Homework #3

Deadline: Friday, 19 September 2014, 18:00

Note: For this homework, and in general, make use of cuts (`!`) only when really needed.

Question 1 (2 points)

Solve Exercise 5.4 from the Lecture Notes (counting occurrences of an object in a list).

Question 2 (3 points)

A formula of propositional logic (involving only negation, conjunction, and disjunction, but not, e.g., implication) is said to be in *negation normal form* (NNF) if it is the case that every subformula that is negated is a negative literal (i.e., the negation of an atomic proposition). That is, for example, $(p \vee \neg q) \wedge \neg r$ is in NNF, while $p \wedge \neg(q \vee r)$ and $\neg\neg p$ are not. Define appropriate Prolog operators for negation, conjunction, and disjunction (e.g., as discussed in class). Then write a predicate `nnf/1` that takes a formula of propositional logic (involving only these three operators) as an argument and that succeeds if and only if the formula provided is in NNF. Examples:

```
?- nnf((p or neg q) and neg r).
Yes

?- nnf(p and neg (q or r)).
No

?- nnf(neg neg p).
No
```

Note: If you are looking for a challenge, you could also try to implement a predicate for translating a given formula into an equivalent formula in NNF (using de Morgan's laws).

Question 3 (3 points)

In the Treaty of Rome (1957) the six founding countries of the European Union specified the voting rule to decide on proposals in the Council of the European Commission. To pass, a proposal has to reach the threshold of 12 votes. The large countries (France, Germany, and Italy) each have 4 votes; the medium-sized countries (Belgium and the Netherlands) each have 2 votes; Luxembourg has 1 vote. Let us represent these facts in Prolog:

```
countries([belgium, france, germany, italy, luxembourg, netherlands]).
weight(france, 4).
weight(germany, 4).
weight(italy, 4).
weight(belgium, 2).
weight(netherlands, 2).
weight(luxembourg, 1).
threshold(12).
```

This may suggest that, say, Germany has twice as much voting power as the Netherlands, which in turn have twice as much power as Luxembourg. But, as we shall see, this would be a rather naïve interpretation of the rule.

A *coalition* of countries (a subset of the six countries) is called *winning*, if their sum of weights is at least equal to the threshold; otherwise it is called a *losing* coalition. Write a predicate `winning/1` that, when given a list of countries, succeeds if and only if that list constitutes a winning coalition. Examples:

```
?- winning([belgium, france, germany, netherlands]).
Yes

?- winning([belgium, netherlands, luxembourg]).
No
```

Let us say that a given country x is *critical* for a given coalition C if (i) C does not include x, (ii) C alone is not winning, but (iii) C together with x is winning. Implement a predicate `critical/2` to check whether a given country is critical for a given coalition. Examples:

```
?- critical(netherlands, [belgium, france, germany]).
Yes

?- critical(netherlands, [france, germany, italy]).
No
```

Next we want to find a way to generate *all* coalitions that are critical for a given country. To this end, implement a predicate `sublist/2` that succeeds when its first argument matches a sublist of the list given as the second argument. It should be possible to use it like this:

```
?- sublist(X, [a, b, c]).
X = [a, b, c] ;
X = [a, b] ;
X = [a, c] ;
X = [a] ;
X = [b, c] ;
X = [b] ;
X = [c] ;
X = [] ;
No
```

Now we can generate all critical coalitions for a given country through enforced backtracking:

```
?- countries(All), sublist(Coalition, All), critical(netherlands, Coalition).
All = [belgium, france, germany, italy, luxembourg, netherlands],
Coalition = [belgium, france, germany, luxembourg] ;
All = [belgium, france, germany, italy, luxembourg, netherlands],
Coalition = [belgium, france, germany] ;
All = [belgium, france, germany, italy, luxembourg, netherlands],
Coalition = [belgium, france, italy, luxembourg] ;
All = [belgium, france, germany, italy, luxembourg, netherlands],
```

```
Coalition = [belgium, france, italy] ;
All = [belgium, france, germany, italy, luxembourg, netherlands],
Coalition = [belgium, germany, italy, luxembourg] ;
All = [belgium, france, germany, italy, luxembourg, netherlands],
Coalition = [belgium, germany, italy] ;
No
```

That is, the 6 different lists bound to the variable `Coalition` above represent the 6 coalitions for which the Netherlands is critical. Let us define the *voting power* of a country as the number of coalitions for which it is critical (i.e, the voting power of the Netherlands is 6). Write a predicate `voting_power/2` to compute a given country's voting power (at this point you will need to make use of a built-in predicate introduced in the Friday lecture). Example:

```
?- voting_power(netherlands, Power).
Power = 6
Yes
```

What is the voting power of Germany? How about Luxembourg? The correct way of answering such a question is to write one sentence each providing the answer and to show both the query you used to obtain that answer and the response given by Prolog in support of your claim. Finally, for a really nice solution, if your answer is surprising or counterintuitive, you could comment on this matter in a further sentence, or you could provide a very short high-level explanation for *why* your answer is correct.

Note: The only place in your program referring to the specific countries or the specific threshold mentioned above should be the Prolog facts given at the start. That is, it should be possible to re-use your program for later incarnations of the European Union (with more countries, different weights, and a different threshold) by only changing those facts.

Question 4 (2 points)

For each of the following five Prolog operators, give a short and precise definition of its purpose and provide a meaningful example for a query involving the operator (together with the corresponding output produced by Prolog): `=/2`, `==/2`, `=:=/2`, `is/2`, and `=../2`. Note that `=../2` will be introduced in the lecture on Friday. We have not seen (and will not see) `==/2` in class; find out what it is used for on your own.