Accessing WordNet from Prolog

Sarah Witzig
Artificial Intelligence Center
The University of Georgia
http://www.ai.uga.edu
sarah_witzig@hotmail.com

2003 May 8

Abstract

WordNet is a lexical reference system, developed by the university of Princeton. This paper gives a detailed documentation of the Prolog database of WordNet and predicates to interface it. [1]

1 Introduction

WordNet is a lexical reference system, developed by the university of Princeton. WordNet 1.7.1 is the most recently available version of the software. It can be downloaded from the Internet. Its design makes the use of dictionaries more convenient. You do not have to spend a lot of time going through the alphabetical list of words to find information about an expression. With WordNet, you type in what you are looking for, and you get all its possible meanings.

Moreover, data from WordNet can be used as input for various applications. It provides a database, written in Prolog. This paper documents the Prolog database of WordNet. It will also provide a few ideas that utilize the database efficiently.

2 Documentation of WordNet from a Prolog programmer's point of view

2.1 Basic concept

WordNet is based on a concept called *synsets*, also known as synonym sets. A synset is a group of words, connected by meaning. Only words of the same part of speech can belong

[1] Special thanks to Dr. Covington for his guidence on the project. Special thanks to Abhishek Jain and John Burke for proofreading this paper.

to the same synset. A synset ID is assigned to every word. Words in the same synset have the same synset ID. As one word can have several meanings, it can belong to more than one synset. Then, the word gets several entries in the Prolog database, and each entry has a different synset ID assigned.

The Prolog clauses, which store this basic information, are defined in the file **wn_s.pl**. Each clause contains one word, an assigned synset ID, and some additional information.

There are 15 other **wn_** *operator***.pl** files, where *operator* corresponds to several different relations in WordNet. For example, the file **wn_s.pl** organizes the synsets, accordingly its operator *s* is derived from the word *synset*.

The relations, described by the operators, are either of semantic or lexical nature. A lexical relation describes a connection between lexical units, say between words. Grouping words in synonym sets is a lexical relation. A semantic relation describes a connection between meanings. For example, *hypernymy* is a semantic relation of *being superordinate or belonging to a higher rank or class* (Fellbaum et al. 2003e).

2.2 The Prolog Files

2.2.1 wn_s.pl

One of the main files is **wn_s.pl**, which stores the synset information of the WordNet corpus. Each word has an entry by the six place predicate

```
s(synset_ID,w_num,'word',ss_type,sense_number,tag_count).
```

The first argument is the 9-digit `synset_ID`, indicating to which synset does the word belong. As discussed before, words belonging to the same synset are synonyms, which signifies similarity in their meaning. The `synset_ ID` encodes information about the syntactic category of the synset. Synset IDs starting with `1` contain only nouns, synset IDs starting with `2` store the verbs. A `3` in the beginning indicates an adjective, a `4` an adverb. The remaining eight digits, called *synset offset*, identify a specific synset.

The second argument `w_num` allows addressing one word in a synset. The words in a synset are numbered serially, starting with one. For example, the knowledge base for the synset `100041682` looks like this:

```
s(100041682,1,'close_call',n,1,0).
s(100041682,2,'close_shave',n,1,0).
s(100041682,3,'squeak',n,2,0).
s(100041682,4,'squeaker',n,2,0).
s(100041682,5,'narrow_escape',n,1,1).
```

The third argument is the word itself, written in single quotes.

The `ss_type` stores the synset type. In WordNet the synset categories are limited to nouns, verbs, adverbs and adjectives. You can not find any pronouns, conjunctions, prepositions or interjections. The `ss_type` instantiated to the letter `n`, indicates the particular word is a noun, the letter `v` refers to a verb, and the letter `r` to an adverb. Adjectives are divided into adjectives and adjective satellites. 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. The two letters `a` and `s` indicate whether a word is an adjective or an adjective satellite.

The fifth argument is the `sense_number`. It gives information about how common a word is. Words within one part of speech are ordered from most to least frequently. The higher the `sense_number`, the less common is the word. For example, here are two entries for the word *mouse*, the first one refers to the animal, the second to the computer device.

```
s(101993048,1,'mouse',n,1,14).
s(103304722,1,'mouse',n,2,0).
```

According to WordNet the word *mouse*, referring to the animal, has the `sense_number 1`. Therefore, it is more common than the word *mouse*, referring to the computer device with `sense_number 2`.

The last argument is the `tag_count`. This number indicates how common a word is in relation to a text. The number is equal to the times the word was found in a test corpus. Therefore the higher the number, the more common the word.

2.2.2 wn_g.pl

The file **wn_g.pl** stores a gloss for every synset. The gloss may contain an explanation, definition and example sentences.

The two place predicate `g` has the following structure:

```
g(synset_ID,'(gloss)').
```

The first argument labels the `synset_ID`, the second argument allocates a gloss, written in parenthesis and single quotes. For example, the synset with the ID `100031541` consists of the two words *walking* and *sledding*. The matching `g` clause gives us a definition for the two words.

```
g(100031541, '(advancing toward a goal; ''persuading him was easy going'';
             ''the proposal faces tough sledding'')').
```

2.2.3 wn_hyp.pl

`Hyp` is short for hypernym. A hypernym is a word that is more generic than a given word. Only verbs and nouns can have hypernyms. For example, *mammal* and *animal* are hypernyms of the word *dog*. Hypernymy is a relation between synsets. Thus, it is a semantical relation.

The file **wn_hyp.pl** stores hypernym relations in the two place predicate

```
hyp(synset_ID_1,synset_ID_2).
```

The synset corresponding to the second argument `synset_ID_2` is a hypernym of the synset corresponding to the first argument `synset_ID_1`. For example, `101752990` is the `synset_ID` of the word *dog*. Consulting the file **wn_hyp.pl** we find

```
hyp(101752990, 101752283).
```

The synset `101752283` consists of hypernyms of the word *dog*, e.g. *canine*. Bear in mind, a hypernym of a hypernym of a word, is also a hypernym of the word. This gives you the possibility of finding hypernym chains.

In the section *Working with WordNet* the predicates `find_hyp/3` and `find_hyp_chains/2` will be introduced to experiment with the relation.

2.2.4 wn_ent.pl

`Ent` is short for entailment. An entailment is a verb describing an event that facilitates another event. Therefore only verbs can be described by the relation entailment. For example, *to sleep* is an entailment of *to snore*, as you need to sleep if you snore. The file **wn_ent.pl** stores all entailment relations. Like hypernyms entailments are semantical relations.

The predicate `ent/2` has the same structure as the predicate `hyp/2`.

```
ent(synset_ID_1,synset_ID_2).
```

`Synset_ID_2` contains verbs that are entailments of the verbs in `synset_ID_1`.

In the section *Working with WordNet* the predicates `find_ent/3` and `find_ent_chains/2` will be introduced to experiment with the relation.

2.2.5 wn_sim.pl

`Sim` is an abbreviation for similar meaning. This relation allocates adjectives that have similar meanings. Again, the relation connects synsets, thus it is semantical.

Like `hyp/2` and `ent/2`, `sim/2` takes two synset IDs as arguments.

```
sim(synset_ID_1,synset_ID_2).
sim(synset_ID_2,synset_ID_1).
```

As similarity works in both directions, there is a reflexive predicate defined for every clause. The two addressed synsets are either two head synsets, or one head synset and one satellite synset. There is no matching `sim` clause for two satellite synsets. Because, if they would have similar meanings, they would be grouped together in one synset.

For example, one satellite synset contains the synonyms *devious*, *circuitous* and *roundabout*. By consulting `sim/2`, we get one head synset with similar meaning, consisting of the word *indirect*.

In the section *Working with WordNet* the predicates `find_sim/2` and `find_sim_meaning/1` will be introduced to experiment with the relation.

2.2.6 wn_mm.pl,wn_ms.pl,wn_mp.pl

The three file extensions `mm`, `ms` and `mp` refer to the semantic meronym relation, also called the part-whole relation. A word X is a meronym of a word Y, if you can apply the sentence *A Y has an X* or *An X is a part of a Y* (Beckwith et al. 1993:8). This relation only holds for nouns. The reflexive relation is called holonym relation.

The extension mm refers to the member meronym relation, also called the member-group relation. For example, the word *person* is a member meronym of the word *faculty*, as we can say *A faculty is a group of persons* or *A person is a member of a faculty*. Applying the reflexive relation, *faculty* is a holonym of the word *person*. Accordingly, the predicate of the **wn_mm.pl** file has the following structure:

```
mm(synset_ID_1,synset_ID_2).
```

The words of `synset_ID_1` are member meronyms of the words of `synset_ID_2`. Reflexive, the words of `synset_ID_2` are member holonyms of the words of `synset_ID_1`.

Note, that the first `synset_ID` contains meronyms of words of the second `synset_ID`, not the other way round. This is explained incorrectly in the original Prolog documentation of WordNet (Fellbaum et al. 2003a).

The extension ms refers to the substance meronym relation. For example, the word *water* is a substance meronym of the word *tear* as we can say *A tear has water as a substance* or *Water is a substance of tears*. Applying the reflexive relation, *tear* is a substance holonym of the word *water*. The predicate has exactly the same structure as `mm/2`, only the operator changed.

```
ms(synset_ID_1,synset_ID_2).
```

The words of `synset_ID_1` are substance meronyms of the words of `synset_ID_2`. Reflexive, the words of `synset_ID_2` are substance holonyms of the words of `synset_ID_1`.

The extension mp refers to the part meronym relation. For example, the word *leg* is a part meronym of the word *table*, as we can say *A table has a leg as a part* or *A leg is a part of a table*. Also, the reflexive relation, *table* being a holonym of *leg*, is valid. The structure and unification of the predicate is the same as `mm/2` and `ms/2`.

```
mp(synset_ID_1,synset_ID_2).
```

The words of `synset_ID_1` are part meronyms of the words of `synset_ID_2`. Reflexively, the words of `synset_ID_2` are part holonyms of the words of `synset_ID_1`.

In the section *Working with WordNet* several predicates will be introduced to experiment with these relations.

2.2.7 wn_cs.pl

The file **wn_cs.pl** describes a semantic relation for verbs. The predicate

```
cs(synset_ID_1,synset_ID_2).
```

takes a `synset_ID_1` as the first argument and assigns a `synset_ID_2` that contains verbs that are caused by the action of the verbs of `synset_ID_1`. For example, the `synset_ID` `200017177`, containing the verb *to anesthetize*, unifies with the predicate

```
cs(200017177,200011887).
```

The `synset_ID 200011887` contains the verbs *to kip*, *to sleep*, *to slumber*, which are all events caused by the event of the verb *to anesthetize*.

The predicate `cause/2`, defined in *Working with WordNet*, takes a verb and gives out a list of verbs connected by this relation.

2.2.8 wn_vgp.pl

The file **vgp.pl** handles lexical relations between verbs. The four place predicate

```
vgp(synset_ID_1,w_num_1,synset_ID_2,w_num_2).
```

takes two `synset_IDs`, whose synsets are similar in meaning. The second and the fourth arguments are word numbers, which usually specify one word in a synset. Nevertheless, the relation holds for any verb in the addressed synset. Therefore, the arguments `w_num_1` and `w_num_2` are instantiated to `0`.

Basically, the arguments two and four have no other use than to indicate that `vgp` is a lexical, not a semantical relation.

For example,

```
vgp(200072911,0,200437549,0).
```

Consulting **wn_s.pl** and **wn_g.pl**, returns that the synset `200072911` containing the words *to grow*, *to develop*, *to produce*, *to get*, *to acquire*, meaning *come to have or undergo a change of physical features and attributes*. The synset `200437549` contains the verbs *to develop* and *to evolve* with a similar meaning: *acquire or build up traits or characteristics*.

Because the relation is reflexive, there is an entry with switched `synset_ID` for every clause.

2.2.9 wn_at.pl

The file **wn_at.pl** describes the attribute relation between noun and adjective synsets. An attribute is a noun that describes a characteristic of an entity. Each attribute has values, which are described by adjectives. For example, the noun *size* is an attribute with the values *little*, *small*, *big* and *large*. This relation between nouns and adjectives are defined by the `at` operator

```
at(synset_ID_1,synset_ID_2).
```

So the entries

```
at(104322959,301148236).
at(104322959,301149766).
```

determine that the synset `104322959`, containing only the noun *weight* is in an attribute relation to the adjectives of the synset `301148236`, containing the word *heavy*, and of the synset `301149766`, containing the word *light*.

The attribute relation between nouns and adjectives is semantical.

2.2.10 wn_ant.pl

`Ant` is an abbreviation for antonymous words. The file `wn_ant.pl` stores all relations between words that are antonyms. Two words are antonyms if they have opposite meanings. For example, the two words *natural object* and *artefact* are antonyms, which tells us their meaning is opposite. Being an antonym of a word, does not mean that you can always say, the antonym *is not* the word. For example *artefact*, being an antonym of a *natural object*, does not necessarily mean *an artefact is not a natural object*.

The four place predicate, storing all this information, has the following structure:

```
ant(synset_ID_1,w_num_1,synset_ID_2,w_num_2).
ant(synset_ID_2,w_num_2,synset_ID_1,w_num_1).
```

The arguments `synset_ID_1` and `w_num_1` refer to one word in the corpus, `synset_ID_2` and `w_num_2` to its antonym.

Note, that the relation antonymy works in both directions, that is being an antonym of a word makes the word itself an antonym of the antonym. Therefore every `ant/4` is followed by an `ant/4`, expressing the reverse relation. Antonymy is a lexical relation, as it relates two words, not two synsets.

2.2.11 wn_sa.pl

The file **wn_sa.pl** relates two words, one giving additional information about the other one. This lexical relation only connects verbs or adjectives. The general structure of the `sa` operator takes four arguments, specifying two words.

```
sa(synset_ID_2,w_num_2,synset_ID_1,w_num_1).
```

First, let us have a look at `sa/4` for verb relations. The first two arguments specify a verb, the third and fourth argument define its phrasal verbs that are similar in meaning. Recall, a phrasal verb is a verb plus a preposition that can be different in meaning to the original verb. For example, *to give up* would be a phrasal verb, meaning *to stop doing something*, which is different to the meaning of the verb *to give*.

```
sa(200001742,1,200003768,3).
sa(200001742,1,200004389,3).
```

The synset `200001742` and word number `1` specify the word *to breathe*. The two words described by the third and fourth argument of the two clauses are its phrasal verbs *to breathe out* and *to breathe in*, which have a similar meaning to *to breathe*. The relation does not return all phrasal verbs belonging to a verb, only the ones that do not change their meaning.

For adjectives, the word, specified by the third and fourth argument, describes the word linked to the first two arguments. If the relation is true for one adjective of a synset, then it holds for any adjective of the synset. Thus, `w_num` is equal to `0` in every predicate entry. As an example, consider the word *abstract*, being the only word of the synset `300012315`. The predicates

```
sa(300012315,0,302319830,0).
sa(300012315,0,301928363,0).
```

reveal, that the words *intangible* and *impalpable* of the synset 302319830 and the word *nonrepresentational* of the synset 301928363 describe the word *abstract*.

2.2.12 wn_ppl.pl

The ppl operator arranges the participle relation between verbs and adjectives. In English, there is the present and the past participle, both describing either a tense form of a verb, or an adjective derived from a verb. As WordNet does not store any information on different tenses, the ppl operator describes the verb-adjective participle relation. The present participle adds *-ing* to the stem of the verb, e.g. the verb *to walk* becomes the adjective *walking*, like in the phrase *a walking person*. The past participle adds *-ed* to the stem of a word to form an adjective, e.g. *the stressed person* where the adjective is derived from the verb *to stress*. The predicate takes four arguments to specify two words.

```
ppl(synset_ID_1,w_num_1,synset_ID_2,w_num_).
```

Synset_ID_1 and w_num_1 determine the adjective related to the verb, recorded by synset_ID_2 and w_num_2. The relation is lexical.

2.2.13 wn_per.pl

The extension per stands for pertain and describes a lexical relation where a word pertains to another word. The predicates per/4 defines two words by their synset ID and word number.

```
per(synset_ID_1,w_num_1,synset_ID_2,w_num_2).
```

The first word, located by the first two arguments, can either be an adjective or an adverb.

In case of an adjective, the second word must be a noun or another adjective. Then, the original adjective pertains to the noun or the other adjective. For example, the first two arguments could refer to the adjective *weekly*, which pertains to the noun *week*, defined by the third and fourth argument. Or, the first two arguments could refer to the adjective *transatlantic*, which pertains to the adjective *Atlantic*, defined by the third and fourth argument.

If the first two arguments describe an adverb, then the assigned second word is the adjective from which the adverb is derived. E.g. the adverb *essentially* is derived from the adjective *essential*.

2.2.14 wn_fr.pl

The last Prolog database file **wn_fr.pl** is intended to offer a generic sentence frame for one or all verbs in a synset. The predicate structure is as follows:

```
fr(synset_ID,f_num,w_num).
```

The first argument is the `synset_ID` of the concerned verb(s). The third argument specifies the word by a word number or is equal to `0` if the relation holds for every word in the synset. The second argument specifies, which sentence frame should be assigned to which verb. Unfortunately, the Prolog database does not offer any information on the sentence frames. In that sense, the predicate is useless. You would need to consult the WordNet documentation (Fellbaum et al. 2003d) to translate which `f_num` belongs to which sentence.

In the software packet belonging to this paper, I included a file called **wn_sen.pl**, which should be added to the original WordNet files. Here, an additional predicate called `sen`, short for sentence, is listed.

```
sen(f_num,string_1,string2).
```

`Sen` takes `f_num` as the first argument and assigns two strings. `String_1` should be the part of the sentence in front of the looked up verb, `string_2` the part after the verb.

For example, the verb *to prefer* is the only word in the synset `201433968`. Consulting **fr_wn.pl** we get the predicate

```
fr(201433968,15,0).
```

Therefore, we know all verbs in the affected synset, in this case only one, are assigned to the sentence frame with `f_num 15`. Now, we can consult the new added predicates and find

```
sen(15,‘Somebody’,‘s something to somebody’).
```

Accordingly, we find out that the verb *to prefer* is used in the general context *Somebody prefers something to somebody.*

To experiment with `fr/3` and `sen/3`, try the following program. Be sure, that you have consulted the files **wn_s.pl**, **wn_fr.pl** and **wn_sen.pl**.

```
% sentence_frame(+Verb)
% take an atom as argument, which should be a verb, and print out a sentence
% on the screen that shows the context in which the verb usually occurs

sentence_frame(Verb) :-
                              s(Num,W_Num,Verb,v,_,_),
                              fr(Num,F_Num,W_Num),
                              sen(F_Num,String_1,String_2),
                              write(String_1),
                              write(Verb),
                              write(String_2), nl.

sentence_frame(Verb) :-
                              s(Num,_,Verb,v,_,_),
                              fr(Num,F_Num,0),
                              sen(F_Num,String_1,String_2),
                              write(String_1),
                              write(Verb),
                              write(String_2), nl.
```

The predicate `sentence_frame/1` takes a verb and finds its `synset_ID`. Then, it consults the predicate `fr/3`, which either succeeds, if the third argument is equal to the `w_num`, or fails and backtracks. In the later case, the `fr/3` succeeds with `0` as the third argument, meaning the relation holds for all the verbs in the synset. In the end, consult `sen/3` and output the matched sentence.

Examples:

```
?- sentence_frame(‘prefer’).
Somebody prefers something.

?- sentence_frame(‘walk’).
Somebody walks.

?- sentence_frame(‘give’).
Somebody gives somebody something.
```

3 Working with WordNet

The following section discusses some basic issues that arise while working with WordNet and offers some handy predicates to work efficiently. For further documentation of the mentioned predicates, please consult the Prolog files of the predicates.

3.1 Indexing and WordNet

The Prolog database files of WordNet consist of 484381 lines of code. Therefore, speeding up the processing of these files is one of the main goals when working with WordNet.

One approach to reduce the running time for looking up a word is indexing. Indexing decreases the number of clauses that are tried before matching one. The simplest version of indexing, used in every Prolog, is to go directly to the right predicate of the asked arity. SWI-Prolog also indexes on the first argument by default (Wielemaker 2003:81).

To look up a word in WordNet we need to consult the predicate

```
s(+Synset_ID,+W_Num,+Word,+SS_Type,+Sense_Number,+Tag_Count).
```

After finding out which `synset_ID` is assigned to the word, one can look up the meaning by consulting

```
g(+Synset_ID,+Gloss).
```

The given word is stored in the third argument of the `s` predicate, which means the default indexing does not affect the unification.

There are two possible attempts to speed up the running time by indexing. The built-in predicate

```
index(+Predicate).
```

specifies on which arguments of a predicate indexing should be performed. The argument `Predicate` gets instantiated to `1` if indexing were to be executed on the argument, otherwise to `0`. Keep in mind that at most four arguments and only the first 32 arguments can be indexed (Wielemaker 2003:82). In this case the predicate

```
index(s(0,0,1,0,0,0)).
```

would optimize the running time of looking up a word. The advantage of this technique is that we can declare indexing on more than one argument, e.g. on the `Synset_ID` and the `Word`. The disadvantage is that we would have to declare the predicate `index/1` in every file we produce, and the database becomes less portable.

The second attempt to speed up the time by indexing, is to change the argument order of **wn_s.pl**. The predicate `improve_file/0` takes the word from the `s` predicate and puts it as the first argument. The other arguments remain in their order. The newly created file is called `wn_s_new.pl`. Now the default indexing of Prolog speeds up the process. Timing the simple predicate of looking up a word shows that the indexing speeds up the process nine times.

3.2 Converting WordNet Files

Working with the WordNet files with respect to Natural Language Processing, there are several generalisations that simplify the interaction with other tools. Especially in developing ProNTo (Prolog Natural Language Tools), we agreed on the following parameters to improve interaction. First, we only want lower case letters to appear in the clauses. Second, instead of using underscores to divide words, we represent the words in lists. Furthermore, we use open lists, in the aspect that one wants to work not only with one word at a time, but with sentences or texts that can be easily turned into lists.

The predicate `convert_file/0` takes care of converting the `wn_s.pl` file accordingly. It is an extended version of the predicate `improve_file/0`. All advantages of using indexing remain the same. The following clause is an example of the new structure:

```
s([human, action|_G4016],100022113,2,n,1,1).
```

3.3 A subset of WordNet

For testing purposes, it is handy to use only a subset of WordNet, which is smaller and contains only the most common words of the English language. Here, I want to introduce the predicate

```
subset_wn_s(+Number).
```

that takes the **wn_s.pl** file as input and creates a new file **wn_s_subset.pl**, containing only the most common words, based on the last argument of the `s/6` predicate (see section *Documentation of WordNet from a Prolog programmer's point of view*). The argument `Number` indicates how many words should be in the subset. This should speed up experimenting with WordNet enormously.

If you decide to work with a subset on a project, it might be interesting to not only shorten the main `wn_s.pl` file, but also all other WordNet files. The predicate

```
subset_wordnet(+Number).
```

calls `subset_wn_s/1`, and then converts all other WordNet files accordingly. The new files are named **wn_*operator*_subset.pl**.

These predicates offer the possibility of knowing the advantages associated with working on subsets of WordNet. Apart from that, they offer an outline on how to program the automatic converting of the files. An extension for the future could be to create different kinds of subsets of WordNet, e.g. a subset containing only nouns, only verbs or only adjectives.

3.4 Useful predicates for working with WordNet

To work with WordNet, we need predicates that interface with the Prolog databases. All predicates listed here, including the ones of earlier sections, are described in detail in the Prolog files. To use a predicate, you just need to consult the file in which the predicate is defined, all other needed files will be consulted by `ensure_loaded`.

First, it needs to be clarified which input should to be accepted by the predicates. As we converted our files to use open lists, it is obvious that open lists are one possible input. To improve the convenience for the user, atoms should be accepted, too. For example, if you want to input the word *dog* you could either type `dog` or `[dog|X]`. There is one exception: as we decided earlier, that underscores are not customisable with other Natural Language Processing Tools, you have to use the open list format if you want to give in two words, e.g. `[physical,thing|X]`.

All predicates are designed to be used with the converted WordNet files, not the originals. Working with subsets or the whole database is optional.

One of the most obvious tasks in WordNet is the looking up of words. The predicate

```
lookup(+Word).
```

looks up a word in **wn_s.pl** and prints out its syntactic category and definition. The extended predicate

```
lookup(+Word,-SynsetList).
```

looks up a word and returns a list of all synsets in that the word was found. This can be useful if you want to get synonyms for all the different meanings of a word. The two predicates are defined in the file **lookup.pl**.

In the first part of this paper, I documented the WordNet files and described how the words are grouped in synsets. The predicate

```
find_synset(+Synset_ID,-WordList).
```

returns a list of all words that are in the synset of the given `Synset_ID`. The predicate is defined in the file **find_synset.pl**.

Also, all semantical and lexical relations have been explained in detail. The following predicates experiment with these relations.

`find_hyp_chains(+Word,+Cat).`

finds all hypernym chains of a word and lists them on the screen. For example,

`[[organism|_G529], [being|_G520]]`

is a hypernym chain for the word dog, so is

`[[animal|_G616], [animate, being|_G607], [beast|_G595], [brute|_G586], ... .`

Bear in mind, that only verbs and nouns can be hypernyms. As a word can only have hypernyms of the same category as itself, the user has to specify the category in the second argument. For example, if you are looking for hypernyms of the noun *dog*, you might find the word *mammal*. You certainly do not want to find the word *to move*, which is a hypernym of the verb *to dog*.

`find_hyp(+Word,+Cat,-HypList).`

finds all hypernyms of a Word and returns them in a list.

The two predicates

`find_ent_chains(+Word), find_ent(+Word,-EntList).`

work the same way as the according hypernym predicates, only that we do not need to specify the category, as an entailment can only be a verb.

The two hypernym predicates and the two entailment predicates are defined in the file **hyp_ent.pl**.

As the structure of the predicates are so similar, the predicates `find_hyp_chains/2` and `find_ent_chains/1` call the predicate `find_chain/3`, which is a generalized procedure, with the fourth argument handing over whether an entailment or a hypernym relation is requested. Likewise `find_hyp/3` and `find_ent/1` calls `find/4`, a generalized procedure, handing over which relation is requested in the third argument.

The predicate

`find_sim_meanings(+Word).`

takes an adjective, finds all its similar meanings and prints it on the screen. The similar predicate

`find_sim(+Word,-SimList).`

returns a list of words with similar meanings of the input word. The predicates are similar to the hypernym and entailment predicates. But, there is one basic difference: Handling similar meanings, it does not seem reasonable to follow chains, like we did for hypernyms and entailments. Therefore, the algorithm just looks for all similar meanings of the given word with the build-in predicate `find_all`. It does not use recursion to find a chain of similar meanings.

As discussed in the earlier sections **wn_mm.pl**, **wn_ms.pl** and **wn_mp.pl** describe several relations between meronyms and holonyms. The following predicates interface these relationships. This relation only holds for nouns. The predicate

```
member_of(+Word,-GroupList).
```

takes a word as input and finds the list of groups that the word is a member of. For example,

```
?- member_of(faculty,X).
X  = [school|_G413]
```

As a result we get that a *faculty* is a member of a *school*.

```
has_member(+Word,-MemberList).
```

finds the list of members that are in the group. For example,

```
?-has_member(faculty,X).
X = [professor|_G407]
```

As a result we get a *professor* is a member of a *faculty*.

Similarly, the predicates

```
substance_of(+Word,-List).
has_substance(+Word,-SubstanceList).
```

find the list of words that a given word is a substance of, or the list of substances that are in one word.

Examples:

```
?- substance_of(water,X).
X = [tear|_G407]

? has_substance(water,X).
X = [h2o|_G407]
```

Water is a substance of a *tear*, *H20* is a substance found in *water*.

The two predicates

```
part_of(+Word,-WholeList), has_part(+Word,-PartList).
```

return the list of things that one word is a part of, or the list of things that are part of one word.

Examples:

```
?- part_of(leg,X).
X = [table|_G407]

? has_part(leg,X).
X = [knee|_G407]
```

The word *leg* is a part of a *table* and a *knee* is a part of a *leg*.

The input can be an atom or an open list, the output is always an open list. The predicates are defined and documented in the file **meronym_holonym.pl**.

The three predicates `member_of/2`, `substance_of/2` and `part_of/2` call the predicate `all_one/4` with the third argument instantiated to the corresponding filename extensions `mm`, `ms` and `mp`. Then the procedure is generalized. First, we form the predicate we are looking for by using the univ operator, then we call the build-in predicate `find_all/3` to get all possible answers. `Make_list/2` then finds all the words belonging to the synset IDs and stores them in a list. There are two alternatives defined, one handling atoms, and one handling open lists. The predicate `all_two/4` is similar, but instead of matching the first argument and returning the second, it is looking for the first one by unifying the second one. By that procedure it handles the cases for the predicates `has_member/2`, `has_substance/2` and `has_part/2`.

Last, I want to describe the predicate

```
cause(+Verb,-CauseList).
```

Given a verb, it creates a list of verbs that are causes of the given verb. For example, the verb *to leak* is a cause of the verbs *to break*, *to get out*, *to get around*. The predicate is defined in the file **cause.pl** and consists of only one line of code. It works exactly the same way as the predicate `member_of/2`. It calls the predicate `all_one/4` with the third argument instantiated to `cs`, the wn extension of the corresponding database file. Now, we realize why the predicate `all_one/4` needs the fourth argument, indicating the category. In the meronym-holonym relation this seemed unreasonable, as we were only handling nouns. But as we reuse the predicate for the cs relation, we need to introduce another argument, so that it can handle verbs, too.

3.5 How to handle common words that are not in WordNet

This section is about words that can not be found in WordNet. This becomes a non-trivial question when faced with the fact that WordNet Files act as the database for Natural Language Processing Tools.

The predicate

```
lookup_text(+FileName).
```

tries looking up all the words in a text, finds out, which words are not in WordNet and prints them on the screen. By running the program, you might find that the words *this*, *which*, *whether*, *of*, *from*, *the*, *my*, *is* are not in the WordNet database.

In what way to handle these words, depends on the problem you are approaching. Extending the knowledge base is one.

`Lookup_text/1` would be more useful if the words would not be printed on the screen, but returned in a list. The predicate

```
lookup_text(+FileName,-List).
```

executes this.

3.6 Interfacing with the ProNTo Morphological Analyser

The ProNTo toolkit are independent Natural Language Processing Tools, which can be used alone or together. If used together, the WordNet database would have to interface with the morphological analyser. Designed by Jason Schlachter, the ProNTo morphological analyser has several outputs that need to be checked in WordNet whether they are words or not.

The morphological analyser is able to analyze only one word at a time. It outputs a list containing a single morphological analyzes. By backtracking you can get all other possibilities. The output could then be

```
[[walk,-ed]].
```

Later, the morphological analyzer has been extended to output not only one analyzes, but a list containing all possibilities of how to split up one word, e.g.

```
[[walked],[walke,-d],[walk,-ed]].
```

Now, to check whether one of the morphological interpretations is a word or not, we call the predicate

```
morph_atoms_lookup(+Morph).
```

which takes one of the just described outputs and succeeds if the word, or one of the words, can be found in WordNet and fails if none can be found.

But the morphological analyzer also interprets longer phrases or sentences. E.g. the output for a sentence could be

```
[[[he]],[[walked],[walke,-ed],[walk,-ed]],[[slowly],[slow,-ly]]].
```

This case is handled by the predicate

```
morph_bag_lookup(+Morph).
```

which takes the described input and succeeds if at least one word of each list, containing different morphological analyses of a word, is found in WordNet and fails if not.

These predicates might be useful for someone who wants to interface our tools. But it might not be enough that the predicates are succeeding or failing, some output containing information would be useful. Therefore I designed two more predicates

```
morph_atoms_lookup(Morph,Result).
morph_bag_lookup(Morph,Result).
```

which are taking the same input, but instead of failing or succeeding, they return a list to the second argument. This list contains a list for each input word. Either, if the word can not be found, it returns `[is,not,a,word]`, or if the word can be found, it returns the list `[word,synset_ID,w_num,category]`.

Examples: [2]

```
?- morph_atoms_lookup([[talk,-ed]],R).
R = [[talk, 100677091, 1, n]] ;
R = [[talk, 105953501, 1, n]]

 ?- morph_bag_lookup([[[he]],[[talked],[talke,-d],[talk,-ed]]],R).
R = [[he, 105716399, 1, n], [talk, 100677091, 1, n]] ;
R = [[he, 105716399, 1, n], [talk, 105953501, 1, n]]
```

For further reference on how the predicates work, please consult the file **morph_lookup.pl**.

4 Conclusion

This paper was written to give a good documentation of the Prolog WordNet Files. The knowlege presented in this paper should simplify the work with the Prolog WordNet database. Of course, the list of interface predicates can be extended, according to the needs, felt by the user.

[2] In this case the pronoun *he* is found in WordNet as there is an entry for *he* of the category noun. But not all pronouns have entries like this. For example the word *she* can not be found in WordNet.

References

Beckwith, Richard; Fellbaum, Christiane; Gross, Derek; Miller, George A.; and Miller, Katherine (1993). *Five papers on WordNet.* Technical report. Princeton University. Computer Science Laboratory.

Fellbaum, Christiane (1999). *WordNet: an electronic lexical database.* Cambridge, Mass: MIT Press.

Fellbaum, Christiane; Langone, Helen; Miller, George A.; Tengi, Randee; Wakefield, Pamela; and Wolff, Susanne (2003a). *Format of Prolog-loadable WordNet files - prologdb(5WN).* WordNet 1.7.1 Reference Manual: Section 5 File Formats. Princeton University. http://www.cogsci.princeton.edu/~wn/man1.7.1/prologdb.5WN.html.

Fellbaum, Christiane; Langone, Helen; Miller, George A.; Tengi, Randee; Wakefield, Pamela; and Wolff, Susanne (2003b). *Format of sense index file - senseidx(5WN).* WordNet 1.7.1 Reference Manual: Section 5 File Formats. Princeton University. http://www.cogsci.princeton.edu/~wn/man1.7.1/senseidx.5WN.html.

Fellbaum, Christiane; Langone, Helen; Miller, George A.; Tengi, Randee; Wakefield, Pamela; and Wolff, Susanne (2003c). *Format of WordNet database files - wndb(5WN).* WordNet 1.7.1 Reference Manual: Section 5 File Formats. Princeton University. http://www.cogsci.princeton.edu/~wn/man1.7.1/wndb.5WN.html.

Fellbaum, Christiane; Langone, Helen; Miller, George A.; Tengi, Randee; Wakefield, Pamela; and Wolff, Susanne (2003d). *Format of lexicographer files - wninput(5WN).* WordNet 1.7.1 Reference Manual: Section 5 File Formats. Princeton University. http://www.cogsci.princeton.edu/~wn/man1.7.1/wninput.5WN.html.

Fellbaum, Christiane; Langone, Helen; Miller, George A.; Tengi, Randee; Wakefield, Pamela; and Wolff, Susanne (2003e). *WordNet 1.7.1. - a lexical database for the English language.* http://www.cogsci.princeton.edu/~wn.

Wielemaker, Jan (2003). *SWI 5.1 Reference Manual.* University of Amsterdam. Dept. of Social Science Informatics (SWI).