeh consists of identifying a fuzzy relation from F and G, R on U and W, which has a consequence over G' on W.

Bousi∼Prolog proceeds in a different way. It constructs a fuzzy relation over the fuzzy domains (more accurately, the set of linguistic labels in a linguistic variable) involved in a BPL program. This fuzzy relation is built at compile time. Afterwards, at run time, it is used by the weak SLD resolution procedure to infer an answer to a query. For instance, Bousi∼Prolog models the following fuzzy inference in a very natural way: "`if` x `is young then` x `is fast`" and "`Robert is middle`" then "`Robert is somewhat fast`".

```
:-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).
```

Now, if we launch the goal:

```
?- speed(robert,somewhat#fast)
```

the BPL system answers "`Yes with 0.375`".

5 Related Work

At the best of our knowledge, the inclusion of linguistic terms into the field of logic programming was first suggested in [6], where some techniques for solving the problem of matching fuzzy constants were introduced. Lately, in [2] a fuzzy pattern matching algorithm based on Baldwin's mass assignment theory was named "*semantic unification*". Also in [7] a fuzzy pattern matching method was developed which is based on necessity and possibility measures. This method has been successfully used in the system *FuzzyClips* [8] and we adopted it for computing some relationship degrees when generating fuzzy relations in Algorithm 1.

More recently, Virtanen [36, 37] presented a fuzzy unification algorithm based on fuzzy equality relations (i.e., similarity relations) indicating the degree of resemblance of two linguistic terms (how to compute these degrees are let unspecified). The proposed algorithm propagates similarity degrees to the bindings of variables and fuzzy equality entries to an auxiliary structure called "Fuzzy Equality Reference". Variables are bound to a set of candidate terms and a set of similarity degrees forming what is called a "pre-substitution". Lately the most suitable fuzzy set is selected and a fuzzy unification degree is obtained. Some of these features were inherited by other semantic unification methods lately.

Rios-Filho and Sandri [31] first introduce the distinction between general and specific knowledge, giving raise to what they call a *contextual fuzzy unification* algorithm. Contrary to [4], [36] or [1] it does not propagate partial matching measures. It uses different classes of measures to verify the matching between two fuzzy constants. Depending on their origin it uses: *inclusion measures* for comparing a general information constant with regard to a specific information constant; *resemblance measures* otherwise. These measures are ordinary relations by definition. They are employed during the so called *decision phase* which returns a condition failure if the expressions do not match. Therefore, the behavior of this algorithm is more rigid than ours and it does not deliver a unification degree.

The work [1] can be seen as a refinement in the line of contextual fuzzy unification. Alsinet and Godo make a clear separation between the syntactic and the semantic

component of a linguistic variable from which we take inspiration. This separation is inherited by their fuzzy unification algorithm. Also they established a similarity measure for quantifying the inclusion degree between two fuzzy constants. The so called *similarity degree* is used in the computation of a *unification degree* associated to a most general unifier of two expressions.

All these proposals share some features in common: (i) In general, they are complex algorithms with a proliferation of cases and sophisticated data structures taken into account sets of linguistic term candidates and propagating partial matching measures. In contrast, our proposal appears to be more simple and structured; (ii) the whole fuzzy unification process is developed at run time managing a considerable amount of intermediate information. We think this may be harmful for efficiency. However, our algorithm can be seen as a two phase procedure where the semantic component of linguistic terms is preprocessed at compile time. The fact that the relationship between linguistic terms is compiled, jointly with the simplicity of the weak unification algorithm, which uses these fuzzy relations, make our proposal more efficient; (iii) they do not work with function symbols and variables only can be bound to (constant) linguistic terms (not general terms). Neither a linguistic term can play the role of a predicate. In general, all these proposals impose severe syntactical limitations. For instance, in Virtanen [37] only the so called linguistic predicates are considered. That is, atomic formulas such as $p(y)$ or $p(x, y)$ where p is the name of a linguistic variable an exactly one argument y is a linguistic term in $T(p)$. However, in our case, it is possible to treat linguistic terms as function or predicate symbols and no limitations are imposed to the first order syntax of the language. This feature may produce big benefices with little effort. For example, Bousi∼Prolog has greater expressivity and can deal with approximate reasoning easily. Think in the following BPL program:

```
wise(X) :- old(X).
very#old(john).
```

If we ask about whether `john` is a `wise` person, "`?-wise(john)`", the BPL system answers "`Yes with 1.0`" (because, the term `very#old` is included into the term `old`).

6 Conclusions and Future Work

Bousi∼Prolog is an extension of the standard Prolog language with a fuzzy unification algorithm based on proximity relations which allows us to work with both fuzzy linguistic and linguistic tools. In this paper, its design and implementation have been detailed and through a number of (small but meaningful) examples extracted from different application areas (such as deductive databases, knowledge-based systems, data retrieval or approximate reasoning), we have shown the potential power of it for these areas and how it is an useful tool for dealing with vagueness and/or imprecision. It is noteworthy that Bousi∼Prolog employs very few extensions to the Prolog syntax, obtaining a natural codification for vague problems that can be understood easily by a Prolog programmer. This is to remark that, the main difference between the BPL system and a standard Prolog system is in the inside, not in the surface. This feature should be essential in any fuzzy logic programming language.

As a mater of future work, we have to investigate how to extend the weak SLD resolution rule with the ability of reasoning with antonyms in order to gear Bousi∼Prolog

towards a more natural fuzzy linguistic framework. We envision that this problem is closely related with negation treatement. On the other hand, we should incorporate: (graphical) tools for helping the programmer to define fuzzy sets; other fuzzy matching options; and better techniques to defuzzify the output of a computation involving linguistic variables. Also, the BPL system should allow the programmer to define its own linguistic modifiers. Finally, we want to continue enhancing the BPL system in order to make it a suitable tool for *Soft Computing*.

References

1. T. Alsinet and L. Godo. Fuzzy Unification Degree. In *Proc. 2nd Int Workshop on Logic Programming and Soft Computing'98, Manchester (UK)*, page 18 (1998).
2. J.F. Baldwin Evidential support logic programming Fuzzy Sets and Systems, 24, pp 1–26 (1987).
3. T. Berners-Lee, J. Hendler, and O. Lassila. *The semantic web. Scientific American*, May (2001).
4. J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. The implementation of FProlog – a fuzzy prolog interpreter. Fuzzy Sets and Systems, 23, pp 119–129 (1987).
5. B. Buckles and F. Petry. *A fuzzy model for relational databases. Fuzzy Sets and Systems*, 7:213–226 (1985).
6. M. Cayrol, H. Farency, and H. Prade. Fuzzy pattern matching. Kybernetes, 11:103-106 (1982).
7. D. Dubois, H. Prade, and C. Testemale. Weighted fuzzy pattern matching. Fuzzy Sets and Systems, 28:313-331 (1988).
8. R.A. Orchard. FuzzyClips Version 6.04A. User's Guide Integrated Reasoning. Institute for Information Technology. Canada (1998).
9. C. Fellbaum. WordNet: An Electronic Lexical Database. MIT Press (1998).
10. Fontana, F., Formato, F.: *Likelog: A logic programming language for flexible data retrieval.* In: Proc. of the ACM SAC, pp. 260–267 (1999).
11. S. Guadarrama, Muñoz and C. Vaucheret. Fuzzy Prolog: A New Approach Using Soft Constraints Propagation. Fuzzy Sets and Systems, Elsevier, 144(1):127–150 (2004).
12. T. R. Gruber. *Toward principles for the design of ontologies used for knowledge sharing. International Journal Human-Computer Studies*, 43(5-6):907–928 (1995).
13. C.J. Hinde. Fuzzy Prolog. International Journal of Man-Machine Studies, Volumen 24, Issue 6, pp.569–595 (1986).
14. M. Ishizuka and N. Kanai. Prolog-ELF Incorporating Fuzzy Logic. In *Proceedings IJCAI'85. Los Angeles, CA, August 1985.*, pp. 701–703. Morgan Kaufmann (1985).
15. P. Julián-Iranzo. *A procedure for the construction of a similarity relation.* In *Proc. of the 12th IPMU*, pages 489–496. Univ. Málaga (2008).
16. P. Julián and C. Rubio. *A similarity-based WAM for Bousi∼Prolog.* In: LNCS, vol 5517, pp. 245–252. Springer, Heidelberg (2009).
17. P. Julián and C. Rubio. *A Declarative Semantics for Bousi∼Prolog.* In: *Proc. of 11th Intl. Symposium on PPDP'09*. ACM SIGPLAN (2009).
18. P. Julián, C. Rubio and J. Gallardo. *Bousi∼Prolog: a Prolog extension language for flexible query answering.* In: ENTCS, vol 248, pp. 131-147. Elsevier, Amsterdam (2009).
19. P. Julián and C. Rubio. *An Efficient Fuzzy Unification Method and its Implementation into the Bousi∼Prolog System.* In: *Proc. of FUZZ-IEEE*, (2010).
20. R.A. Kowalski. *Algorithm = Logic + Control. Communications of the ACM*, 22(7):424–436 (1979).

21. R.C.T. Lee. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19(1):119–129 (1972).
22. D. Li and D. Liu. *A fuzzy Prolog database system*. John Wiley & Sons, Inc. (1990).
23. V. Loia, S. Senatore, M. I.Sessa, Similarity-based SLD Resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems*, 144(1):151–171 (2004).
24. H.Liu and P. Singh. *Conceptnet – a practical commonsense reasoning tool-kit. BT Technology Journal*, 22(4):211–226 (2004).
25. K. K. Majumdar and D.D. Majumde. *Fuzzy knowledge-based and model-based systems. Journal of Intelligent and Fuzzy Systems*, 18(4):391–403 (2007).
26. A. Motro. *VAGUE: A user interface to relational databases that permits vague queries. ACM Transactions on Office Information Systems*, 6(3):187–214 (1988).
27. E.T. Mueller. Commonsense Reasoning. Morgan Kaufmann (2006).
28. T. Pedersen, S. Patwardhan and Jason Michelizzi. WordNet::Similarity - Measuring the Relatedness of Concepts. In: *Proc. AAAI-04*, pp. 1024-1025 (2004).
29. H. Prade and C. Testemale. *Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. Information Science*, 34:115–143 (1984).
30. F. P. Romero, P. Julián-Iranzo, M. Ferreira-Satler, and J. Gallardo-Casero. Classifying unlabeled short texts using a fuzzy declarative approach. *Language Resources and Evaluation*, 47(1):151-178 (2013).
31. L.G. Rios-Filho and S.A. Sandri. *Contextual Fuzzy Unification.* In: *Proc. of IFSA'95*,pp. 81–84 (1995).
32. M.I. Sessa. *Approximate reasoning by similarity-based sld resolution. Theoretical Computer Science*, 275(1-2):389–426, 2002.
33. R. Shalfield. *LPA-PROLOG: Flint reference.* Technical report, Logic Programming Associates ltd (2005).
34. S. Shenoi and A. Melton. *Proximity relations in the fuzzy relational database model. Fuzzy Sets and Systems*, 100:51–62 (1999).
35. A. Sobrino. The Role of Synonymy and Antonymy in 'Natural' Fuzzy Prolog. In *Studies in Fuzziness and Soft Computing* vol. 273, pp. 209–236 (2012).
36. H.E. Virtanen. Fuzzy Unification. In: *Proceedings IPMU'94, Paris (France)*, pp. 1147–1152 (1994).
37. H.E. Virtanen. *Linguistic Logic Programming. Logic Programming and Soft Computing*, p.p: 91–128 (1998).
38. L.A. Zadeh. *Fuzzy sets. Information and Control*, 8:338–353 (1965).
39. L.A. Zadeh. *The Concept of a Linguistic Variable and its Applications to Approximate Reasoning I, II and III.* J. of Information Sciences 8 and 9, Elsevier (1975).
40. L.A. Zadeh. *Fuzzy Logic and Approximate Reasoning.* Synthese, 30(3–4):407–428, Springer Netherlands (1975).

View publication stats