2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Reasoning with Words: A First Approximation

Clemente Rubio-Manzano, Pascual Julián-Iranzo

***Abstract*— This paper aims to propose a model of reasoning based on semantic relations among words and to incorporate it in the inference mechanism of a logic programming language. This model is integrated in a fuzzy logic programming framework and it is implemented into the Bousi∼Prolog system by using WordNet. All this process is transparent to the programmer and the reasoning with words is automatic. The lexical semantics between symbols (words) provides us with the ability of reasoning with words and turns the knowledge representation in a more natural and less complex process.**

I. Introduction

When symbols of a formal language are interpreted as words, then it is possible to establish a direct relationship between formal language and natural language, in the sense that the symbols gain the meaning of the words which they represent. As a consequence the properties or phenomena produced among some of them are extrapolated on the symbols (semantic relations, vagueness, impression or ambiguity, are some examples). In this context, classical logic does not enables to work in a natural way with these phenomena or properties, mainly due to its rigid inference mechanism. Therefore, such mechanism must be extended and adapted to new requirements. The following example shows this fact.

Example 1: Let's suppose that we want to formalize a common sense reasoning rule: *"if you are at your car and you drive from home to airport, then you are at the airport"*. A possible formalization, leaving temporal considerations aside, would be the following:

$$at(I, car) \wedge driving(I, home, airport) \rightarrow at(I, airport).$$

In order to make this example to work correctly with the properties and phenomena mentioned before, a great amount of additional elements are required; for example, defining precisely the terms in the sentence (car, driving, home, airport) and its interrelations, which makes the process of representing knowledge a very complex task when the objective is obtaining a complete commonsense reasoning.

On the other hand, while logic reasoning is deductive, human reasoning is mainly inductive, abductive and empirical; also, concepts, generalizations, categories based on experience and perceptions play a central role. Logic has an expressive language in which it is not easy to model and where simulation of human reasoning, which is mainly inductive and highly associative, it is a complex process.

Such considerations lead us to consider the use of semantic relations in a logic programming framework with the purpose of improving its representation and inference mechanisms. Also, quantification of semantic similarity between symbols provide us with the ability of reasoning regarding concepts and turns the knowledge representation into a more natural and less complex process. While logic is a synthetic representation by creating and manipulating symbols in a closed world, logic enriched with the semantic relations of the natural language is an open, empirical representation, combining concepts that have already been definitive and related to the human language [6].

In this work we present a first approximation to solve this kind of problems. It consists of designing a model of reasoning based on semantic relations among words and to incorporate it in the inference mechanism of a fuzzy logic programming language. In this new framework, semantic relations are handled by means of fuzzy relations which are used within a weak unification process. This way we obtain a more natural fuzzy logic programming language. Specifically, this model is implemented into the Bousi∼Prolog by using WordNet.

The structure of the paper is as follows. First, Section II introduces the general concepts regarding natural languages and thesauri. Note that we are interested in the lexical semantics and the thesauri as a basis for knowledge; so, we only review a little part of the natural language theory. Then, in the Section III we propose a model of reasoning based on the semantic relations among words by means of the definition of inference rules based on synonymy, antonymy and hypernymy which are incorporated to a fuzzy logic programming framework. From a more pragmatic stand point, Section IV shows the incorporation of WordNet into the Bousi∼Prolog system and the implementation of this model of reasoning. Next, Section V shows the potential problems that we can find reasoning with words. Finally, the conclusions and the future work are included in Section VII.

II. Natural Language, Thesauri and WordNet

Although natural languages are composed of a number of components (such as phonetic, morphology, pragmatics and so on) here we concentrated on syntax and semantics. Syntax establishes the rules and principles by which words and sentences are constructed. Semantics studies the meaning of these constructions at two levels, lexical semantics (word meaning) and sentence meaning. In this first approximation we pay attention to the lexical semantics.

An important field in lexical semantics is the study of the existing relations of words, since the meaning of a word depends largely on its semantic relations with other words. The main relations are: **Synonymy**, which relates words with similar meanings in a determined context. Thus, two

Clemente Rubio-Manzano with the Department of Information Systems, University of the Bío-Bío, Chile (email: clrubio@ubiobio.cl).
Pascual Julián-Iranzo. Department of Information Technologies and Systems, University of Castilla-La Mancha, Spain (email: Pascual.Julian@uclm.es).

978-1-4799-2072-3/14/$31.00 ©2014 IEEE

words are synonyms when they have the same meaning or a similar meaning. Synonyms may appear in nouns (baby-infant, student-pupil, illness-disease), verbs (grow-increase, buy-purchase), adjectives (pretty-attractive, sick-ill, rich-wealthy) or adverbs (fast-quickly, really-certainly, currently-presently); **Antonymy**, are pairs of words whose meaning is in opposition, such as hot-cold, fat-thin, up-down. Words could have different antonyms, depending on their meaning, for example, long and tall are antonyms of short. There are three types of antonyms. *Gradable antonyms* represent points on a scale that are approximately equal in distance (slightly-completely, hot-cold, happy-sad or early-late); *Complementary antonyms* are pairs of words that express a totally opposite meaning (mortal-immortal, on-off, alive-dead); and *Relational Antonyms* (or *Converse Antonyms*) which are pairs of antonyms in which one word describes a relation between two objects and the others describe the inverse relation (parent-child, teacher-student, buy-sell); (c) **Hypernymy**, a word A is an hypernym of another word B if B is a class of A. It refers to categories or general concepts. For example, "vehicle" is hypernym of "train, car, airplane, bicycle", "animal" is an hypernym that included in its meaning "dog, cat, horse, lion"; (d) **Hyponymy** is the opposite of hypernym. And hyponym is a word whose meaning is included in that of another word, its hypernym. And hyponym describes things in a specific way. For example, rose, daisy or tulip are hyponyms in relation of its hypernym flower; or table, sofa or bed are hyponyms which hypernym is furniture. Besides those relations mentioned before, there are other semantic relations: Polysemy, Homonymy or Meronomy.

A way to obtain the semantic relations among the words, in a particular context, is by using a thesaurus. A thesaurus is a vocabulary (terms and descriptions) with links towards synonyms, equivalents, generics, specific or related terms. A thesaurus can be general and come from a natural language (WordNet [7] o ConceptNet [6], in English, o FDSA [9], in Spanish) or specialized and come from mastering a specialized subject (EuroVoc [1] or UMLS [5]). We are interested in digital thesauri that can be accessed by using a standard programming language. In particular, we have chosen WordNet, though the techniques that we present and explain here can be extrapolated to any other context and any other thesaurus.

We are going to use WordNet in order to obtain semantic relations. In particular, synonyms, antonyms and hypernyms. WordNet is a lexical database of English language where words (separated by noun, verbs, adjectives and adverbs) are linked through semantic relations. The semantic relations that handles were chosen because they are recurrent in English and due to its familiarity. That is to say, a user does not have to be an expert to understand them. Wordnet includes the following semantic relations [7]: (a) synonymy is the basic relation in WordNet since it uses sets of synonyms (synsets) to represent the meaning of words. Synonymy is a symmetric relation between words; (b) Antonymy (opposite name) is also a symmetric relation between words, it is especially important in order to organize the meaning of adjectives and adverbs; (c) Hyponymy (sub-name) and its inverse relation, hypernymy (super-name), are transitive relations between synsets. Since there is only one hypernym, this semantic relation organizes the meaning of names as a hierarchy; (d) Meronomy (part-name) and its inverse, Holonomy (whole-name), are complex semantic relations. WordNet distinguishes component parts, substantive parts, and member parts; (e) Troponymy (manner-name) is for verbs what hyponymy is for nouns, although the resulting hierarchies are much shallower; (f) Entailment relations between verbs are also coded in WordNet.

III. Reasoning with Words

In this section we are going to present a model of reasoning based on the semantic relations among words and we will include it in the weak resolution procedure of a fuzzy logic programming language such as Bousi∼Prolog [4]. This inference mechanism makes use of two important concepts: proximity equations and weak unification. Proximity equations allow us to define similarity or proximity relations and, in general, fuzzy relations. From a syntactic point of view, a proximity equation "$a \sim b = \alpha$" defines an entry $\mathcal{R}(a, b) = \alpha$ of a relation $\mathcal{R}$ establishing that a is close to b with a degree $\alpha \in [0, 1]$. Roughly speaking, the so called *Weak Unification Algorithm* states that two terms $f(t_1, \ldots, t_n)$ and $g(s_1, \ldots, s_n)$ weakly unify if the root symbols f and g are close, with a certain degree, and each of their arguments t_i and s_i weakly unify. The following example serves to illustrate the syntax and the operational semantics of the language.

Example 2: Assume a fragment of a deductive database that stores information about people and their preferences on teaching.

```
% PROXIMITY EQUATIONS
physics ~ math = 0.8.
physics ~ chemistry = 0.8.
chemistry ~ math = 0.6.

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

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

In a standard Prolog system, if we ask about who can teach mathematics, ``?-can_teach(X,math)'', the system does not produce any answer. However the Bousi∼Prolog system answers ``X=john with 0.8'' and ``X=mary with 0.6''. Since we have specified that `physics` is close to `math`, with degree 0.8, these two terms may unify "weakly" with approximation degree 0.8, leading to a refutation. Similarly for the terms `physics` and `chemistry`.

As it has been shown in the example, a fuzzy logic programming based on weak unification enables to establish and

quantify the relation between symbols. On the other hand, when symbols are interpreted as words it will be possible to model the semantic relations between them making use of a thesaurus $\mathcal{T}$. A word $w_1 \in \mathcal{T}$ is related to a word $w_2 \in \mathcal{T}$ with a degree α through a semantic relation (synonymy, antonymy or hypernymy) which is obtained from $\mathcal{T}$, what it is represented in Bousi∼Prolog by using a proximity equation "$w_1 \sim w_2 = \alpha$". So, words and their relations become part of the first order language alphabet and take part in the inference process naturally. This requires an inference rule based on the semantic relation that has been used to obtain the set of proximity equations between words. More formally, we will call *program*, Π, a set of first order horn clauses in which there is a set of symbols that we interpret as words. This program has an associated dictionary that we will call vocabulary of Π and that are composed of all those symbols (predicate, function or constant) present in the program and that are defined in a thesaurus $\mathcal{T}$.

Definition 1 (Vocabulary of a program Π*):* Given a program Π. The vocabulary of Π, denoted by $\mathcal{V}_\Pi$, is made of all the predicate, function or constant symbols in the program.

Definition 2 (Vocabulary of Π *and* $\mathcal{T}$*):* Given a program Π and a thesaurus $\mathcal{T}$. The vocabulary of Π and $\mathcal{T}$, denoted by $\mathcal{V}_\Pi^\mathcal{T}$, is made up of all the predicate, function or constant symbols in the program Π that are words in $\mathcal{T}$. That is, symbols which are interpreted as words in a thesaurus $\mathcal{T}$.

Example 3: Let Π be the program of Example 1 and a thesaurus $\mathcal{T}$ of the English language as WordNet. The vocabulary $\mathcal{V}_\Pi^\mathcal{T} = \{at, car, home, airport, driving\}$.

Now, each of the elements in $\mathcal{V}_\Pi^\mathcal{T}$, will be used by the thesaurus $\mathcal{T}$ in order to obtain the semantic relations: synonyms, antonyms and hypernyms. To establish the reasoning model, we have to define with precision the inference rules based on the semantic relations among words.

A. Reasoning with Synonymy

Reasoning with synonymy consists of using this semantic relation to infer. It is required to obtain inference rules based on this property. The deductive calculus based on synonymy has two rules. A direct inference rule that established that if I have A then I can infer B as long as A and B are synonyms; and an adaptation of the modus ponens rule that establishes that if I have A and B $\rightarrow$ C then I can infer C as long as A and B are synonyms. This is formally stated in the following definitions:

Definition 3 (Synonymy-based Direct Inference Rule): The synonymy-based direct inference rule states that from A it can be inferred as immediate consequence B, if A is a synonym of B.

Definition 4 (Synonymy-based Modus Ponens): The Modus ponens rule based on Synonymy states that from A and $B \rightarrow C$ it can be inferred as immediate consequence C if A is a synonym of B.

In order to simulate this reasoning, in our framework we have to compute the synonyms for each of the elements of the vocabulary of a program Π. Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $cV_\Pi^\mathcal{T}$, then for all $w \in \mathcal{V}_\Pi^\mathcal{T}$, the set of synonyms of w extracted from $\mathcal{T}$ is denoted as $\mathcal{S}(w)_{\mathcal{V}_\Pi^\mathcal{T}} = \{s_1, \ldots, s_n\}$.

Example 4: Given the vocabulary of Example 3, for the thesaurus WordNet, it is possible to obtain a set of synonyms for each element of the vocabulary: $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(at) = \{helium, \ldots\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(car) = \{sedan, bike, motorcar, \ldots\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(home) = \{home\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(airport) = \{airdrome, heliport, aerodrome, \ldots\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(driving) = \{dynamic, impulsive, energetic, \ldots\}$.

Once the vocabulary of synonyms has been constructed, the proximity equations are created (synonym equations in this case) from the vocabulary of the program and its set of associated synonyms.

Definition 5 (Synonymy equations for Π*):* Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $\mathcal{V}_\Pi^\mathcal{T}$. Then for all $w \in \mathcal{V}_\Pi^\mathcal{T}$ and its synonyms $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(w) = \{s_1, \ldots, s_n\}$, an entry $w \sim s_i = \alpha_i$ is created for each synonym s_i, with approximation degree α_i.

Example 5: Given a program $\Pi = \{loves(a, b)\}$, the thesaurus $\mathcal{T} = WordNet$ and the vocabulary of the program and the thesaurus $\mathcal{V}_\Pi^\mathcal{T} = \{loves\}$, we obtain the synonyms of the vocabulary, that is, $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(loves) = \{dotes\_on\}$. Then the synonym equation "$loves \sim dotes\_on = 1.0$" is created. Therefore, it could be asked if `a` dotes on `b`, launching the query "`?- dotes_on(a,b)`". Then, the system would respond "`Yes`", since `a` loves `b` and, by deduction based on synonymy, it also dotes on `b`.

B. Reasoning with Antonymy

Reasoning with antonymy consists of using this relation to obtain a type of reasoning based on this semantic relation. Following [11], antonymy is a phenomenon of the natural language which involves a pair of words (P, Q) with some important features [1]: (a) Having the pair (P, Q) then P is an antonym for Q and Q is an antonym for P, i.e. $P = aQ$ and $Q = aP$; (b) Every object x fulfills that "If x is aP then it is not the case that x is P", but it does not fulfill that "If x is not P then x is aP". For example, "the bottle is empty" means that it is not full, but "the bottle is not full" does not mean that it is empty, i.e. aP implies not P, but not P does not imply aP (with some exceptions).

We are going to handle antonymy as synonymy of opposite concepts. Consequently, the Synonymy-based Inference Rules are used but taking into account, in this case, that if P is an antonym of Q we have that P is a synonym of aQ, i.e. Q is the opposite of P. On the other hand, and from a syntactical point of view, it will be necessary to use a modifier of antonymy "not#" to provide information regarding the concepts that are antonyms. So, for example, if "cold" is antonym of "hot" then "cold" is synonym of

[1]For simplicity we only enunciate two of them, the necessary ones in order to define the inference rules based on this semantic relation, see [11] for more information.

"not#hot". Note that the modifier "not#hot" must not be confused with the logical operator of negation and it only provides information on the words (concepts) which are antonymous.

Now we proceed in a similar way as for synonymy by estimating, in this case, the antonyms for each of the elements that are in $\mathcal{V}_{\Pi}^{\mathcal{T}}$. Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $\mathcal{V}_{\Pi}^{\mathcal{T}}$, then for all $w \equiv (not\#u) \in \mathcal{V}_{\Pi}^{\mathcal{T}}$, $\mathcal{A}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(u) = \{a_1, \ldots, a_n\}$ denotes the set of antonyms of u extracted from $\mathcal{T}$. Once the vocabulary of antonyms has been constructed, the proximity equations are created (antonym equations in this case) from the vocabulary of the program and its set of associated antonyms.

Definition 6 (Antonymy equations for Π*):* Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $\mathcal{V}_{\Pi}^{\mathcal{T}}$. Then for all $w \equiv not\#u$ with $w \in \Pi$, and its antonyms $\mathcal{A}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(u) = \{a_1, \ldots, a_n\}$, an entry $not\#u \sim a_i = \alpha_i$ is created for each antonym a_i, with approximation degree α_i.

Example 6: Given a program Π = { bottle(coke,full). ; drink(X) ← bottle(X,not#empty). } and the thesaurus $\mathcal{T} = WordNet$, $\mathcal{V}_{\Pi}^{\mathcal{T}} = \{bottle, coke, full, drink\}$ and $\mathcal{A}_{\mathcal{T}}(empty) = \{full\}$ (for simplicity we only consider "full"). Then, the antonym equation $not\#empty \sim full = 1.0$ is obtained (also by simplicity, our current implementation assigns a fixed approximation degree in the antonymy equations). Therefore, it could be asked if the bottle will be drunk "`?-drink(X)`", and the system would respond affirmatively by computing "`X=coke`".

C. Reasoning with Hypernymy

Reasoning with hypernymy consists of using hypernymy to obtain a type of reasoning based on this relation. It is formalized by the following definitions.

Definition 7 (Hypernymy-based Direct Inference Rule): The Hipernymy-based Direct Inference Rule states that from A it can be inferred B as immediate consequence, if B is hypernymy of A.

Definition 8 (Hypernymy-based Modus Ponens): Modus ponens based on Hypernymy states that from A and $B \rightarrow C$ it can be inferred as immediate consequence C if B is hypernym of A.

We can proceed just like in the case of synonymy and antonymy calculating in this case the hypernyms of each element of the vocabulary of a program Π. Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $\mathcal{V}_{\Pi}^{\mathcal{T}}$, then for all $w \in \mathcal{V}_{\Pi}^{\mathcal{T}}$, $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(w) = \{h_1, \ldots, h_n\}$ denotes the set of hypernyms of w extracted from $\mathcal{T}$.

Example 7: Given the program in Example 1, the thesaurus $\mathcal{T} = WordNet$ and the vocabulary $\mathcal{V}_{\Pi}^{\mathcal{T}}$ = { at, car, home, airport, driving }. It is possible to obtain a set of hypernyms for each element of the vocabulary: $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(thinking) = \{\ \}$; $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(at) = \{$ element, halogen $\}$; $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(car) = \{$ compartment $\}$; $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(home) = \{$ beginning, origin, root, rootage, source $\}$; $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(airport) = \{$ airfield, field$\}$; $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(driving) = \{$ travel, traveling, travelling, steering, guidance, direction$\}$

Once the hypernyms vocabulary has been constructed, the proximity equations are created, in this case hypernymy equations.

Definition 9 (Hypernym equations for Π*):* Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $V_{\Pi}^{\mathcal{T}}$. Then for all $w \in \mathcal{V}_{\Pi}^{\mathcal{T}}$, and its hypernyms $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(w) = \{h_1, \ldots, h_n\}$, an entry is created for each hypernym h_i such that $w \sim h_i = 1.0$.

Example 8: Let's suppose that someone goes to an exhibition of pictures and we want to model the knowledge that a person expresses in a natural language, for example: "there were pictures of goldfinch and sparrows". In this particular case, if we know that "goldfinches" and "sparrows" are "birds" I could induce that the exposition was about birds. Let's suppose a program $\Pi = \{picture(a, goldfinch), picture(b, sparrow).\}$ and the thesaurus $\mathcal{T} = WordNet$, then $\mathcal{V}_{\Pi}^{\mathcal{T}} = \{picture, goldfinch, sparrow\}$, with $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(picture) = \{show\}$, $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(goldfinch) = \{bird\}$, $\mathcal{H}_{\mathcal{V}_{\Pi}^{\mathcal{T}}}(sparrow) = \{bird\}$. Now, in particular, the system generates the proximity equations $goldfinch \sim bird = 1.0$ and $goldfinch \sim sparrow = 1.0$. So, you could ask about the pictures that are about birds "?-picture(X,bird)", and the system would respond "X=a" and "X=b".

IV. Implementation of the Model of Reasoning in Bousi∼Prolog

In this section we are going to explain how the model of reasoning detailed in the previous section can be incorporated into the Bousi∼Prolog system [2], [3], [4]. In order to obtain the semantic relations we employ the general thesaurus WordNet. We begin with the syntactic aspects. Then, we describe the different implementation phases and data structures which are generated to facilitate the reasoning with words.

A. Syntax

All sorts of programming languages must provide specific instructions to declare and define their data structures. Bousi∼Prolog makes use of one directive which allows us to specify the thesaurus $\mathcal{T}$ we want to connect. The concrete syntax of this directive is: "`:-thesaurus(Thesaurus_Name,[SR_1,...,SR_N])).`" where, $Thesaurus\_Name$ is the name of the thesaurus and $[SR1,\ldots,SRN]$ is a set of (eventually empty) semantic relations defined on such thesaurus.

Example 9: The following directive connects a program to WordNet and will make use of the semantic relations of synonymy, and antonymy for reasoning:

```
:-thesaurus(wordnet,[synonymy,antonymy]).
```

B. Compiling semantic relations

One of the most important features of the Bousi∼Prolog system is its ability to compile all the information regarding the thesaurus defined in a program. In this section the different compilation phases of the thesaurus are detailed. During the syntactic 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 directive "thesaurus" creates a connection with the thesaurus and generates the vocabulary of Π and $\mathcal{T}$. Once the connection has been established, the next phase is focused on generating fuzzy relations from the semantic relations indicated as arguments in the directive. This is formalized by the Algorithm 1 whose objective is to compile all the semantic information associated with a program Π and a thesaurus $\mathcal{T}$.

Algorithm 1:

Input: A program Π and a thesaurus $\mathcal{T}$.
Output: A set of entries which defines a semantic relation $\mathcal{R}$.
Initialization: $\mathcal{R} := \emptyset$
Calculate $\mathcal{V}_\Pi^\mathcal{T}$
For each w $\in \mathcal{V}_\Pi^\mathcal{T}$
1) Compute Synonyms: $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(w) = \{s_1, \ldots, s_n\}$ $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(w, s_i) = \alpha_i\}$;
2) If $w \equiv not\#u$,
   a) Compute Antonyms: $\mathcal{A}_{\mathcal{V}_\Pi^\mathcal{T}}(u) = \{a_1, \ldots, a_n\}$
   b) $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(not\#u, a_i) = 1.0\}$;
3) Compute Hyperonyms: $\mathcal{H}_{\mathcal{V}_\Pi^\mathcal{T}}(w) = \{h_1, \ldots, h_n\}$ $\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}(w, h_i) = 1.0\}$;
endFor
Return $\mathcal{R}$

Example 10: Following with the Example 1 let's suppose that we want to reason by using the semantic relations (synonymy, antonymy and hypernymy). This can be codified in Bousi∼Prolog as follows:

```
:-thesaurus(wordnet,
              [synonymy, antonymy, hypernymy]).

at(X,airport) :- at(X,car),
                 driving(X,home,airport).
at(john, taxi).
driving(X,not#away,aerodrome).
```

The vocabulary generated for this program is a merge of the ones appeared in the Examples 4, and 7, in combination with the set of antonyms of the word "`away`" which is prefixed by the modifier "`not#`": $\mathcal{A}_{\mathcal{V}_\Pi^\mathcal{T}}(away) = \{\text{home}\}$. Thanks to the implementation of Algorithm 1 the proximity equations "`aerodrome∼airport=1.0`" (by synonymy), "`not#way∼home=1.0`" (by antonymy) , "`taxi∼car=1.0`" (by hypernymy) are generated. Then, if we ask the question "`?-at(X, aerodrome).`" the system responds "`X=john`" with approximation degree `1.0`.

Observe that general rules have been developed without having to specify each element, since this is provided automatically by WordNet and our reasoning is done based on it. Note also that this way to proceed can cause problems, since all the semantic relations of a given word are used. This can cause inefficiencies and even an inadequate reasoning. Therefore, automatic techniques to prune and to improve ontologies are necessary. Also the problem of the *context treatment* emerges. This problem and a potential solution is presented in the following section.

C. Integration of WordNet in Bousi∼Prolog

The Bousi∼Prolog architecture consists of three layers: GUI, compiler and abstract machine. The compiler is responsible for translating Bousi∼Prolog programs into Similarity-based Abstract Machine Code. The Similarity-based WAM [2] is an extension of the WAM that allows the execution of Bousi∼Prolog programs. It modifies the two parts of which the original machine consists: the memory layout and the instruction set.

In order to be able to use WordNet together with Bousi∼Prolog, a tool for connecting with WordNet, called WN∼Bousi[2], has been built and incorporated to the Bousi∼Prolog architecture by enhancing the compiler layer. This tool allows to consult semantic relations for each one of the symbols present in a Bousi∼Prolog program. To be precise, WN∼Bousi is a GUI for WordNet and WordNet::Similarity which uses a library called RitaWN[3] which allows to get all the features of WordNet for a Java software application. This can be integrated into the Bousi∼Prolog system or it can work in an independent way. The WN∼Bousi architecture has two layers: presentation and domain.

- The presentation layer: is composed by two classes (FMessage.java and FWordNet.java) and it allows to perform the I/O in a graphical way by means of a WordNet Window. This has options to consult the semantic relations for a particular word and to enable the use of WordNet::Similarity. It is accessed via the menu bar of the Bousi∼Prolog GUI (see Figure 1).
- The domain layer: is composed by two classes (WN-Symbol.java and WordNet.java). It is responsible for the hole functionality described in previous sections, that is: (i) it computes the semantic relations (indicated in the directive `thesaurus`) for each symbol (representing a word) in the program and (ii) it generates the proximity equations.

V. Problems in reasoning with words

When we employ a general thesaurus to obtain the semantic relations, the context is not taken into consideration. In our framework, this is produced because all the possible proximity equations are used in the inference process. That is, we do not use context information to generate the most adequate proximity equations that must be employed. The following program shows this fact:

```
% proximity equations from wordnet::similarity
bank~deposit=1.0.    bank~swear=1.0.
bank~trust=1.0.
```

[2]Available at: **http://www.face.ubiobio.cl/∼clrubio/bousiTools/**
[3]Available at: **http://www.rednoise.org/rita/wordnet/documentation/**

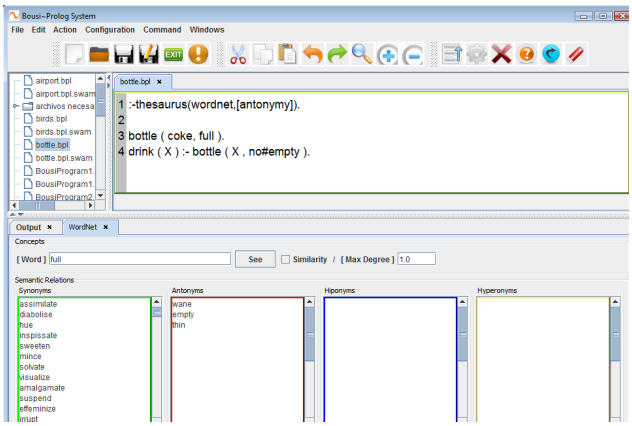

Fig. 1. WN∼Bousi: WordNet in the GUI of the Bousi∼Prolog system

```
% knowledge representation
person(john,job(bank)).
person(mary,job(park)).
```

In this case, we use WordNet::Similarity to consult which words are closer to "bank", we obtain "deposit", "swear" and "trust". Note that the context is not taken into consideration, what generates certain unnecessary proximity equations. A potential solution is to determine which is the appropriate semantic relation for a concrete situation, e.g., the meaning of a word in a particular context. A way to deal with this problem is to use the symbols close to the symbol in question to identify the context where it appears, what we call structural context (SC for short) for a symbol "s". For example, because the word "bank" appears in an atomic formula like "person(john,job(bank))" we can infer that there exists a structural dependency with regard to the words "job" and "person" ("bank" is a job of a person). From there we may construct the structural context: SC(bank)={person, job}. Then, in order to select the more adequate proximity equations we can compute the approximation degree between the symbols which are related to "s" ("bank" in this case) and its structural context. We obtain three groups: (a) {job ∼ deposit = 0.426, person ∼ deposit = 0.256} for "deposit"; (b) {job∼swear=0.0, person∼swear=0.0} for "swear"; and (c) {job∼trust=0.439, person∼trust=0.318} for "trust". So the proximity equations "bank∼trust=1.0" and "bank∼deposit=1.0" would be the right candidates. Finally we could discard the proximity equation "bank∼swear=1.0" because it has very little relation with the structural context of "bank".

VI. Related Works

The approach closer to ours is [10] in which a linguistic natural fuzzy Prolog is suggested. This prolog consists of a synonymy-based unification mechanism and a antonymy-based resolution procedure. This idea is employed in our work in some sense. The main difference is that, in our framework, antonymy is not employed directly but as a similarity of opposite concepts. The reason is that we adapt the model of reasoning described in this paper to be implemented in our system i.e. Bousi∼Prolog and WordNet.

VII. Conclusions and Future Work

We have presented a first approximation for reasoning with words. The idea consists of employing the semantic relations between words as a constituent part of a weak unification algorithm of a fuzzy logic programming language. As main advantage we can mention that knowledge is obtained from the thesaurus in an automatic way and hence it is not necessary to introduce all relations explicitly. In order to materialize these ideas, we have integrated the thesaurus WordNet into the Bousi∼Prolog system and we have extended some syntactical and semantical aspects of this language. All this changes enable reasoning in presence of synonymy, antonymy and hypernymy. Although these techniques are in their first stages, we think they can contribute to meet fuzzy logic programming languages based on weak unification and the computing with words paradigm.

Despite its power, there are several important problems that must be fit in this new framework. First, it will be necessary to establish its theoretical foundation and secondly the treatment of the context will be an essential requirement to improve the inference processes.

Acknowledgements: This work has been partially supported by FEDER and the Spanish Ministry of Economy and Competition under grant TIN2013-45732-C4-2-P and it is the result of the work of the research group, SOMOS (SOftware - MOdelling - Science), funded by the Dirección de Investigación and Facultad de Ciencias Empresariales of the Universidad del Bío-Bío under grant 130415 GI/EF.

References

[1] J. De Smedt and B. Vatant. The EUROVOC Thesaurus Ontology Schema. Publications Office of European Union (2009).
[2] P. Julián-Iranzo and C. Rubio-Manzano. A similarity-based WAM for Bousi∼Prolog. In: LNCS, vol 5517, pp. 245–252. Springer, Heidelberg (2009).
[3] P. Julián-Iranzo and C. Rubio-Manzano. An Efficient Fuzzy Unification Method and its Implementation into the Bousi∼Prolog System. In: Proc. of FUZZ-IEEE'10, (2010).
[4] C. Rubio-Manzano and P. Julián-Iranzo. Fuzzy linguistic prolog and its applications Journal of Intelligent and Fuzzy Systems, vol. 26 pp. 1503–1516 (2014).
[5] A. Kumar, B. Smith The Unified Medical Language System and the Gene Ontology: Some Critical Reflections. Lecture Notes in Artificial Intelligence 2821, 135–148 (2003).
[6] H. Liu and P. Singh ConceptNet: a practical commonsense reasoning tool-kit BT Technology Journal Vol 22 No 4 October (2004).
[7] G.A. Miller. WordNet: A Lexical Database for English. Communications of the ACM,1995, 38, pp. 39–41 (1995).
[8] T. Pedersen, S. Patwardhan and Jason Michelizzi. WordNet::Similarity - Measuring the Relatedness of Concepts. In: *Proc. AAAI-04*, pp. 1024-1025 (2004).
[9] S. Fernández Lanza et al. Introducing FDSA: applications on information retrieval and stand-alone use. In J. Mathware and Soft Computing 10 (2-3): 57-70, 6 Ref (2003).
[10] A. Sobrino. The Role of Synonymy and Antonymy in 'Natural' Fuzzy Prolog. Soft Computing in Humanities and Social Sciences. Studies in Fuzziness and Soft Computing Vol. 273 Springer (2012).
[11] E. Trillas, S.Cubillo and E. Castiñeira. On Antonymy from Fuzzy Logic. X Spanish Conference on Fuzzy Logic (2000).
[12] L.A. Zadeh. Computing With Words and Perceptions: A Paradigm Shift. In Joint Colloquium Distinguished Lecture Series June 22 (2009).