COMP24412, Lab 4

We now have an inference engine that can operate directly over dependency trees, and machinery for obtaining dependency trees for fairly simple natural language sentences. The goal now is to stick these two things together.

The first part of the current lab is to write programs to do things with WordNet. The obvious use for WordNet is for deciding for a given pair of words $W_1$ and $W_2$ whether there is a synset of $W_1$ which is a subset of $W_2$. The course repository contains two files, `wn_s.pl` and `wn_hyp.pl`, which link words to synsets and provide subset/superset relations between synsets. `wn_s.pl` and `wn_hyp.pl` basically are WordNet – there are some other files that attempt to specify other relations, but they are sparser and even more error prone than these two, which are the things that get used most widely.

```
...
s(102472293,2,'man',n,4,29).
s(102472293,3,'human being',n,1,21).
s(102472987,2,'human race',n,1,0).
s(102472987,3,'humanity',n,3,0).
s(102472987,4,'humankind',n,1,0).
s(102472987,5,'human beings',n,1,0).
s(102473307,1,'Homo erectus',n,1,0).
s(102473554,1,'Pithecanthropus',n,1,0).
s(103716327,1,'man',n,10,0).
...
```

Figure 1: Extract from `wn_s`

```
...
hyp(100002137,100001740).
hyp(100002452,100001930).
hyp(100002684,100001930).
hyp(100003553,100002684).
hyp(100003993,100003553).
...
```

Figure 2: Extract from `wn_hyp`
The entries in wn_s link root forms to synset IDs, e.g. one of entries for the word ‘man’ is linked to a synset whose internal ID is 102472293 and another is linked to a synset whose ID is 103716327; and ‘human race’, ‘humanity’, ‘humankind’ and ‘human beings’ are all linked to the same synset, whose ID is 102472987.

The first entry in wn_hyp says that the synset whose ID is 100002137 denotes a subset of the one whose ID is 100001740, and likewise for the others.

**Task (i)** Write a Prolog predicate `synonyms` which returns all the words that have some synset in common with the target word, e.g.

```prolog
| ?- synonyms(man, P).

P = [homo,'human being',human,world,'human race',humanity,humankind,'human beings',
    humans,mankind,piece,'adult male',serviceman,'military man',
    'military personnel',valet,'valet de chambre',gentleman,'gentleman’s gentleman'] ? yes
```

Hint: use `findall` (from the SATCHMO lab). Note the `findall` makes a list of things that look like its first argument and that satisfy its second. Consider:

```prolog
| ?- L = [1,2,3,4], findall([X, Y], (member(X, L), member(Y, L), X < Y), Z).

L = [1,2,3,4],
Z = [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]] ? yes
```

This collects all pairs [X, Y] where X and Y are members of L and X is less than Y.

For task (i) you just need to remember that the second argument can be a conjunction of goals – that you’ve got your target word and you want to find a synset of that word and then another word with the same synset. The solution to task (i) is short. If you find yourelf writing something that takes more than 10 lines of Prolog you’re thinking on the wrong lines. **3 marks**

**Task (ii)** Write a Prolog predicate `:=` which succeeds if its first argument is a subset of its second:

```prolog
| ?- man := animal.

yes
| ?- boy := animal.

no
```

To define `:=` as an infix operator, include

```prolog
:- op(700, xfx, :=).
```
in your program.

Hint: you probably want to define two predicates, \( \equiv \) itself which links words to synsets and synsets back to words, and \texttt{ancestor}, which looks at links between synsets. You’re going to want to use \texttt{ancestor} for task (iii), at which point you’re going to want to know how far it is from the seed to the ancestor, so it would be a good idea to define it here so that it keeps a counter of the number of steps it has taken to get from the seed to the ancestor, and to ignore that part of the result for now. 

You can also use WordNet as the basis for seeing whether two words are similar. There are various ways of doing this: one of the most widely used involves seeing how far up the hierarchy of synsets you have to go to get a common ancestor of a pair of words – the lowest common ancestor of ‘man’ and ‘woman’, for instance, is ‘adult/grown-up’. If two words have a common ancestor which is quite near to them they are probably quite similar – ‘man’ and ‘woman’ are both 1 level below ‘adult’ whereas the lowest common ancestor of ‘man’ and ‘turtle’ is ‘vertebrate’, which is 5 levels above ‘man’ and 4 above ‘turtle’.

But the distance to the common ancestor isn’t all you need: ‘tortoise’ and ‘terrapin’ have a common ancestor one level up, ‘mammal’ and ‘aquatic vertebrate’ also have a common ancestor one level up, but tortoises and terrapins are obviously more similar than mammals and aquatic vertebrates (which includes ‘jawless fish’ and ‘placoderm’, whatever they might be). How can we deal with this? The higher up in the overall tree you are, the more things that you cover, so the less similar the groups of things below you are. We can sort of calculate this by taking into account the distance to the top of the tree. Wu-Palmer similarity (Wu and Palmer 1994) is given by

\[
\frac{\text{dist}(W, LCA) + \text{dist}(W', LCA)}{\text{dist}(W, \text{top}) + \text{dist}(W', \text{top})}
\]

There is a slight complication here. Wu-Palmer similarity is defined for synsets, we want similarity for words. Following our general strategy, we want to take the similarity for a pair of words to be the lowest similarity for any pair of synsets of the two words (assuming that the more similar the two words are the lower the score).

| ?- wup(man, woman, P). |
| P = 0.15384615384615385 |
| yes |

| ?- wup(turtle, terrapin, P). |
| P = 0.07142857142857142 |
| yes |
| ?- wup(man, terrapin, P).
P = 0.5263157894736842 ?
yes

| ?- wup(mammal, 'aquatic vertebrate', P).
P = 0.18181818181818182 ?
yes

| ?- wup(lorry, turtle, P).
P = 0.6666666666666666 ?
yes

**Task (iii)** Write a Prolog predicate `allAncestors` that finds all the ancestors of all the synsets of a given word.

Hint: you’ve already got a predicate that will find you ancestors of a synset and tell you how far it is from the seed synset to the ancestor. So all (!) you need is to use `findall` with this and collect the pairs of ancestors and distances. Again, the solution to this one is short. If you find yourself writing 10 or more lines of code then you’re going in the wrong direction. **4 marks**

**Task (iv)** Write a Prolog predicate `lca` that finds the lowest common ancestor of a pair of words.

Use `allAncestors` on each of the words, then find something which is a common ancestor of each for which the sum of the distances from each word to this ancestor is lower than for any other ancestor of these two words. **3 marks**

**Task (v)** Use `lca` to write a predicate `wup` to calculate the Wu-Palmer distance between two words. `lca` will do most of it, but you also need to find the most distant ancestor of each word (find all the ancestors: given that WordNet is supposed to be a tree, so each synset ought to have at most one parent, the length of this list should be the distance from the starting point to the top). **3 marks**

But seeing whether isolated words entail one another or are similar to one another isn’t much use by itself. We generally want to use this as part of some overall task such as seeing whether one sentence entails another or is similar to it. A simple version of this is to say that one sentence is similar to another if they contain similar words.

**Task (vi)** Write a Prolog predicate `bow` that calculates the cheapest order-preserving match between two sentences where the first sentence may be longer than the second (use your definition of `wup` for comparing individual words):

---

1The assumption that no synset has more than one parent isn’t strictly true, but it works fine in most cases.
This one is tricky. **2 marks** if your version of `bow` works when the two sentences are the same length, and an extra **2 marks** if you can get it to do anything remotely sensible in cases where the first sentence is longer than the second.

This is the ‘weighted string-edit distance’ solution to the problem of textual entailment (Dagan et al. 2005). It’s not the state of the art, but it’s fairly respectable and reasonably fast – $O(len(S1) \times \text{len}(S2))$ (the obvious way to do it in Prolog is not as efficient as that: to do it in $O(len(S1) \times \text{len}(S2))$ you need to implement ‘dynamic time warping’ (Sakoe and Chiba 1978), and while I am a big fan of Prolog for things that Prolog is good for, I wouldn’t use it for implementing DTW).

**References**

