Atlantis Studies in Uncertainty Modelling, volume 1

11th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2019)

WordNet and Prolog: why not?

Pascual Julián-Iranzo[a] **and Fernando Sáenz-Pérez**[b]

[a]Dept. of Information Technologies and Systems,
University of Castilla-La Mancha, Spain, Pascual.Julian@uclm.es

[b]Dept. of Software Engineering and Artificial Intelligence,
Universidad Complutense de Madrid, Spain, fernan@sip.ucm.es

Abstract

This paper describes a set of Prolog modules with predicates to access the information stored in the lexical database WordNet. The aim is to use the defined predicates for empowering (fuzzy) logic programming systems for approximate reasoning tasks. To achieve this goal it is necessary to have means to calculate the degree of relationship between the words. Because WordNet relates words but does not give graded information of the relation between those words, it is necessary to implement standard methods to compute that gradation.

Keywords: Prolog, WordNet, Similarity Measures.

1 Introduction

WordNet is a lexical English language database that was manually constructed and maintained by the Cognitive Science Laboratory of Princeton University under the direction of psychology professor George A. Miller [11]. The project began in 1985 and later, has been promoted by Christiane Fellbaum [3, 4] among other researchers.

WordNet stores words of four syntactic categories: (1) nouns, (2) verbs, (3) adjectives, and (4) adverbs. These words are grouped into sets of synonyms called *synsets*. Roughly speaking, the words of a synset have the same meaning in a determined context and they represent a *concept* (or word sense). Each synset has a `synset_ID` which is a nine byte field, where the first byte defines the syntactic category of the synset (i.e., `synset_ID`s of a noun starts with 1, verbs with 2, etc.) and the remaining eight bytes are a `synset_offset`. Because a word has different senses (meanings), it can belong to different synsets. WordNet is structured as a semantic net where words are interlinked by lexical relations, and synsets by semantic relations. Synonymy and antonymy are the major lexical relations. Semantic relations serve to build knowledge structures (i.e., networks of synsets –concepts). Depending on the syntactical category, there are different semantic relations able to build that structures.

Among the semantic relations that create knowledge structures, we are mainly interested by those that build hierarchies of concepts. Only nouns (through the hyponymy/hypernymy relation) and verbs (through the hyponymy/hypernymy relation and their different entailment relations) can be organized as hierarchical structures. There are 25 of these hierarchies for nouns and 15 for verbs. All are linked to a unique “root” synset, in order to link these kinds of words. Noun hierarchies are far deeper than verb hierarchies. Adjectives are more complex. They can be visualized as “dumbbells” rather than as “trees”.

In this paper, our will is to use Prolog technology to access WordNet. Surprisingly, there has been little activity when applying declarative technologies, and in particular Prolog, to consult the information stored in WordNet. We only know the work of transforming the WordNet database into Prolog format made by Eric Kafe (available at `https://github.com/ekaf/wordnet-prolog`), a preliminary work by Sarah Witzig [16] (which, although available to the public, was never published in a conference), and a SWI-Prolog package supplied by Jan Wielemaker (available at `https://github.com/JanWielemaker/wordnet`). Only the work of Witzig defines predicates able to provide complex information such as chains of hypernym and other semantic information (see below). However, we believe that the use of Prolog to access the contents of WordNet can provide efficiency, ease of programming, and deductive capacity.

We have developed a library of Prolog programs, able to retrieve informations from WordNet, that can be loaded into a Prolog system or incorporated to other (fuzzy) logic programming system implemented in Prolog. As an additional motivation for this work, we have incorporated this library into the Bousi∼Prolog fuzzy logic programming system [15, 8] to facilitate approximate reasoning tasks and a more flexible query answering process.

2 WordNet and Prolog

WordNet can be accessed via a web interface or locally. In the last case there are different options, but we are interested in the Prolog version for the ease of connection to our logic programming systems. This version is the WordNet 3.0 database released by Eric Kafe which can be found at the URL `https://github.com/ekaf/wordnet-prolog`. The information stored in the WordNet database is provided as a collection of Prolog files. Each file contains the definition of what is called an *operator*, which corresponds to a WordNet relation. Files are named as `wn_<`*`operator`*`>.pl`, where `<`*`operator`*`>` is the name of a specific operation (relation). Therefore, each WordNet relation is represented by a Prolog predicate which is stored in a separate file and defined by a set of Prolog facts. The specification of these predicates are detailed in [5]. In the following, we describe those predicates which are interesting for the present work.

The file `wn_s.pl` contains all the information about words stored in the WordNet database. It defines the `s` operator, which has an entry for each word. The structure of the `s` operator is:

```
s(Synset_id, W_num, Word, Ss_type,
             Sense_number, Tag_count)
```

Where the `W_num` parameter indicates which word in the synset is being referred to. The words in a synset are numbered serially, starting with 1. The third argument is the word itself (which is represented by a Prolog atom). The `Ss_type` parameter is a one character code indicating the synset type: `n` (noun); `v` (verb); `a` (adjective); `s` (satellite adjective)[1] and `r` (adverb). The `Sense_number` parameter specifies the sense of the word, within the part of speech encoded in the `Synset_id`. The higher the sense number, the less common is the word. Finally, the `Tag_count` indicates the number of times the word was found in a test corpus. A higher tag count number means that the word is more common than others with a lower tag count.

[1]A word is an adjective if it belongs to a head synset. It is an adjective satellite if it belongs to a satellite synset. Head synsets contain at least one word that has an antonym. Satellite synsets do not contain any word that has an antonym but their words are connected by similarity with the words of a head synset. [16]

The file `wn_hyp.pl` stores hypernymy relations in the binary predicate

```
hyp(synset_ID1,synset_ID2).
```

specifying that the second synset is a hypernym of the first synset. This semantic relation only holds for nouns and verbs. Because hyponymy is the inverse relation of hypernymy, the operator `hyp` also specifies that the first synset is a hyponym of the second synset.

The file `wn_ent.pl` stores entailment relations in the binary predicate

```
ent(synset_ID1,synset_ID2).
```

which specifies that the second synset is an entailment of first synset. As already commented, this semantic relation only holds for verbs.

The file `wn_sim.pl` defines a relation between adjectives which are similar in meaning.

```
sim(synset_ID1,synset_ID2).
```

Note that there is a symmetric entry `sim(synset_ID2, synset_ID1)` defined for every clause, because similarity works in both directions. This relation only holds for adjectives. More precisely, it applies either two head synsets, or one head synset and one satellite synset. It does not apply to two satellite synsets.

3 The connection with WordNet

`WN_CONNECT` is a software application prototype that aims to access the lexical database WordNet. One of its main features is that it has been fully implemented using Prolog. It is divided into ten modules, and some of them will be explained later on. `WN-CONNECT` is available at `https://dectau.uclm.es/bousi-prolog/applications/`. Since the source code is well documented, in this paper we concentrate in the functionality of predicates instead of a more implementation oriented explanation of the predicates implemented in the modules that conform this software.

A general characteristic of the predicates implemented in these modules is that the parameter Word (occurring in that predicates) is a term that follows the syntax `Word[:SS_type[:Sense_num]]` and actually represents a concept identified by a synset ID. Where `SS_type` is a one character code indicating the synset type (`n`, `v`, `a`, `s`, or `r`) and `Sense_num` specifies the sense number (meaning) of the word, within the part of speech encoded in the synset identifier. `Sense_num` is a natural number: 1, 2, 3, ... Note that sometimes this term may be partially specified; that is, `SS_type` and Sense_num could be variables (or even omitted). Often, in this paper we call those terms *word terms*.

3.1 Base modules

Module wn : This module was implemented by Jan Wielemaker (available at: https://github.com/JanWielemaker/wordnet). It discloses the Wordnet Prolog files in a more SWI-Prolog friendly manner. It exploits SWI-Prolog demand-loading and SWI-Prolog Quick Load Files to load 'just-in-time' and as quickly as possible.

The system creates Quick Load Files for each wordnet file needed if the .qlf file does not exist and the wordnet directory is writeable. For shared installations it is adviced to run load_wordnet/0 as user with sufficient privileges to create the Quick Load Files.

Module wn_synsets : This module implements predicates to retrieve information about words and synsets stored in WordNet. It uses the modules wn_portray or wn depending on whether the evironment variable WNDEVEL is set or not.

The public predicates implemented in this module are:

```
wn_word_info(+Word)
wn_gloss_of(+Word, -Gloss)
wn_synset_ID_of(+Word, -W_Synset_ID)
wn_synset_of(+Word, -W_synset)
wn_synset_components(+Synset_ID,-Synset_Words)
```

with the obvious declarative semantics. A couple of goal examples follow:

```
?- wn_word_info(lion).
INFORMATION ABOUT THE WORD 'lion' :
 Synset_id = 102129165
 Word Order num. = 1
 Synset type (n, v, a, s, r) = n
 Sense number = 1
 Tag_count = 2
 -----------
 Gloss:
 large gregarious predatory feline of Africa
 and India having a tawny coat with a shaggy
 mane in the male
 -----------

?- wn_synset_of(lion:n:1, W_synset).
W_synset = [lion:n:1, 'king of beasts':n:1,
            'Panthera leo':n:1].
```

Module wn_hypernyms : This module implements predicates to retrieve information about hypernyms of a concept (synset). These predicates only work with either nouns or verbs. It uses the modules wn_synsets and wn_utilities.

The public predicates implemented in this module are:

```
wn_hypernyms(+Hyponym, -List_SynSet_HyperNym)
wn_display_hypernyms(+Hyponym)
wn_display_graph_hypernyms(+Hyponym)
wn_lcs(+List_of_Words, -LCS)
wn_lcs/(+Word1, +Word2, -LCS)
```

Notably, wn_hypernyms/2 returns a list List_SynSet_HyperNym of hypernym synset_IDs of a word (term) Hyponym. This is a non-deterministic predicate and, therefore, it can compute all the HyperTrees of the word Hyponym, and wn_display_graph_hypernyms/1 displays them graphically.

The predicates wn_lcs/2 and wn_lcs/3 compute the Least Common Subsumer (LCS) of a set of words and two words, respectively.

These predicates play an important role in the computation of similarity measures between concepts.

Figure 1 shows the result of invoking the goal wn_display_graph_hypernyms(person).

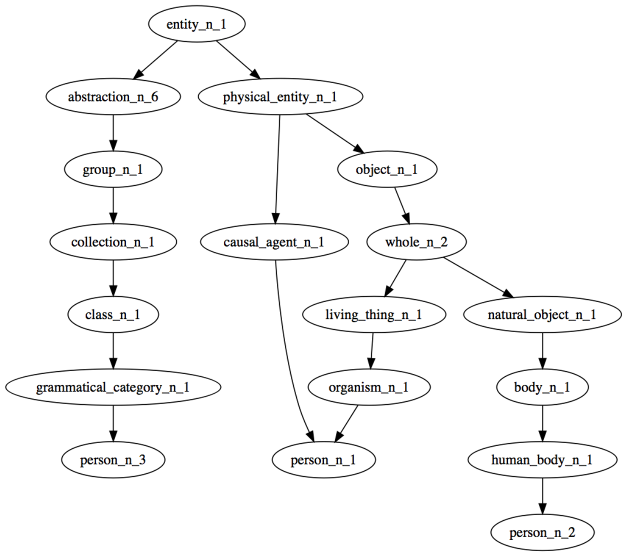

Figure 1: Hypernyms of the word person (all senses)

Note that, in Figure 1, each node draws the representative word of the respective synset (i.e., those with W_num equal to one). For obtaining information about all the words compounding the related synsets, the predicate wn_display_hypernyms/1 is available:

```
?- wn_display_hypernyms(person:n:3).
[entity:n:1]
 >> [abstraction:n:6,abstract entity:n:1]
 >> [group:n:1,grouping:n:1]
 >> [collection:n:1,aggregation:n:1,
     accumulation:n:2, assemblage:n:4]
 >> [class:n:1,category:n:1,family:n:3]
 >> [grammatical category:n:1,
     syntactic category:n:1]
 >> [person:n:3]
```

Module wn_hyponyms : This module implements predicates to retrieve information about hyponyms of a concept (synset). These predicates only work with either nouns or verbs. It uses modules `wn_synsets` and `wn_utilities`.

The public predicates implemented in this module are:

```
wn_hyponyms(+Hypernym, -List_SynSet_Hyponyms)
wn_gen_all_hyponyms_of(+Synset_ID,
        -List_all_Hyponym_IDs)
wn_hyponyms_upto_level(+Hypernym, +Level,
        -List_SynSet_Hyponyms)
wn_gen_hyponyms_upto_level(+Synset_ID,+Level,
        -List_Hyponym_IDs)
wn_display_graph_hyponyms(+Word, +Level)
```

These predicates works with a either a word term (`Hypernym`) or a synset identifier (`Synset_ID`), giving a list of synset identifiers of its hyponyms.

The predicate `wn_gen_all_hyponyms_of/2` generates all the hyponyms of a concept (`Synset_ID`) and it is specially useful for computing the information content of a concept.

Among the predicates implemented in this module, `wn_display_graph_hyponyms/2` is relevant because it shows a graphic representation of all the hyponyms corresponding to all the senses of the word (term), level by level. Nodes of the graph only show the most representative word associated to that hyponym synset. For instance, Figure 2 shows the result of submitting the goal `wn_display_graph_hyponyms('Homo sapiens',2)`.

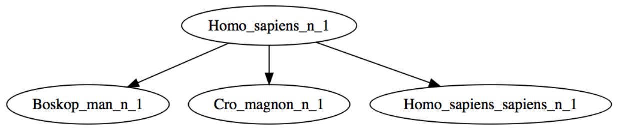

Figure 2: Hyponyms of `Homo sapiens` (sense 2)

Module wn_similar_adjectives : This module defines predicates that find the adjectives which are similar in meaning to a input adjective. Do not confuse "similar" with "synonym". Synonym words are grouped in a synset and they are words equal in meaning. In other words, synonym words must have the maximum (top) degree of similarity.

Most of the predicates defined in this module are based on the operator `sim`: `sim(synset_id,synset_id)`. The `sim` operator specifies that the second synset is similar in meaning to the first synset. This means that the second synset is a satellite of the first synset (or viceversa), which is the cluster head. This relation only holds for adjective synsets contained in adjective clusters.

This module uses module `wn_synsets` and implements the following public predicates:

```
wn_sim_adjectives_of(+Adjective,
        -List_sim_SynSets)
wn_display_sim_adjectives_of(+Adjective)
wn_display_graph_sim_adjectives_of(Adjective)
wn_display_graph_cluster_of(Adjective)
```

with the obvious declarative meaning.

The predicate `wn_display_graph_cluster_of/1` displays a graphical representation of a cluster. A *cluster* is a structure that links a head adjective with its antonym adjective, and both of them with their respective satellite adjectives.

3.2 Reasoning with WordNet

Once implemented the declarative tool WN-CONNECT, which allows us to easily access the information contained in WordNet, it can be used to formulate all kinds of more complex questions. For example, it can be asked how many names does this lexical database store:

```
?- findall(Word, wn_s(_,_,Word,n,_,_),
           Word_List),
   length(Word_List, Word_Nouns).
Word_List = [entity, 'physical entity', ...],
Word_Nouns = 146347.
```

Or better yet, it is possible to discover inconsistencies in the stored information. The following queries:

```
?- wn_hyp(X,Y), wn_hyp(Y,X).
X = 202422663,
Y = 202423762 .

?- wn_synset_ID_of(X_Word, 202422663).
X_Word = restrain:v:1 .

?- wn_synset_ID_of(Y_Word, 202423762).
Y_Word = inhibit:v:4 .
```

reveal that `restrain` is a hyperonym of `inhibit` and vice versa, which constitutes an inconsistency.

But, above all, our main interest is to use the deductive potential of a language like Prolog in combination with WordNet for text mining (information retrieval, text classification, and even sentiment analysis).

3.3 Implementing relatedness measures

Although a large variety of measures of semantic relatedness and similarity have been proposed [2], only a limited number of tools have been implemented to perform this task. Perhaps, WordNet::Similarity [13] is the most prominent. This tool has three similarity measures based on path lengths between concepts (PATH, WUP [17] and LCH [9]), and three based on information content[2] (RES [14], JCN [7] and LIN [10]). Additionally, WordNet::Similarity provides three relatedness measures (HSO [6], LESK [1] and VECTOR [12]).

We have concentrated on the implementation of these standard similarity measures. Semantic similarity is a special case of relatedness which quantifies how much two words are alike (or more precisely: how similar are the concepts they denote).

In this section, we use the following definitions and notations usually used when working in the framework of WordNet:

- Given a HyperTree, the *length of the shortest path* from synset c_1 to synset c_2 is denoted by $len(c_1, c_2)$. The *depth of a node* c is the length of the shortest path from the global root to c, i.e., $depth(c) = len(root, c)$.
- Some similarity measures use the notion of *least common subsumer* (LCS) of two concepts, which is the most specific concept they share as an ancestor. We write $lcs(c_1, c_2)$ for the least common subsumer of c_1 and c_2.

Module wn_sim_measures : This module implements predicates to compute standard similarity measures between concepts based on counting edges. It uses modules `wn_hypernyms` and `wn_synsets`, implementing the following public predicates:

```
wn_path(+Word1, +Word2, -Degree)
wn_wup(+Word1, +Word2, -Degree)
wn_lch(+Word1, +Word2, -Degree)
```

The predicate `wn_path/3` implements the PATH similarity measure. It takes two concepts (word terms – `Word:SS_type:Sense_num`) and returns the degree of similarity between them. Note that we do not explicitly require information about the synset type and sense number of a word term (that can be variables).

We check that both `Word1` and `Word2` are either nouns or verbs but not combinations of them.

A concept may have different HyperTrees. Therefore, depending on the different HyperTrees of $c1$ and $c2$ involved in the computation, different similarity values can be obtained according to the formula:

$$sim_{PATH}(c1, c2) = 1/len(c1, c2)$$

[2]The information content of concepts is derived from tagged as well as un-tagged corpora of plain text.

where $len(c1, c2) = (depth(c1) - LCS\_depth) + (depth(c2) - LCS\_depth) + 1$

This predicate combines all HyperTrees of $c1$ and $c2$, computes the respective similarity values and returns the maximum.

The predicate `wn_wup/3` implements the WUP similarity measure. The computation scheme is like the one explained in the wn_path/3 measure.

$$sim_{WUP}(c1, c2) = \frac{2 \times depth(lcs(c1, c2))}{depth(c1) + depth(c2)}$$

The predicate `wn_lch/3` implements the LCH similarity measure. The computation scheme is like the one explained in the wn_path/3 measure.

$$sim_{LCH}(c1, c2) = -ln(\frac{len(c1,c2)}{2 \times max\{depth(c) | c \in \text{WordNet}\}})$$

where

$$\begin{aligned} len(c1, c2) &= (depth(c1) - LCS\_depth) + \\ &\quad (depth(c2) - LCS\_depth) + 1 \end{aligned}$$

and $max\{depth(c) | c \in \text{WordNet}\}$ is the maximum depth of a concept in the WordNet database. In practice, is a fixed constant for each part of speech: $maxDepth(n) = 20$ (nouns); $maxDepth(v) = 14$ (verbs).

Module wn_ic_measures : This module implements predicates to compute standard similarity measures between concepts based on information content (IC). It uses the modules `wn_hypernyms` and `wn_synsets`. The public predicates implemented in this module are:

```
wn_res(+Word1, +Word2, -Degree)
wn_jcn(+Word1, +Word2, -Degree)
wn_lin(+Word1, +Word2, -Degree)
wn_information_content(+Word, -IC)
```

These predicates take two concepts (word terms – `Word:SS_type:Sense_num`) and returns the degree of similarity between them.

As in the last case, we do not explicitly require information about the synset type and sense number of a word, and we check that both `Word1` and `Word2` are nouns or verbs but not combinations of them.

The predicate `wn_res/3` implements the RESNIK similarity measure, based on information content. Because a concept can have different HyperTrees, depending on the HyperTrees of $c1$ and $c2$ involved in the computation, different LCSs are obtained, leading to different similarity values:

$$sim_{RES}(c1, c2) = IC(lcs(c1, c2))$$

This predicate combines all HyperTrees of $c1$ and $c2$, computes the respective similarity values, and returns the maximum.

The predicate `wn_jcn/3` implements the Jiang & Conrath [7] similarity measure, based on information content. The computation scheme is like the one explained in the `wn_res/3` measure.

$$sim_{JCN}(c1, c2) = \frac{1}{IC(c1)+IC(c2)-2\times IC(lcs(c1,c2))}$$

The predicate `wn_lin/3` implements the LIN similarity measure, based on information content. It follows the same computation scheme just explained for the `wn_res/3` measure.

$$sim_{LIN}(c1, c2) = \frac{2 \times IC(lcs(c1, c2))}{IC(c1) + IC(c2)}$$

The predicate `wn_information_content/2` computes the information content IC of the concepts designed by the different senses of `Word`. It relies on the private predicate `information_content/3`, which computes the information content IC of the concept denoted by the `Synset_ID`. The function IC is defined as the natural logarithm of the probability of encountering an instance of a concept c (measured in terms of a relative frequency of use of the concept c in a corpus). That is, it is defined as:

$$IC(c) = -ln(\frac{frequency\_of\_use(c)}{frequency\_of\_use(Root\_ID)})$$

The parameter $Root\_ID$ is the synset number of the concept in the root of the hierarchy.

Module wn_rel_measures : This module implements a new relatedness measures between concepts based on the Jaccard index. This measure has not a good performance and it has to be improved in a future work.

This module uses module `etu` (the Michael A. Covington's Efficient Tokenizer), module `wn_synsets` and module `library(snowball)` (the Snowball multilingual stemmer library).

The public predicates implemented in this module is `wn_yarm(+Word1, +Word2, -Degree)`. YARM (Yet Another Relatedness Measure) compares the glosses `SGL_W1` and `SGL_W2` of two words, after removing stop words and stemming, by computing the Jaccard index for them as the relatedness Degree:

$$\begin{aligned} Degree &= \frac{|SGL\_W1 \cap SGL\_W2|}{|SGL\_W1 \cup SGL\_W2|} \\ &= \frac{|SGL\_W1 \cap SGL\_W2|}{(|SGL\_W1|+|SGL\_W2|-|SGL\_W1 \cup SGL\_W2|)} \end{aligned}$$

4 Installing `WN-CONNECT`

In this section we give a brief guide on how to install WN-CONNECT. First, keep in mind that `WN-CONNECT` requires SWI-Prolog to be pre-installed.

In order to install `WN-CONNECT`, follow these steps:[3]

1. Download WordNet 3.0 Prolog version from `http://wordnetcode.princeton.edu/3.0/WNprolog-3.0.tar.gz`, and unzip it in a directory of your choice (for example: `/usr/local/WordNet-3.0/wn_prologDB`).

2. Open a terminal and set the environment variable `WNDB` to this newly created directory. For example, In a Bourne-like Shell, write: `export WNDB=/usr/local/WordNet-3.0/wn_prologDB`.

3. Download the modules of `WN-CONNECT` from `https://dectau.uclm.es/bousi-prolog/applications/` and unzip it in a directory of your choice (for example: `/home/myuser/wn`).

4. In the terminal opened, extends the environment variable PATH with the previous directory. For example write: `export PATH=/home/myuser/wn:$PATH`

5. Execute the shell script `wn.sh`.

If graph display predicates are used, Graphviz must be accessible via the `PATH` environment variable.

5 Integration into Bousi∼Prolog

As mentioned in the introduction, we started this work with the initial intention of integrating WordNet into the BPL system and thereby providing our language with linguistic capabilities.

Bousi∼Prolog (BPL) [15, 8] is a fuzzy logic programming language that replaces syntactic unification of classical SLD-resolution by a fuzzy unification algorithm. This algorithm provides a weak most general unifier and a corresponding unification degree (which takes a numeric value between 0 and 1). Intuitively, the unification degree represents the truth degree associated with the computed instance to a goal. A proximity relation is a reflexive and symmetric, but not necessarily transitive, binary fuzzy relation on a set. Each entry in this relation sets the approximation degree D between two elements X and Y in the universe of discourse, and it takes the form $X \sim Y = D$. The operational semantics of BPL includes a fuzzy unification algorithm which takes the proximity relation to deliver the approximation degree for both fuzzy (weak) unification and goal solving. Also, proximity relations are used to implement fuzzy sets (linguistic variables),

[3]These steps are intended for installing `WN-CONNECT` in a unix-like system. Similar steps would be needed for a Windows system.

which leads to an elegant, simple, natural and efficient fuzzy Prolog system based on weak unification. In addition, a user-defined lambda-cut sets the minimum truth degree for goal solving, therefore pruning non-relevant solving paths. Filtering, a recent technique included in this system, also aids in this pruning at compile-time, therefore notably improving its space consumption and solving time.

At least, a couple of system implementations have been developed for this language, and its high-level version has recently received different improvements along the last years.[4] Since version 3.1, the proposed work in this paper has been integrated in BPL for making available WordNet in a fuzzy logic programming setting. It has been tailored to extract linguistic properties from WordNet and represent them as an ontology. It is capable of automatically extracting semantic similarity information from either the IS-A relation (a generalization-specialization relation taken from Wordnet's *hypernymy-hyponymy* relation) or based on the frequency of use concepts as explained before (Section 3.3). Therefore, relatedness measures are a key understanding for a system user to interpret the results of this automatic process.

Provided that the user is interested in the semantic similarity between specific concepts, a suitable ontology can be built. The system provides a language directive which can be used to automatically build such an ontology:

```
:- wn_gen_prox_equations(+Measure,
                         +ListOfListsOfPatterns).
```

Where `Measure` is the similarity measure which can have any of the following constant values in $M = \{$ `path`, `wup`, `lch`, `res`, `jcn`, `lin` $\}$. These constants correspond to the relatedness measures introduced previously (Section 3.3). The second argument `ListOfListsOfPatterns` is a list for which each element is another list containing the patterns that must be related by proximity equations. The pattern can be either a word or a word term (i.e., a structure *Word*:*Type*:*Sense* as explained in Section 3). If the pattern is simply a word, then a sense number of 1 is assumed, and its type is made to match all other words in the same list.

A very simple example program containing only the definition of an ontology is:

```
:- wn_connect.
:- wn_gen_prox_equations(wup,
       [[man,human,person],
        [grain:n:8,wheat:n:2]]).
```

[4]See its web page at `https://dectau.uclm.es/bousi-prolog`. The system can be downloaded from `https://dectau.uclm.es/bousi-prolog/downloads/`, and there is also a web interface at: `https://dectau.uclm.es/bplweb`.

Note that the first line includes the directive `:- wn_connect`, which establishes the connection between Bousi~Prolog and WordNet. It can has an optional parameter indicating the specific directory in which WordNet is installed. When executed, it loads the modules introduced in previous sections to make them available for the generation of ontologies. On the first run, it compiles those modules on-the-fly so that a first program loading can take more time than probably expected. Further runs use the already compiled files and loading times are greatly improved.

This example requires the system to automatically build proximity equations between each pair of patterns found for each list. For the first list, the sense number is 1 and the common type is `n`. For the second one, specific patterns are provided and only a proximity equation is generated. All such equations are:

$$
\begin{aligned}
R = \{ & man \sim human = 0.56, \\
& man \sim person = 0.8888888888888888, \\
& person \sim man = 0.8888888888888888, \\
& human \sim person = 0.6086956521739131, \\
& grain \sim wheat = 0.2608695652173913\}
\end{aligned}
$$

The proximity relation is defined by the reflexive and symmetric closure R^+, containing a total of 13 entries. The system computes each element $X \sim Y = D$ in R by calling the predicate `wn_`*`measure`*`(`X`, `Y`, `D`)` for each pair of atoms (X,Y) in the corresponding lists, where *`measure`* is one of the symbols in the measures M, so that goals to predicates such as `wn_wup/3` are called for computing D. These predicates are those defined in Section 3.3

With this program loaded in the system, the user can submit a goal like the following:

```
BPL> man~human=D
D = 0.56
With approximation degree: 1 .
Yes
```

which returns in the logic variable `D` the approximation degree corresponding to the weak unification of the atoms `man` and `human` (that fits with their WordNet semantic similarity as computed by the WUP measure).

6 Conclusions

In this paper, we have described a set of Prolog modules with predicates to access the information stored in the lexical database WordNet. The predicates defined in these modules can be either consulted from a

Prolog interpreter or integrated in a (fuzzy) logic programming system as built-in predicates, providing an enhanced functionality to those systems. For instance:

1. Common but useful information about words and synsets stored in WordNet can be obtained.
2. HyperTrees, which are represented as lists of hypernyms synset IDs of an hyponym, can be computed. These HyperTrees can be textually or graphically displayed.
3. All the hyponyms of an hypernym can be computed (level by level). The result is a tree of hyponyms that can be graphically displayed.
4. All standard similarity measures found in WordNet::Similarity [13] has been implemented. Also we provide a new relatedness measure based on the Jaccard index.

The main motivation of this work was to construct a library for empowering a fuzzy logic programming language called Bousi∼Prolog. The library `WN-CONNECT` gives Bousi∼Prolog access to WordNet and the ability to generate automatically what we call proximity equations, linking two words with an approximation degree. Proximity equations are the key syntactic structures that, in addition to a weak unification algorithm, make possible a flexible query answering process in this kind of programming languages.

Acknowledgement

This work has been partially supported by the EU (FEDER) and the State Research Agency (AEI) of the Spanish Ministry of Economy and Competition under grants TIN2016-76843-C4-2-R (project MERINET), TIN2013-44742-C4-3-R (project CAVI-ART) and Madrid Regional Administration under grants P2018/TCS-4339 (BLOQUES-CM) and S2013/ICE-2731 (N-GREENS Software-CM).

References

[1] S. Banerjee, T. Pedersen, Extended gloss overlaps as a measure of semantic relatedness, in: Proc. of the IJCAI, Morgan Kaufmann, 2003, pp. 805–810.

[2] A. Budanitsky, G. Hirst, Evaluating wordnet-based measures of lexical semantic relatedness, Computational Linguistics 32 (1) (2006) 13–47.

[3] C. Fellbaum, WordNet: An Electronic Lexical Database, MIT Press, 1998.

[4] C. Fellbaum, WordNet(s), in: K. B. E. in Chief) (Ed.), Encyclopedia of Language & Linguistics, Second Edition, Vol. 13, Elsevier, Oxford., 2006, pp. 665–670.

[5] C. Fellbaum *et. al.*, WordNet File Formats: prologdb (5WN), at: `https://wordnet.princeton.edu/documentation/prologdb5wn` (2006).

[6] G. Hirst, D. St-Onge, Lexical chains as representations of context for the detection and correction of malapropisms, in: WordNet: An electronic lexical database, MIT Press., 1998, pp. 305–332.

[7] J. J. Jiang, D. W. Conrath, Semantic similarity based on corpus statistics and lexical taxonomy, in: Proc. of the ROCLING Intl. Conference and (ACLCLP), 1997, pp. 19–33.

[8] P. Julián-Iranzo, C. Rubio-Manzano, A Sound and Complete Semantics for a Similarity-based Logic Programming Language, Fuzzy Sets and Systems (2017) 1–26.

[9] C. Leacock, M. Chodorow, Combining local context and wordNet similarity for word sense identification, in: WordNet: An electronic lexical database, MIT Press., 1998, pp. 265–283.

[10] D. Lin, An Information-Theoretic Definition of Similarity, in: Proc. of the Intl. Conf. on Machine Learning, Morgan Kaufmann, 1998, pp. 296–304.

[11] G. A. Miller, WordNet: A Lexical Database for English, Commun. ACM 38 (11) (1995) 39–41.

[12] S. Patwardhan, Incorporating Dictionary and Corpus Information into a Context Vector Measure of Semantic Relatedness, Tech. rep., Master's thesis, Univ. of Minnesota, Duluth (2003).

[13] T. Pedersen, S. Patwardhan, J. Michelizzi, WordNet::Similarity - Measuring the Relatedness of Concepts, in: Proc. of the Nat. Conf. on AI and Conf. on Innovative Applications of AI, AAAI Press / The MIT Press, 2004, pp. 1024–1025.

[14] P. Resnik, Using information content to evaluate semantic similarity in a taxonomy, in: Proc. of the IJCAI 95, 2 Volumes, Morgan Kaufmann, 1995, pp. 448–453.

[15] C. Rubio-Manzano, P. Julián-Iranzo, Fuzzy Linguistic Prolog and its Applications, Journal of Intelligent and Fuzzy Systems 26 (2014) 1503–1516.

[16] S. Witzig, Accessing wordnet from prolog, available at Prolog Natural Language Tools: `http://ai1.ai.uga.edu/mc/pronto/` (2003).

[17] Z. Wu, M. S. Palmer, Verb semantics and lexical selection, in: Proc. of the Annual Meeting of the ACL, Morgan Kaufmann Publishers / ACL, 1994, pp. 133–138.