The final paper corresponding to this preprint has been published in:

JULIÁN-IRANZO P, SÁENZ-PÉREZ F. Implementing WordNet Measures of Lexical Semantic Similarity in a Fuzzy Logic Programming System. Theory and Practice of Logic Programming. 2021;21(2):264-282. doi:10.1017/S1471068421000028.

Implementing WordNet Measures of Lexical Semantic Similarity in a Fuzzy Logic Programming System∗

PASCUAL JULIÁN-IRANZO

Dept. of Information Technologies and Systems, University of Castilla-La Mancha,
13071 Ciudad Real, Spain
(*e-mail:* Pascual.Julian@uclm.es)

FERNANDO SÁENZ-PÉREZ

Faculty of Computer Science, Complutense University of Madrid,
28040 Madrid, Spain
(*e-mail:* fernan@sip.ucm.es)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

In this paper we provide techniques to integrate WordNet into a Fuzzy Logic Programming system. Because WordNet relates words but does not give graded information of the relation between them, we have implemented standard similarity measures and new directives allowing for generating the 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. This addition widens the scope of Fuzzy Logic Programming, allowing certain forms of lexical reasoning, and reinforcing Natural Language Processing applications.

KEYWORDS: Fuzzy Logic Programming, WordNet, Proximity Equations, System Implementation

1 Introduction and Motivation

Fuzzy Logic Programming (Lee 1972) integrates concepts coming from fuzzy logic (Zadeh 1965) into logic programming (van Emden and Kowalski 1976) in order to deal with the essential vagueness of some problems by using declarative techniques. In recent years there has been a renewed interest on this field, with multiple lines of work. When the fuzzy unification algorithm is weakened using a similarity relation (i.e., a reflexive, symmetric, transitive, fuzzy binary relation) the approach is usually named *Similarity-based Logic Programming* (Fontana and Formato 1999; Fontana and Formato 2002; Loia et al. 2001; Sessa 2002).

We have extended Similarity-based Logic Programming by introducing new theoretical concepts and developing two fuzzy logic programming systems: Bousi∼Prolog (BPL for short) (Rubio-Manzano and Julián-Iranzo 2014; Julián-Iranzo and Rubio-Manzano 2017) and Fuzzy-DES (Julián-Iranzo and Sáenz-Pérez 2017; Julián-Iranzo and Sáenz-Pérez 2018b). Their syntax

∗ Work partially funded by the Spanish Ministry of Economy and Competitiveness, under the grants TIN2016-76843-C4-2-R (MERINET), TIN2017-86217-R (CAVI-ART-2), and by the Madrid Regional Government, under the grant S2018/TCS-4339 (BLOQUES-CM), co-funded by EIE Funds of the European Union.

is based on the clausal form, and they embody a *Weak SLD (WSLD) resolution* operational semantics which uses a fuzzy unification algorithm based on the concept of *proximity relation* (i.e., a fuzzy binary relation supporting unification that, although reflexive and symmetric, is not necessarily transitive) (Julián-Iranzo and Rubio-Manzano 2015; Julián-Iranzo and Sáenz-Pérez 2018a). A proximity relation is defined by *proximity equations*, denoted as $a \sim b = \alpha$, whose intuitive reading is that two constants (either n-ary function symbols or n-ary predicate symbols), a and b, are approximate or similar with a certain degree α.

For instance, assume a deductive database that stores information about people and their family relationships coded using the Bousi∼Prolog language.

```
%% PROXIMITY EQUATIONS
ancestor~ascendant=1.0.     ancestor~progenitor=0.9.

%% FACTS
father(abraham,isaac).      father(isaac,esau).      father(isaac,jacob).
mother(sara,isaac).         mother(rebeca,jacob).    mother(rebeca,esau).

%% RULES
direct_ancestor(X,Y) :- father(X,Y); mother(X,Y).

ancestor(X,Z) :- direct_ancestor(X,Z).
ancestor(X,Z) :- direct_ancestor(X,Y), ancestor(Y,Z).
```

In a standard Prolog system (without proximity equations), asking the progenitors of `isaac` with the query `progenitor(X,isaac)` produces no answer. However, Bousi∼Prolog answers `X=abraham with 0.9` and `X=sara with 0.9` thanks to its proximity-based unification algorithm. Since we have specified that `progenitor` is close to `ancestor` with degree 0.9, these two terms can "weakly" unify with approximation degree 0.9, leading to a refutation.

Here, the proximity equations are axiomatically given by the programmer. It would be interesting that the system could provide assistance through its connection to a lexical resource such as WordNet (Fellbaum 1998; Fellbaum 2006; Miller 1995). So, this work deals with generating the set of proximity equations both automatically and with a minimal intervention of the programmer. We integrate this and the implementation of several relatedness measures and built-in predicates, that provides the ability of reasoning with linguistic terms in the readily available system Bousi∼Prolog. Because of the declarative nature of this system, we used Prolog technology to face these tasks and the integration with WordNet. Usefulness of this proposal include applications such as text mining in information retrieval, text classification, and even sentiment analysis (Allahyari et al. 2017; Baeza-Yates and Ribeiro-Neto 2011; Serrano-Guerrero et al. 2015).

2 The Lexical Resource WordNet and Prolog

WordNet is a lexical English language database that stores words of five syntactic categories: nouns, verbs, two kinds of adjectives, and adverbs. Words of the same syntactic category 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. 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 such structures: nouns, as well as verbs, are interconnected by the *hyponymy* relation (IS-A relation), which links specific concepts to more general ones.[1] Hypernymy is the opposite relation, that is, a hypernym is a word whose meaning includes a group of other words. Both relations are transitive. Note also that, both nouns and verbs are organized as separate hierarchical structures.

WordNet can be accessed either 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 fuzzy 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 WordNet 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 (Fellbaum and *et. al.* 2006). In the following we describe the predicates which are most interesting for the present work.

The file `wn_s.pl` contains all the information about words stored in WordNet. 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, if present, 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) 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 least 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.

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.

3 WordNet and Lexical Semantic Similarity

WordNet relates words but does not provide their grade of relationship. Measuring lexical semantic relatedness has many applications for Natural Language Processing (NLP) and its integration in a fuzzy logic programming system such as Bousi∼Prolog is very appropriate because of its similarity-based operational semantics. The syntax of our language uses symbols (words) that are endowed with a fuzzy semantics via proximity equations. So, we are interested in techniques to measure the grade of similarity between words in order to facilitate the construction of proximity equations with linguistic criteria. Semantic similarity quantifies how much two words are alike (or more precisely: how similar are the concepts they denote).

[1] A verb is a hyponym of another verb if it represents a (kind of) activity of the other.

WordNet is well suited for similarity measures because it organizes nouns and verbs into hyponymy/hypernymy-based hierarchies of concepts (synsets). Since the different parts of speech are isolated in distinct graph structures, they cannot be compared using this type of measures based on path lengths in that structures. Specifically, similarity measures in WordNet are limited to noun pairs and verb pairs.

Although a large variety of measures of semantic relatedness and similarity have been proposed (Budanitsky and Hirst 2006), only a limited number of tools have been implemented to perform this task. Perhaps, WordNet::Similarity (Pedersen et al. 2004) is the most prominent. This tool has three similarity measures based on path lengths between concepts (PATH, WUP (Wu and Palmer 1994) and LCH (Leacock and Chodorow 1998)), and another three based on information content (RES (Resnik 1995), JCN (Jiang and Conrath 1997) and LIN (Lin 1998)).

Table 1 summarizes some features of these measures, where its first column contains the measure name, the second its type (either Similarity or Information Content), the third its description, the fourth its range, and the last one whether it preserves block structure.[2] In order to understand the description of similarity measures precisely, we introduce the following standard definitions and notations used when working in the framework of WordNet:

- We differentiate between "words" and "concepts". We use the term "*word*" as a shorthand of "word form," and the term "*concept*" (i.e., "synset") to refer to a specific sense or word meaning. Words will be denoted by the letter w, and concepts by the letter c, possibly with subscripts. A word w with sense s is denoted as $w:s$.
- Similarity measures use the so-called HyperTrees (Hypernym Trees). They are IS-A hierarchies. 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., $depth(c) = len(root,c)$.
- The *least common subsumer* (LCS) of two concepts c_1 and c_2 is the most specific concept they share as an ancestor. It is denoted by $lcs(c_1,c_2)$.

RES, JCN and LIN measures are based on the notion of *Information Content* (IC) (Resnik 1995). For a concept c, $IC(c) = -ln(p(c))$, where p is the probability of encountering an instance of the concept c in a corpus. In our case, this probability is measured in terms of a relative frequency[3] of use of the concept c stored in WordNet. It is computed by adding the `Tag_count` of the concepts subsumed by the concept c. Then:

$$p(c) = Frequency(c)/Frequency(Root)$$

where *Root* is the concept (virtual or not) on the top of the concept hierarchy.

Section 5 will show how to integrate WordNet and the aforementioned lexical semantic similarity measures into the state-of-the-art fuzzy logic programming system Bousi∼Prolog. But before going on, we briefly summarize the architecture of this system and the role and place allocated to the modules that make the integration with WordNet effective.

[2] We have noticed that some measures have the ability to group concepts in blocks of proximity respecting certain semantic similarity groupings that human users perform naturally. The concept of proximity block is a technical concept described in (Julián-Iranzo and Rubio-Manzano 2015; Julián-Iranzo and Sáenz-Pérez 2018a). A proximity block is a maximal clique of the proximity relation. Only the symbols in the same block are considered "equal" and, therefore, can be unified.

[3] Also named *frequency count*, which is a measure of the number of times that an item or event occurs in a particular context.

Table 1. *Some relatedness measures and their features.*

Msr.	Type	Description	Rng.	Blk.
PATH	Sim.	$sim_{PATH}(c_1,c_2) = 1/len(c_1,c_2)$	$[0,1]$	Yes
WUP	Sim.	$sim_{WUP}(c_1,c_2) = \frac{2\times depth(lcs(c_1,c_2))}{Depth(c_1)+Depth(c_2)}$	$[0,1]$	Yes
LCH	Sim.	$sim_{LCH}(c_1,c_2) = -log(\frac{len(c_1,c_2)}{2\times \max\{depth(c)\mid c\in \text{WordNet}\}})$	$[0,\infty]$	Yes
RES	IC	$sim_{RES}(c_1,c_2) = IC(lcs(c_1,c_2))$	$[0,\infty]$	Yes
JCN	IC	$sim_{JCN}(c_1,c_2) = \frac{1}{IC(c_1)+IC(c_2)-2\times IC(lcs(c_1,c_2))}$	$[0,\infty]$	No
LIN	IC	$sim_{LIN}(c_1,c_2) = \frac{2\times IC(lcs(c_1,c_2))}{IC(c_1)+IC(c_2)}$	$[0,1]$	No

4 The architecture of the Bousi∼Prolog system and the wn-connect subsytem

Bousi∼Prolog can be downloaded from `dectau.uclm.es/bousi-prolog/downloads` (sources and binaries). Sources also include instructions to build binaries, and in particular, the `wn-connect` subsystem is stored inside the directory `wn` since Bousi∼Prolog version 3.3.

The Bousi∼Prolog system is composed of three subsystems with a total of nine modules. Figure 1 shows the current structure of the system and, in broad strokes, the modules that integrate each of its components, focusing on a simplified view of the `wn-connect` layered structure.

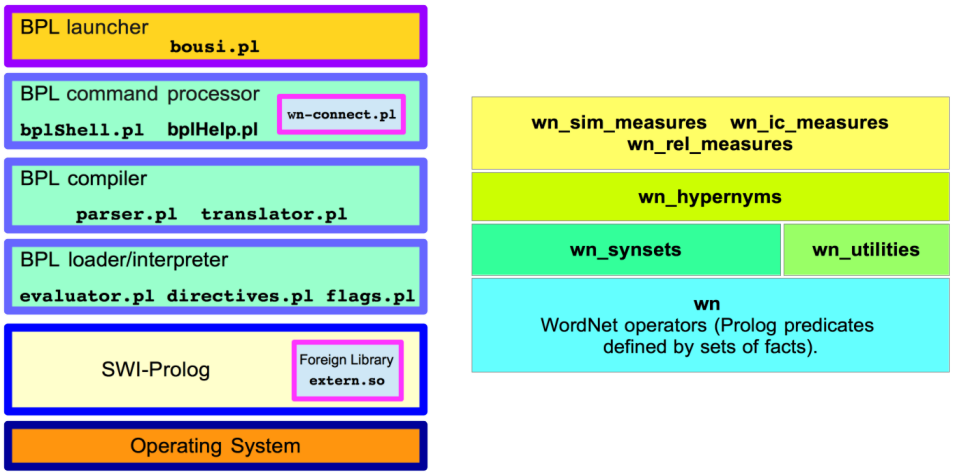

Fig. 1. Architecture of the Bousi∼Prolog system and the wn-connect subsystem.

The `wn-connect` subsystem provides the basis to the connection between WordNet and the Bousi∼Prolog system, and relevant parts of its implementation and their features will be de-

scribed in the following sections.[4] Actually, `wn-connect` is a software application in itself with ten Prolog modules, which implement predicates for managing synsets, hypernyms and hyponyms that give support to the `wn_sim_measures` and `wn_ic_measures` modules which, in addition, implement the standard similarity measures defined in former Section 3. The base modules are summarily described next:

- The `wn_synsets` module implements predicates to retrieve information about words and synsets stored in WordNet. It uses the `wn` module implemented by Jan Wielemaker[5] which exploits SWI-Prolog demand-loading and Quick Load Files (QLF) for 'just-in-time' fast loading.
- The `wn_hypernyms` module implements predicates to retrieve information about hypernyms of a concept (synset). Part of its implementation is detailed in Subsection 5.1. It uses the modules `wn_synsets` and `wn_utilities`. Notably, the predicate `wn_hypernyms/2` returns a list `List_SynSet_HyperNym` of hypernyms (as synset identifiers) for a word term `Hyponym`.
- The `wn_hyponyms` module implements predicates to retrieve information about hyponyms of a concept (synset). Remarkably, 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.

5 Integrating WordNet into Bousi∼Prolog

This section describes the implementation of some of the most relevant pieces in the integration of WordNet and Bousi∼Prolog.

5.1 Implementing Similarity Measures

A first step for a more ambitious goal is to automatically extract semantic similarity information from WordNet IS-A hierarchies, and other attributes as the frequency of use as explained before in Section 3. Here we show the implementation of the WUP similarity measure as a specific example and some insights about the implementation of the similarity measures based on information content.

To a greater or lesser extent, all measures of similarity are based on the computation of the LCS of words `Word1` and `Word2` (actually, two concepts). The predicate `wn_sim_measures:lcs/6` returns the LCS and also measures its depth from the root of the HyperTree. Roughly speaking, it computes the HyperTrees of `Word1` and `Word2` and compares them, returning the `synset_ID` previous to the first mismatch (which is the LCS of both concepts). Additionally, the predicate `lcs/6` returns the depth of `Word1` and `Word2` because of efficiency reasons: we want to go through the hypernym lists only once, so these quantities are calculated while calculating the LCS.

Computing the hypernyms of a concept is performed by the predicate `wn_hypernyms:hyper`

[4] In (Julián-Iranzo and Sáenz-Pérez 2019) a more detailed description of this subsystem from the user's point of view is available.

[5] Available at: `https://github.com/JanWielemaker/wordnet`.

`nym_chain/2`, which computes a list (`SynSet_HyperNyms`) of `synset_IDs` designating the hypernyms of a concept (`SynSet_Hyponym`). So, it computes a HyperTree (i.e., a chain of hypernyms) that will be used in the former comparison to compute the LCS. It is implemented as follows:

```
% hypernym_chain(+SynSet_Hyponym, -SynSet_HyperNyms)
%
hypernym_chain(SynSet_Hyponym, SynSet_HyperNyms):-
hypernym_chain(SynSet_Hyponym, [SynSet_Hyponym], SynSet_HyperNyms).

hypernym_chain(SS_Hyp, SS_Hyper_Accs, SS_Hypers) :-
        (wordnet:wn_hyp(SS_Hyp, SS_Hyper),
         hypernym_chain(SS_Hyper, [SS_Hyper|SS_Hyper_Accs], SS_Hypers)
         ;
         \+ wordnet:wn_hyp(SS_Hyp, _SS_Hyper),
         SS_Hypers = SS_Hyper_Accs).
```

Once the depth of the LCS and the words to be compared are known, it is easy to compute the relationship degree between them by following the guidelines given in Section 3. For example, the WUP measure is implemented by the predicate `wn_wup/3` which takes two concepts (expressed as terms of the form `Word:SS_type:Sense_num`) and returns the degree of similarity between them. It relies on the private predicate `wup/3` that calls `lcs/6` to generate and inspect a pair of HyperTrees associated to `Word1` and `Word2`, and obtains the degree of similarity between `Word1` and `Word2` (according to that pair of HyperTrees):

```
% wup(+W1:T:S1, +W2:T:S2, -Degree)
%
wup(W1:T:S1, W2:T:S2, Degree) :-
                lcs(W1:T:S1,W2:T:S2,_,LCS_depth,DepthW1,DepthW2),
                Degree is (2 * LCS_depth / (DepthW1 + DepthW2)).
```

Because a concept can have more than one HyperTree, several pairs of HyperTrees are possibly considered, and a list of similarity degrees `Degrees` is obtained using `wup/3` on each of these pairs of HyperTrees. Finally, the maximum degree in the list `Degrees` is selected as a result, which is actually what `wn_wup/3` returns. This is implemented by the predicate `max_wup/3`, where `findall/3` is responsible for computing the aforementioned list of similarity degrees `Degrees`, and the private predicate `max_list`, for selecting the maximum degree in `Degrees`:

```
% max_wup(+W1:T:S1, +W2:T:S2, -Degree)
%
max_wup(W1:T:S1, W2:T:S2, Degree) :-
                findall(Deg, wup(W1:T:S1, W2:T:S2, Deg), Degrees),
                max_list(Degrees, Degree).S
```

Regarding similarity measures based on the information content, the key idea lies in the implementation of the notion of frequency of use. The operator `wn_s/6` stores information on how common a word is. The `tag_number` indicates the number of times the word was found in a test corpus. Therefore, the higher the number, the more common the word is. So, this parameter can be employed to obtain the use of a word and, summing the `tag_number` of all words in a synset, the specific use associated to a whole synset (i.e., to a concept) can be obtained. The predicate `synset_tag_num/2` implements this:

```
% synset_tag_num(+Synset_ID, -Frequency)
%
synset_tag_num(Synset_ID, Frequency) :-
```

```
        integer(Synset_ID),
        Synset_ID > 100000000,
        Synset_ID < 500000000,
        findall(Tag_num, wn_s(Synset_ID,_,_,_, _,Tag_num), W_freqs),
        sum(W_freqs, Frequency).
```

where `sum/2` adds the word frequencies obtained by `findall/3`. Then, the frequency of use of a concept is obtained by adding the "synset tag num" of all concepts subsumed by that concept:

```
% frequency_of_use(+Synset_ID, -Frequency),
%
frequency_of_use(Synset_ID, Frequency) :-
        gen_all_hyponyms_of(Synset_ID, All_Hyponym_IDs),
        sum_synset_tag_num([Synset_ID|All_Hyponym_IDs], Frequency).
```

In general, to compute all hyponyms of a concept is an expensive process. Therefore, we have implemented the predicate `gen_all_hyponyms_of/2` using Definite Clause Grammar rules which provide a neat interface to difference lists. Difference lists allows for efficiently concatenating lists, an operation intensively used by `gen_all_hyponyms_of/2`.

As explained in Section 3, the information content of a concept is a function of the ratio between the frequency of use of that concept and the frequency of use of the root concept of the hierarchy. Finally, the information content based measures are computed as indicated in Table 1 by specific predicates. Note that we have taken the option of smoothing the frequencies of use with a value of 0, which we substitute for a very small number close to 0. So, some relationship values do not exactly match to those that would be obtained when using tools like `wordnet::similarity` (Pedersen et al. 2004).

5.2 Directives for Generating Proximity Equations

Bousi∼Prolog can load both ontologies (consisting of proximity equations) and fuzzy logic programs (with fuzzy logic rules and possibly proximity equations). Thus, it would be of interest to use the similarity measures implemented in the last section to automatically construct such ontologies.

To define the semantic similarity between selected concepts, we provide a Bousi∼Prolog directive for automatically generating the proximity equations which define an ontology:

- `:-wn_gen_prox_equations(+Msr, +LL_of_Pats)`
  where `Msr` is the similarity measure which can be any of: `path`, `wup`, `lch`, `res`, `jcn` and `lin`. The second argument `LL_of_Pats` 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 the structure `Word:Type:Sense`, where `Word` is the word, `Type` is its type (either `n` for noun or `v` for verb),[6] and `Sense` is the sense number in its synset. If the pattern is simply a word, then a sense number of 1 is assumed, and its type is made to match to all other words in the same list.

An example of this directive is:

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

[6] Note that, because similarity measures only relate nouns with nouns and verbs with verbs, the words of a set must be of the same part of speech.

In this case, as only words are provided in the first list, the sense number is 1, and their types are equal two by two (nouns for these words). The second list explicitly specifies the pattern of each word to be related. Then, excluding for simplicity reflexive and symmetric entries, the following proximity equations are generated for a lambda cut of 0:

```
sim(man,    human,  1, 0.56).
sim(man,    person, 1, 0.8888888888888888).
sim(person, man,    1, 0.8888888888888888).
sim(human,  person, 1, 0.6086956521739131).
sim(grain,  wheat,  0, 0.2608695652173913).
```

Note that there are two blocks, numbered with 1 for the first four equations, and with 0 for the last one. Clearly, words in the first list are not made to be related to the ones in the second list and therefore they must occur in different blocks. In addition, proximity equations are generated only for the words stored in WordNet.

Another form of this directive automatically builds an ontology in terms of the tokens in the BPL program:

- `:-wn_gen_prox_equations(+Msr, +Automatic)`
  where the argument `Automatic` is the constant `auto`.

Finally, only the symbols that occur in a program are related because it would not be practical to relate the symbols of the program with all that occur in WordNet. That would lead to an undesirable increase in the size of the program (in terms of growth in the number of proximity equations) adding, most likely, an unnecessary complexity.

5.3 Implementing the Generation of Proximity Equations

Bousi∼Prolog processes a file (either a program or an ontology) with the load command `ld` *`file`* of the BPL Shell module (named `bplShell`), where its argument is the name of the file to load (with default extension `bpl`). Upon execution of this command, a source file (*`file`*`.bpl`) is parsed, compiled to Prolog (*`file`*`.tpl`), and consulted.

The implementation of this procedure is as follows: When executing a command `ld` *`file`*, the predicate `bplShell:load_file` is called, which in turn calls `bplShell:load_bpl`, and this predicate calls `translator:translate_program`. This last predicate firstly calls the parser with `parser:parse_program` and, after some other required tasks, it firstly generates proximity equations and then their corresponding blocks (all these as in-memory data structures). After the translation of rules (with `expand_rules`, which requires to know the just computed proximity equations), the output file *`file`*`.tpl` is written. Finally, this output file is consulted as a normal Prolog program. An advantage of having this file available is avoiding recompilations in further sessions.

When parsing a directive `:-wn_gen_prox_equations`, it is first checked for validity, and then replaced in the target Prolog file with the proximity equations corresponding to the pairs formed with the symbols derived from its arguments. As explained, there are two cases for this directive, and they are handled in a different way:

- Explicit indication of words to be related.
  Here, the proximity equations can be directly generated from each list of words, kept in memory (as asserted Prolog facts) and output to the translated program in the `.tpl` file at a later stage. The procedure is as it would be expected: for each pair of different

words `W1` and `W2` in a list, generate the proximity equation `sim(W1,W2,D)`, where `D` is the approximation degree for the normalized measure given as the first parameter of the directive. Normalization is required because measures are generally not in the interval $(0,1]$ that proximity equations range.

- Automatic generation of proximity equations.
  This case is different from the former because, when processing the directive, the rules in the program have not been parsed already, so their tokens are not available yet. Thus, it is processed after parsing the remaining program, by performing a syntactic analysis in order to extract the sets of constant, functor and predicate identifiers and adding the resulting proximity equations for each separated set of tokens (with the same shape as in the other case) to memory.

The directives that generate proximity equations are based on the private predicate `gen_prox_equation`. It generates a proximity equation `sim(Word1, Word2, NormalizedDegree)` in terms of a given measure (`Measure`) and a pair of words which can be completely specified with either a pattern `Word:Type:SenseNumber` or only with its syntactic form as plain words. In this last case, their first sense number is selected and the same word type is enforced:

```
% gen_prox_equation(+Measure, +Pattern1, +Pattern2, -Equation)
%
(1) gen_prox_equation(Measure, Pattern1, Pattern2,
                                sim(Word1, Word2, NormalizedDegree)) :-
(2)           wn_sim_measure_goal(Measure, Module, Goal),
(3)           MeasureGoal =.. [Goal, Pattern1, Pattern2, Degree],
(4)           Module:MeasureGoal,
(5)           measure_max_value(Measure, Max),
(6)           NormalizedDegree is Degree/Max,
(7)           Pattern1 = Word1:Type:_,
(8)           Pattern2 = Word2:Type:_.
```

Its first goal `wn_sim_measure_goal` in line `(2)` returns the name of the goal in a module to be executed for the given measure. Next, the goal is built in line `(3)` with the univ operator (`=..`), and it is executed in its corresponding module in line `(4)`. The maximum value the measure can take is retrieved in line `(5)`, which is used to compute the normalized degree in line `(6)`. Finally, lines `(7)` and `(8)` extract the words in the input patterns to build the proximity equation, forcing the same type.

An example of an actual call in line `(4)` is `wn_sim_measures:wn_wup(grain:n:8, wheat:n:2,Degree)` which computes the similarity of those words, in their specific meanings, using the WUP measure.

5.4 Accessing WordNet

The `wn_connect` subsystem must be made visible before using the built-ins (public predicates) defined in its modules. In Bousi∼Prolog, this can be done from within a program or interactively through the BPL command processor.

Accessing WordNet, and all the ancillary stuff for related predicates which are implemented by `wn_connect`, from a BPL program is via the following directive:

- `:-wn_connect.`
  This performs the necessary actions to connect Bousi∼Prolog to the WordNet resources. It

gives access to a wide repertoire of built-in predicates that allow to interact with WordNet. This directive must occur before any other access to WordNet, and it assumes that the environment variable `WNDB` has been set to the OS directory in which the WordNet database is stored.

The following goal enables interactive access to the `wn_connect` subsystem:

```
BPL> ensure_loaded(wn(wn_connect))
true.
```

Note also that nearly all the predicates implemented in the `wn-connect` subsystem are crisp, returning the top approximation degree. However, the binary similarity predicates (`wn_path/2`, `wn_wup/2`, `wn_lch/2`, etc.) are fuzzy predicates that return the similarity degree of two concepts. For instance:

```
BPL> wn_wup(lion:n:1, tiger:n:2).
true
With approximation degree: 0.9375 .
```

Note that in these cases we also maintain ternary predicates available to programmers, since they provide a direct access to the approximation degree `D`, which may be very useful for its explicit handling. For instance:

```
BPL> wn_wup(lion:n:1, tiger:n:2, D).
D = 0.9375.
```

Thanks to the repertoire of built-in predicates implemented in the `wn-connect` subsystem, the user of the BPL system can extract information from WordNet, deepening into the structure of the relationships between its linguistic terms. This becomes especially evident for the predicate `wn_display_graph_hypernyms/1`. For example, by submitting the goal `wn_display_graph_hypernyms(god)`, a graphical representation of the hypernym hierarchy of all senses of the word `god` (see Figure 2) is displayed.[7]

Moreover, with these built-in predicates, a certain form of linguistic reasoning is possible. For example, the predicate `wn_lcs/2`, which computes the LCS of a set of concepts, can help to obtain the most specific generalization of a set of concepts and to contribute to knowledge discovery. Also, the predicate `wn_gen_hyponyms_upto_level/3`, which generates all the hyponyms of a concept (`Synset_ID`) up to a certain depth level (`Level`), can be used to generate an ontology of close related terms to the given concept that can be used to implement flexible queries and text mining tasks.

6 Conclusions

We have presented techniques to embody the information stored in the lexical database WordNet (Fellbaum 1998; Fellbaum 2006; Miller 1995) into the fuzzy logic programming language Bousi∼Prolog (Rubio-Manzano and Julián-Iranzo 2014; Julián-Iranzo and Rubio-Manzano 2017; Julián-Iranzo and Sáenz-Pérez 2018a). However, the techniques developed can be used to connect WordNet to any logic programming language that uses an operational semantics based on some variant of WSLD resolution.

The main contributions of this work have been the following:

[7] In Figure 2, each node draws the representative word of the respective synset (i.e., those with `W_num` equal to one).

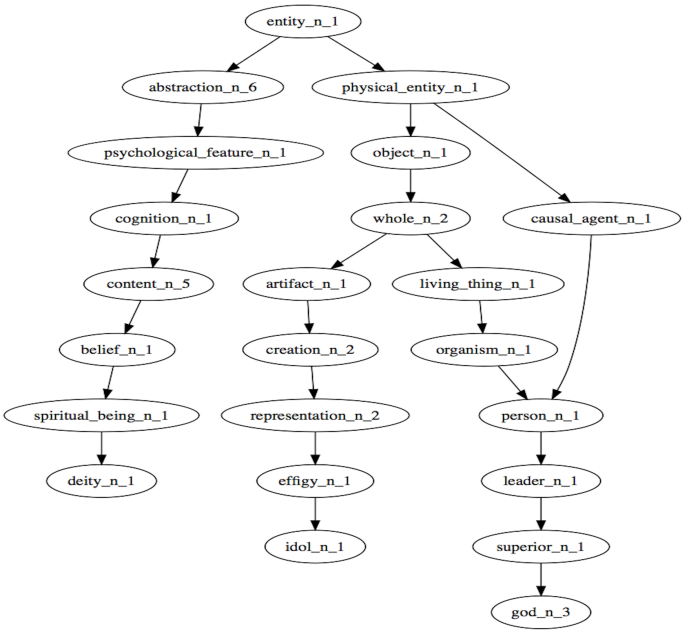

Fig. 2. Hypernyms of the word `god` (all senses).

1. We have implemented in Prolog all the usual repertoire of similarity measures (both those based on counting edges and those based on the information content of the concepts) that can be found in standard tools such as `wordnet::similarity` (Pedersen et al. 2004) and sets of words can be related according to them.
2. In order to give support to the implementation of these similarity measures, a whole BPL subsystem, named `wn-connect`, has been developed. It provides a number of built-in predicates that can be used to obtain common but useful information about words and synsets stored in WordNet. It is noteworthy that the predicates defined in the modules that compound `wn-connect` can also be consulted from a Prolog interpreter, providing an enhanced functionality to this system.
3. We have implemented several directives to generate proximity equations from a set of words, linking them with an approximation degree. Proximity equations are key syntactic structures that, in addition to a weak unification algorithm, allow for a flexible query answering process. Hence, the relevance of this work is to make possible a fuzzy treatment of concepts via proximity relations, and also endowing Bousi∼Prolog with linguistic characteristics.
4. We have developed a working system implementing these techniques which can be installed as a desktop application (for Windows, Mac and Linux OS's – `dectau.uclm.es/bousi-prolog`), and also used in an on-line web system (`dectau.uclm.es/bplweb`) which eases a first hands-on experience.

References

ALLAHYARI, M., POURIYEH, S., ASSEFI, M., SAFAEI, S., TRIPPE, E., GUTIERREZ, J., AND KOCHUT, K. 2017. A brief survey of text mining: Classification, clustering and extraction techniques. *CoRR abs/1707.02919*.

BAEZA-YATES, R. AND RIBEIRO-NETO, B. 2011. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England.

BUDANITSKY, A. AND HIRST, G. 2006. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics 32,* 1, 13–47.

FELLBAUM, C. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

FELLBAUM, C. 2006. WordNet(s). In *Encyclopedia of Language & Linguistics, Second Edition*, K. B. E. in Chief), Ed. Vol. 13. Elsevier, Oxford., 665–670.

FELLBAUM, C. AND *et. al.* 2006. Wordnet file formats: prologdb(5wn). available: `https://wordnet.princeton.edu/documentation/prologdb5wn`.

FONTANA, F. A. AND FORMATO, F. 1999. Likelog: A logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC'99)*. 260–267.

FONTANA, F. A. AND FORMATO, F. 2002. A similarity-based resolution rule. *Int. J. Intell. Syst. 17,* 9, 853–872.

JIANG, J. J. AND CONRATH, D. W. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the 10th Research on Computational Linguistics International Conference, ROCLING 1997, Taipei, Taiwan, August 1997*. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP), 19–33.

JULIÁN-IRANZO, P. AND RUBIO-MANZANO, C. 2015. Proximity-based Unification Theory. *Fuzzy Sets and Systems 262*, 21–43.

JULIÁN-IRANZO, P. AND RUBIO-MANZANO, C. 2017. A Sound and Complete Semantics for a Similarity-based Logic Programming Language. *Fuzzy Sets and Systems*, 1–26.

JULIÁN-IRANZO, P. AND SÁENZ-PÉREZ, F. 2017. FuzzyDES or how DES met Bousi Prolog. In *2017 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2017, Naples, Italy, July 9-12, 2017*. 1–6.

JULIÁN-IRANZO, P. AND SÁENZ-PÉREZ, F. 2018a. An Efficient Proximity-based Unification Algorithm. In *2018 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2018, Rio de Janeiro, Brazil, July 9-12, 2018*. 1–8.

JULIÁN-IRANZO, P. AND SÁENZ-PÉREZ, F. 2018b. A Fuzzy Datalog Deductive Database System. *IEEE Transactions on Fuzzy Systems 26*, 2634–2648.

JULIÁN-IRANZO, P. AND SÁENZ-PÉREZ, F. 2019. WordNet and Prolog: why not? In *The 11th Conference of the European Society for Fuzzy Logic and Technology, EUSFLAT 2019, Prague, September 9-13*. 1–8.

LEACOCK, C. AND CHODOROW, M. 1998. Combining local context and wordNet similarity for word sense identification. In *WordNet: An electronic lexical database*, C. Fellbaum, Ed. MIT Press., 265–283.

LEE, R. 1972. Fuzzy Logic and the Resolution Principle. *Journal of the ACM 19,* 1, 119–129.

LIN, D. 1998. An Information-Theoretic Definition of Similarity. In *In Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, 296–304.

LOIA, V., SENATORE, S., AND SESSA, M. I. 2001. Similarity-based SLD resolution and its implementation in an extended prolog system. In *FUZZ-IEEE*. 650–653.

MILLER, G. A. 1995. WordNet: A Lexical Database for English. *Commun. ACM 38,* 11, 39–41.

PEDERSEN, T., PATWARDHAN, S., AND MICHELIZZI, J. 2004. WordNet::Similarity - Measuring the Relatedness of Concepts. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*. AAAI Press / The MIT Press, 1024–1025.

RESNIK, P. 1995. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 448–453.

RUBIO-MANZANO, C. AND JULIÁN-IRANZO, P. 2014. Fuzzy Linguistic Prolog and its Applications. *Journal of Intelligent and Fuzzy Systems 26*, 1503–1516.

SERRANO-GUERRERO, J., OLIVAS, J. A., ROMERO, F. P., AND HERRERA-VIEDMA, E. 2015. Sentiment analysis: A review and comparative analysis of web services. *Inf. Sci. 311*, 18–38.

SESSA, M. I. 2002. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science 275,* 1-2, 389–426.

VAN EMDEN, M. AND KOWALSKI, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM 23,* 4, 733–742.

WU, Z. AND PALMER, M. S. 1994. Verb semantics and lexical selection. In *32nd Annual Meeting of the Association for Computational Linguistics, 27-30 June 1994, New Mexico State University, Las Cruces, New Mexico, USA, Proceedings.* Morgan Kaufmann Publishers / ACL, 133–138.

ZADEH, L. 1965. Fuzzy sets. *Information and Control 8*, 338–353.