Imperial College London
Department of Computing

MSc Prolog Programming – Assessed Exercise No. 2

Issued: Tuesday 26 November 2013
Due: **Tuesday 10 December 2013**

Lab Sessions:	Tuesday 26 November (pm)	Thursday 5 December (am)
	Thursday 28 November (am)	Monday 9 December (am)
	Tuesday 3 December (pm)	Tuesday 10 December (am)

The Exercise

This exercise asks you to write Prolog predicates that can help in the decoding of (very simple!) secret messages. You have been supplied with an empty Prolog file called `spies.pl` in a gitlab respository named `mcslab_5_<login>`, where `<login>` is your DoC login. You are required to fill it with definitions for the predicates described below. Feel free to also define any auxiliary predicates which you feel are necessary.

1. Define a predicate *decode(Message, Decoded_Message)* which should produce *Decoded_Message* by decoding *Message* using the following code breaking key:

   - Replace the word 'bear' by the word 'double'.
   - Replace the word 'cub' by the word 'agent'.

   For example, if *Message* is `[bill, is, a, bear, cub, and, bill, went, shopping, with, jim]` then *Decoded_Message* should be `[bill, is, a, double, agent, and, bill, went, shopping, with, jim]`. Note that in this exercise words are constants. The sentences *Message* and *Decoded_Message* are therefore lists of constants.

2. Using the *decode/2* predicate or otherwise, define the predicate *agents(Message, Decoded_Message, ListofAgents)*, which should produce not only the decoded version of *Message* but also the list of names the *Message* accuses of being double agents. The list should have no duplicates and should be sorted. Someone is accused of being a double agent if they are said to be a bear cub. The name of an accused individual is always one word and never clashes with the code-words (i.e. a name cannot be 'bear', 'cub', 'double' or 'agent'). Your program should also work if *Message* contains no accusations, in which case *ListofAgents* should be empty.

3. Define a predicate *count_word(W,L,C)* such that *C* is the number of times the word *W* occurs in list *L*. You can assume that in any call to *count_word/3* the first two arguments are fully instantiated.

4. Using your predicates from questions 2 and 3, or otherwise, define the predicate *count_ag_names(Message, Ag_name_counts)* such that *Message* is a coded message and *Ag_name_counts* is a list of elements of the form *(Name, Count)* where *Name* is a name accused of being a double agent in *Message*, and *Count* is the number of times that name occurs in *Message*. For example if *Message* is `[bill, is, a, bear, cub, and, bill, went, shopping, with, jim, and, mary, is, a, bear, cub]` then *Ag_name_counts* is `[(bill, 2),(mary,1)]`. The list *Ag_name_counts* should be ordered on Name. Again, your program should return an empty list if *Message* contains no accusations.

5. Using any of the above predicates or otherwise, define a predicate *accusation_counts(M, AC)* such that given a coded message *M*, *AC* is a list of elements of the form *(Name, Count)* where *Name* is a name accused of being a double agent, and *Count* is the number of times that name is accused of being a double agent in *M*. For example if *M* is `[jack, is, a, bear, cub, and, bill, is, a, bear, cub, and, bill, went, shopping, with, jim, and, mary, is, a, bear, cub, and, jack, is, a, bear, cub]` then *AC* will be the list `[(jack, 2), (bill, 1), (mary,1)]`. The list *AC* should be in descending order of `Count`, and your program should return an empty list if *M* contains no accusations.

Submission

Commit your solution to your locally cloned git repository, and then push the respository back to the gitlab server. Submit the commit token for the version you intend to be tested in a text file named `cate_token.txt` to CATE.

Your submission should be self-contained: it should be possible to consult your file successfully using the lab-installed version of Sicstus Prolog and to test your programs without needing to consult additional Sicstus Prolog libraries. See below for how to include directives in your program for loading libraries.

Assessment

Questions 1-5 have maximum marks of 20%, 15%, 15%, 20%, and 30%, respectively. In each case 85% of the marks are allocated for program correctness, mostly checked through auto-testing. Thus, it is very important that you use predicates exactly as specified (spelling and number of arguments). Of course, you can/should define your own auxiliary predicates in order to reduce the size of predicate definitions. 15% of the marks are allocated to programming style, including for example:

- Clear and helpful commenting.
- No long predicate definitions. In case a clause gets too long, try to split it into auxiliary predicates.
- Readable code: indentation, sensible naming etc.

Useful Prolog Directives

If your program requires the lists library add the following entry (called a Prolog directive) at the top of your file:

```
:- use_module( library( lists ) ).
```

This causes Sicstus to automatically load the lists library upon consulting your file.

Another directive you may find useful in the above exercises is

```
:- set_prolog_flag( toplevel_print_options, [max_depth(100)] ).
```

If lists are of a certain length then Sicstus, by default, abbreviates them uses ellipses rather than displaying the entire list. The above is a Sicstus meta-directive for overriding this: it increases the depth (complexity) of terms after which Sicstus will use ellipses when displaying terms on the screen (i.e. this directive will display more detailed/lengthy terms without abbreviation).

You should not require any other libraries/directives.