An Efficient Fuzzy Unification Method and its Implementation into the Bousi∼Prolog System

Pascual Julián-Iranzo and Clemente Rubio-Manzano

***Abstract*— Bousi∼Prolog is a fuzzy logic programming language whose main objective is to make flexible the query answering process. Its operational mechanism is a extension of the SLD-resolution (called *weak resolution*) where the classical syntactic unification algorithm has been replaced by a fuzzy one. This paper presents a generic method for the unification of linguistic terms (i.e. fuzzy sets) which is also applicable to other programming languages with an operational semantics based on some kind of weak resolution mechanism. The basic idea is to compile the information provided by fuzzy sets, generating a binary fuzzy relation on the set of their associated linguistic labels. Subsequently, this fuzzy relation can be used in a standard, completely integrated way inside the unification mechanism of the Bousi∼Prolog system, what allows us to handle linguistic labels on an equal basis with regard other syntactic symbols occurring in the source program. This is a novel approach because it is the first time that fuzzy sets are introduced into the core of a Prolog system by means of compilation techniques and combining fuzzy relations with weak unification. An important feature of this approach is its simplicity, since the inclusion is carried out in a very natural way without affecting the operational semantics of the Bousi∼Prolog language and with very few syntactical modifications. All these reasons convert our approach in a good alternative to the techniques used by other fuzzy Prolog systems.**

I. Introduction

Fuzzy Logic Programming is a research area which investigates how to introduce fuzzy logic concepts into logic programming in order to deal with the uncertainty and/or vagueness existing in the real world explicitly. There are multiple lines of work ricocheting between the ones that modify the classical resolution procedure and replace it by a fuzzy resolution mechanism [15], [17], [24] and those that extend the classical SLD resolution principle, modifying its classical unification algorithm and replacing it by a fuzzy unification algorithm [9], [20], [23]. An example of an intermediate approach is [3], [4] where both the unification algorithm and the resolution principle are modified, combining probabilistic and fuzzy uncertainty into a single framework.

Bousi∼Prolog (BPL, for short) [11], [12], [13] is an extension of the standard Prolog language. Its operational semantics is an adaptation of the SLD resolution principle where classical unification has been replaced by a fuzzy unification algorithm based on proximity relations (i.e. binary fuzzy relations that fulfill the reflexive and symmetric properties), allowing us to generalize proposals such as the ones appeared in [8], [9], [20]. Informally, the *weak unification algorithm* states that two terms $f(t_1, ..., t_n)$ and $g(s_1, ..., s_n)$ weakly unify if the root symbols f and g are approximate and each of their arguments t_i and t_i weakly unify.

Pascual Julián-Iranzo. Department of Information Technologies and Systems, University of Castilla-La Mancha, Paseo de la Universidad, 4. 13071 Ciudad Real, Spain (phone: +34 926 29 53 00; email: Pascual.Julian@uclm.es).

Clemente Rubio-Manzano. Department of Information Technologies and Systems, University of Castilla-La Mancha, Paseo de la Universidad, 4. 13071 Ciudad Real, Spain (phone: +34 926 29 53 00; email: Clemente.Rubio@uclm.es).

At the present time, there are two implementations formats[1]: a high level [11] and a low level implementation [12], [14]. The high level implementation is written in Prolog through a meta-interpreter and it is used for prototyping new features of the BPL language. The low level implementation, written in Java, consists of an architecture made up of a compiler and an enlargement of the Warren Abstract Machine [1], [25] which is able to handle fuzzy relations and to execute BPL programs efficiently.

The following example serves to illustrate both the syntax and the operational semantics of the language.

Example 1: Assume a fragment of a database that stores information about films and user preferences encoded by means of *proximity equations*.

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

%% FACTS
film(the_lord_of_the_rings,adventures).
film(terminator, action).
film(stargate, science_fiction).

%% PROXIMITY EQUARIONS
adventures~action=0.9.
adventures~science_fiction=0.8.
```

In a standard Prolog system, if we ask about whether `stargate` is an action film, "`?-film(stargate, action)`", the system fails. However Bousi∼Prolog allows us to obtain the answer "`Yes with 0.8`" since the goal weakly unifies with the fact "`film(stargate, science_fiction)`". This is possible because the `transitivity` directive activates, at compile time, the automatic generation of the transitive closure of the fuzzy relation defined by the programmer by means of the proximity equations (introduced in the source program). Hence, the entry "`action~science_fiction = 0.8`" is obtained and, lately, it is used, at run time, during the weak unification process.

This paper is motivated by the necessity of improving the current resouces that the BPL system have for the management of vagueness and approximate reasoning. To

[1]Both of them are publicly available and can be found at the URL address: `http://dectau.uclm.es/bousi.html` .

achieve this objective, we aim to incorporate the domain (data type) fuzzy set into the core of the Bousi∼Prolog system. Enhancing Bousi∼Prolog with this feature extends its application area: fuzzy control, fuzzy databases, approximate reasoning and, in general, whatever application where the management of vagueness using the joint power of fuzzy sets and declarative languages is necessary. On this last respect, in recent years, a new computational paradigm which combines linguistic terms and logic programming is arising [23]. The inclusion of fuzzy sets places Bousi∼Prolog in a central position among the theoretical and practical realizations of this new paradigm (as we shall discuss later). Summarizing, the aim is strengthen Bousi∼Prolog as an authentic fuzzy logic programming language useful for *Soft computing*.

The outline of the paper is as follows. In Section II, we introduce the weak unification algorithm, which is syntactical in nature and based on fuzzy relations. Then, to carry out the main objective of this work, we start from the formal concept of linguistic variable [27] and we develop, in Section III, what can be seen as a new method for the unification of linguistic terms. A *linguistic variable* is a structure composed of a set of linguistic terms (also called linguistic labels) whose semantic component are fuzzy subsets (linked to each linguistic term). Therefore, we need to manage the semantics of linguistic terms. The developed method consists of transforming, at compile time, the information stored by the fuzzy subsets in a fuzzy relation on the set of the linguistic labels. Hence, the weak unification algorithm can handle the linguistic labels in an analogous way as the rest of syntactic symbols of the language. We observe that this approach is useful for those fuzzy logic programming languages like LIKELOG [7] or SiLog [16] which also use an operational mechanism based on weak SLD resolution. In Section IV we detail how to incorporate this generic approach into the core of the (low level implementation) of the BPL system. We specify a couple of directives to declare fuzzy sets, a grammar to generate linguistic terms and the distinct phases of their implementation. Section V relates our proposal with others existing in the literature and discusses some of its main features. Section VI shows the usefulness of introducing linguistic variables in the core of Bousi∼Prolog by discussing a fuzzy control application where a steam turbine is modeled. This example reveals the expressive power of the BPL language to solve these kind of problems. Finally, in Section VII we give our conclusions and some lines of future research.

II. Fuzzy Relations and Weak Unification

The concept of a fuzzy relation was introduced by Zadeh in [26]. A *binary fuzzy relation* on a set U 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 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. Bousi∼Prolog uses *mim*-transitivity when the transitivity flag is enabled.

The *weak unification algorithm* used by Bousi∼Prolog is an extension of the one appeared in [20], 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 [2], 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., $G_n = Fail$).

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:

- Term decomposition:

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

- 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[3], 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. Next sections are examples of the usefulness of following this path.

III. A Generic Approach for Semantic Unification

The weak unification algorithm described in the last section 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 propose an efficient generic approach to include fuzzy sets and semantic unification in the framework of fuzzy logic programming. Our approach differs

[2]Here, the symbol "$\approx$" represents that the arguments in $\mathcal{E}_1$ and $\mathcal{E}_2$ are capable to be equals by proximity.

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

from others because it compiles the information provided by a linguistic variable into a set of fuzzy relations. Therefore, this approach is suitable for those programming languages with an operational mechanism based on some kind of weak resolution supporting a syntactic unification algorithm guided by fuzzy relations, as is the case of Bousi∼Prolog or, in some respect, of LIKELOG [7] and SiLog [16].

Formally, our approach relies on the concepts of a linguistic variable [27] and a fuzzy relation [26].

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—).

Regarding the concept of linguistic variable, the following remarks are convenient. $T(X)$ is the syntactic component of X, its elements are *linguistic terms*. Linguistic terms are generated by means of the grammar G (for the moment, we let it unspecified). It is usual to make the distinction between *atomic terms* (also called, *primary terms*) and *composite terms* which are composed of primary terms. The domain U is an ordinary set (not necessarily numerical). The semantic rule M associates meaning to composite terms by means of precise computations starting from the meaning of their atomic terms. The meaning of primary terms is defined axiomatically, assigning to each atomic term a fuzzy subset on U. In other words, the fuzzy subsets that M applies to composite terms are calculated, while the ones applied to primary terms are defined (in a subjective and context-dependent way).

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 proceeds 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 [5], [6], which have been successfully used in the system *FuzzyClips* [10].

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 the 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 section, it is important to note that, if we want to cope with the distinction between general and specific knowledge introduced in [19], the fuzzy relation constructed during this process must fulfill the reflexive property but not necessarily the symmetric and/or transitive properties. In [19], and lately in [2], 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 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 [23], but also it takes into account the distinction between general and specific knowledge made in [19].

IV. Fuzzy Sets in the Bousi∼Prolog Language

In this section, we explain at length the implementation of a linguistic variable structure into the Bousi∼Prolog language. We begin with the syntactic aspects. Then, we describe the different implementation phases and data structures which are generated to facilitate its implementation and later execution.

A. Sintax

All sorts of programming languages must provide specific instructions to declare and define their data structures. The BPL language makes use of two directives to declare and define the structure of a linguistic variable X. In fact, only its semantic component is defined: the domain or universe of discourse U and the fuzzy subsets which are associated to primary linguistic terms in $T(X)$ (the rest, as we shall see, is calculated automatically). On the other hand, we shall not make a lexical distinction between the syntactic and the semantic component of X. That is, we denote with the same name both the linguistic variables and their domains. We proceed with the linguistic terms and their values similarly, employing the same symbol to designate them. Hence, we shall rely on the context for disambiguation.

The domain directive: It allows to declare and define the universe of discourse or domain associated to a linguistic variable. In general, we use the same name both for the linguistic variable and the domain. For practical reasons and because it is not a significant limitation, in this implementation, we only deal with domains of the real interval. 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.

Example 2: The following directive defines a domain with name `age`, whose values are numbers (between 0 and 100) measured in `years`: "`:-domain(age(0,100,years))`".

The fuzzy_set directive: It 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,
    [Set1(a1,b1,c1[,d1]), ..., SetN(aN,bN,cN[,dN]]).
```

Fuzzy subsets are defined by indicating their name, `Set_i`, 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 an efficient information representation and computation).

Example 3: The following directive defines a list of fuzzy subsets, whose universe of discourse is the domain `age` defined in Example 2.

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

In particular, it defines three fuzzy subsets associated to the primary terms `young`, `middle` and `old` whose membership functions are trapezoidal functions characterized by their arguments.

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
```

Example 4: For the linguistic variable `Age` defined in the examples 2 and 3, some of the composite terms that may be generated from primary terms are: `very#young`, `more_or_less#middle` or `extremely#old`

Domain points and domain ranges: 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. We advance that, domain points have a different behavior (meaning) depending on either they are representing specific knowledge or general knowledge. 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 of the following grammar.

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

where `<Dom_Val>` 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`".

TABLE I
TABLE OF LINGUISTIC TERMS: MEMORY REPRESENTATION FOR A FRAGMENT CONCERNING THE VARIABLE Dom.

Term	Domain	Membership Function			
$LingTerm_1$	Dom	a_1	b_1	c_1	d_1
$LingTerm_2$	Dom	a_2	b_2	c_2	d_2
...					
$LingTerm_n$	Dom	a_n	b_n	c_n	d_n

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_Val>#<Dom_Val>
<Composite_DR> ::= <RModif>#<Dom_Range>
<RModif> ::= about
```

As in the previous case, a domain range can be fuzzified adding the modifier `about`.

Example 5: Continuing with Example 2 and the linguistic variable `Age`. Examples of domain points are: `age#22` and `about#age#22`. Examples of domain ranges are: `age#20#30` and `about#age#20#30`.

B. Compiling fuzzy sets

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 section the different compilation phases of the data type fuzzy set are detailed.

Syntactical Analysis: During the syntactical analysis 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[4] 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 $f(x, a, b, c) = max(min((x-a)/(b-a), (c-x)/(c-b)), 0)$. Conceptually, this process may be understood as one that builds a table of linguistic terms along with their meanings, as Table I illustrates.

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

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

TABLE II

MEANING OF COMPOSITE LINGUISTIC TERMS.

Modifier	Description	Modifier	Description
very(y)	y^2	more_or_less(y)	$\sqrt{y}$
somewhat(y)	$y^{0.333}$	extremely(y)	y^3

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 (see Table II).

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 IV-A, 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. We coded this fact introducing a null entry (symbolized as "$\perp$" in Table III) into the membership function field of a linguistic terms table. This design decision will require us a special treatment of such kind 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.

Ending this subsection, 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.

Example 6: Continuing with Example 2, regarding the linguistic variable `Age`, assume a fragment of a database that stores information about people:

```
person(john,young).       person(paul,about#age#40).
person(mary,age#35).      person(warren,very#young).
```

For the above program, the result of this compilation phase is shown in Table III.

Generation of fuzzy relations: Once the table of linguistic terms has been built, the next phase is focused on generating fuzzy relations between linguistic terms in order to compile all the semantic information associated with them. The generation phase is implemented by means of the following algorithm, which takes into account the distinction between general and specific knowledge. (for the sake of simplicity, we focus our attention on linguistic terms associated to one single linguistic variable).

TABLE III

LINGUISTIC TERMS TABLE, GENERATED AFTER THE SYNTACTICAL ANALYSIS, FOR THE FRAGMENT PROGRAM OF EXAMPLE 6.

Term	Domain	membership Function
young	*age*	μ_{young}
middle	*age*	μ_{middle}
old	*age*	μ_{old}
35	*age*	$\perp$
about_40	*age*	μ_{about_40}
very_young	*age*	μ_{very_young}

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) $\mu_{x_i} \neq \perp$ and $\mu_{x_j} \neq \perp$:
         $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(x_i, x_j) = match(\mu_{x_j}, \mu_{x_i})\}$;
      2) $\mu_{x_i} \neq \perp$ and $\mu_{x_j} = \perp$:
         Let $x_j = dom\#u_j$ in $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(x_i, x_j) = \mu_{x_i}(u_j)\}$;
      3) $\mu_{x_i} = \perp$ and $\mu_{x_j} \neq \perp$:
         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 $\mathcal{R}$
```

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 [19], 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 [5], [6], 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. The following examples clarify this point.

Example 7: Assume a simple program containing the only one fact `person(mary, age#28)`. If we launch the goal `?- person(mary, young)`, an admissible answer may be `yes with degree` α where $\alpha = \mu_{young}(28) = 1.0$. The reason is that Mary is a young person because she is 28 years old and the youth degree must be the membership degree of `28` in the fuzzy set `young`. The weak SLD resolution principle is able to give this answer if the original program is completed with the entry $\mathcal{R}(young, age\#28) = \mu_{young}(28)$.

Example 8: Now assume a simple program containing the only one fact `person(mary, young)`. If we launch the goal `?- person(mary, age#28)`, we should expect a positive answer but with a degree β lower than the one obtained in Example 7. This lower degree is due to the fact that in this example the specific knowledge is that "Mary is young" and the general knowledge is that "Mary is 28 years old", opposite to the situation reflected by Example 7. Although a 28 years old person must be considered as young, the specific knowledge "young" involves other age values and we do not know the precise age of Mary. Certainly, Mary's age might be "20 years old" or other age instead of "28 years old". In order to preserve the constrain that the degree β is lower than $\mu_{young}(28)$, the BPL system follows this procedure: the domain point `age#28` is fuzzified, converting it into a singleton fuzzy set characterized by a trapezoidal function $f(x, 28, 28, 28, 28)$ (i.e., $\mu_{age\#28}(x) = 1$ if $x = 28$ and $\mu_{age\#28}(x) = 0$ otherwise); then, the entry $\mathcal{R}(age\#28, young) = matching(\mu_{young}, \mu_{age\#28}) = 0.5$ is built. We have verified analyzing several examples that the matching function behaves well in these cases.

These two last examples help to understand the decisions taken in the specification of cases 2 and 3 in Algorithm 1.

The following example shows the effect of applying Algorithm 1 on a set of linguistic terms.

Example 9: Starting from data contained in Table III, Algorithm 1 generates the entries defining a fuzzy relation. A fragment of these entries are shown in Table IV, where the term `Term_1` represents the general knowledge and `Term_2` the specific knowledge.

Once the compilation phase is concluded, all the meaningful 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.

Example 10: Continuing with Example 2, related to the linguistic variable `Age` and taking the clauses of Example 6. If we ask about what people are young, "`?-person(X,young).`", the BPL system will answer:

TABLE IV
FUZZY RELATION: ENTRIES FOR THE TERM `young`.

Term_1	Term_2	Relationship degree
young	*young*	1.0
young	*middle*	0.38
young	*old*	0
young	*age*#35	0.75
young	*about*#*age*#40	0.52
young	*very*#*young*	1.0

```
X=john with 1.0 ;          X=paul with 0.52 ;
X=mary with 0.75 ;         X=warren with 1.0
```

V. Discussion and Related Work

The method for fuzzy unification we have just presented in this paper is a semantic one, that is, it is centered in the unification of linguistic terms, but it is supported by a weak unification algorithm (see Section II) which is syntactic in nature. In this section we discuss this two-faced reality.

Weak unification emerges from the theoretical works of Gerla et al. [8], [9], where the classical syntactic unification algorithm is extended with the ability of dealing with similarity relations. The main practical realization of this research line is the fuzzy logic language LIKELOG [7], with a fuzzy resolution rule relying in the replacement of the classical syntactic unification algorithm by the new similarity-based unification algorithm. LIKELOG is an interpreter implemented in Prolog using rather direct techniques and cumbersome concepts. A second line of research, which is the closest to ours, is based on [20], where the concepts of weak unification and weak SLD resolution were developed. In [16], a similarity-based logic programming language, named SiLog, was presented. SiLog is an interpreter written in Java. Our contribution to this line of work has been to pay attention of the importance of overcoming the knowledge representation limitations associated to similarity relations, extending the operational mechanism with proximity relations (more appropriated to this end) and proposing a completely new declarative semantics view for a framework combining logic programming and proximity relations [13]. Summarizing, we can enumerate some additional features that distinguish Bousi∼Prolog from these other proposals: it gives automatic support for the user when computing closures of the fuzzy relation specified in a BPL program; it provides a more flexible operational mechanism and additional expressive power; it is a true extension of Prolog with an efficient implementation[5] [12]. These and other related features were largely discussed in [12] and [13]. We direct the interested reader to those papers.

At the best of our knowledge, the inclusion of linguistic terms into the field of logic programming was first suggested in [5], where some techniques for solving the problem of

[5]Neither LIKELOG nor SiLog are publicly available and therefore a practical comparison is impossible. However, we supply an implementation based on an enhancement of WAM that should be more efficient than an interpreter.

matching fuzzy constants were introduced. Lately, in [3] a fuzzy pattern matching algorithm based on Baldwin's mass assignment theory was named "*semantic unification*". Also in [6] 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* [10] and we adopted it for computing some relationship degrees when generating fuzzy relations in Algorithm 1.

In later years, Virtanen [22], [23] 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 is 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 [19] first introduce the distinction between general and specific knowledge, giving raise to what they call a *contextual fuzzy unification* algorithm. Contrary to [3], [22] or [2] their algorithm 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; or *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 an unification degree.

The work [2] 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:

• 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.

• 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.

• These algorithms do not work with function symbols and variables only can be bound to linguistic 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 [23] 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 and 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$\sim$Prolog has greater expressivity and can deal with approximate reasoning easily.

Example 11: 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`).

VI. A Fuzzy Control Application

In this section we implement a fuzzy logic controller which is an adaptation of the one described in [21]. 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.

Now we can proceed to define the rules. 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).
```

We can define the rest of rules, in a similar 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:

```
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` [6]: when the goal "`?-defuzzify(throttle(_),Y).`?" is launched, the answer "`Y = throttle#-14.09`" is obtained.

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 [7]. Hence, the main difference between a BPL program and a Prolog program is in the inside, not in the surface.

VII. Conclusions and Future Work

Bousi∼Prolog is a fuzzy logic programming whose aim is the management of vagueness and incomplete information using declarative techniques. In this paper has been detailed how to integrate fuzzy sets into the BPL system. The general idea consists of defining a fuzzy relation on the set of terms, $T(X)$, associated to a linguistic variable, X, in such a way that this (included its semantic component —i.e, the fuzzy subsets—) could be treated at a purely syntactic level in the context of a logic program. This implementation technique is simple, elegant and efficient, allowing the use of fuzzy sets in a very natural way (in the context of those languages whose operational semantics is based on some kind of weak resolution mechanism). On the other hand, this approach is original since it is the first time that fuzzy sets are introduced into the core of a Prolog system by compiling them into fuzzy relations (which serve as a basis for executing a weak unification algorithm).

As a mater of future work, on the theoretical side, we have to investigate the formal properties of the resulting algorithm. On the practical side, 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 complementing the BPL system in order to make it a suitable tool for *Soft Computing*. To this end, we want to enhance the BPL operational mechanism allowing the use of several continuous t-norms, other than the minimum. Also, we are working for adding to the system tools for managing term ontologies modelled by proximity equations.

Acknowledges: This work has been partially supported by FEDER and the Spanish Science and Innovation Ministry under grant TIN 2007-65749 and by the Castilla-La Mancha Regional Administration under grant PII1I09-0117-4481.

References

[1] H. At-Kaci. Warren's Abstract Machine: A Tutorial Reconstruction. *The MIT Press*, Cambridge, MA (1991).

[2] T. Alsinet and L. Godo. Fuzzy Unification Degree. In *Proc. of the 2nd Intl. Workshop on Logic Programming and Soft Computing'98, Manchester (UK)*, pp 18 (1998).

[3] J.F. Baldwin Evidential support logic programming. Fuzzy Sets and Systems, 24, pp 1–26 (1987).

[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] M. Cayrol, H. Farency, and H. Prade. Fuzzy pattern matching. Kybernetes, 11:103-106 (1982).

[6] D. Dubois, H. Prade, and C. Testemale. Weighted fuzzy pattern matching. Fuzzy Sets and Systems, 28:313-331 (1988).

[7] F. A. Fontana and F. Formato. Likelog: A logic programming language for flexible data retrieval. In *Proc. of the ACM SAC*, pp. 260–267, 1999.

[8] F.A Fontana and F. Formato. A similarity-based resolution rule. *Int. J. Intell. Syst.*, 17(9):853–872 (2002).

[9] F. Formato, G. Gerla, and M.I. Sessa. Similarity-based unification. *Fundam. Inform.*, 41(4):393–414 (2000).

[10] R.A. Orchard. FuzzyClips Version 6.04A. User's Guide Integrated Reasoning. Institute for Information Technology. Canada (1998).

[11] 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).

[12] P. Julián and C. Rubio. A similarity-based WAM for Bousi∼Prolog. In: LNCS, vol 5517, pp. 245–252. Springer, Heidelberg (2009).

[13] P. Julián and C. Rubio. A Declarative Semantics for Bousi∼Prolog. In: *Proc. of 11th Intl. Symposium on PPDP'09*. ACM SIGPLAN (2009).

[14] P. Julián and C. Rubio. UNICORN: A Programming Environment for Bousi∼Prolog. In: *Proc. of PROLE'09*. (2009).

[15] R.C.T. Lee. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19(1):119–129 (1972).

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

[17] M. Mukaidono, Z.L. Shen and L. Ding. Fundamentals of fuzzy Prolog. Intl. J Approximate Reasons, 3, pp. 1080–1086 (1989).

[18] H.T. Nguyen and E.A. Walker. *A First Course in Fuzzy Logic*. Chapman & Hall/CRC, Boca Ratón, Florida, 2000.

[19] L.G. Rios-Filho and S.A. Sandri. Contextual Fuzzy Unification. In: *Proc. of IFSA'95*,pp. 81–84 (1995).

[20] M.I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2): 389–426, 2002.

[21] R. Shalfield. LPA-PROLOG: Flint Reference. Logic Programming Associates ltd. (2005).

[22] H.E. Virtanen. Fuzzy Unification. In: *Proc. of IPMU'94, Paris (France)*, pp. 1147–1152 (1994).

[23] H.E. Virtanen. Linguistic Logic Programming. In *Proc. of the 2nd Intl. Workshop on Logic Programming and Soft Computing'98*, pp. 91–110, Wiley (1998).

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

[25] David H. D. Warren. An Abstract Prolog Instruction Set. Technical note 309, SRI International, Menlo Park, CA., October,1983.

[26] L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965.

[27] 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).

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

[7] We suggest to compare our solution with the one appeared in [21].