Homework #6

Deadline: Wednesday, 22 October 2014, 18:00

Question 1 (2 points)

Write a predicate `occurs/2` that will succeed if the variable given as the first argument occurs within the term given as the second argument. Examples:

```
?- occurs(X, f(Y,g(X))).      ?- occurs(X, f(Y,g(b))).
Yes                           No
```

You will probably need the built-in predicate `==/2`. You might also need `var/1`. Do not use any other built-in predicates directly related to the occurs-check or to sound unification.

Question 2 (2 points)

Amend the program `tableau/2` shown in class to also work with bi-implications, i.e., with propositional formulas involving the connective $\leftrightarrow$ (use the operator name `iff`, for *if and only if*). Show two sample queries of `tautology/1` involving bi-implications, one that succeeds and one that fails.

Question 3 (2 points)

Write a predicate `entails/2` that, when given a list of propositional formulas as the first argument and a single formula as the second argument, succeeds if and only if the latter is logically entailed by the former. Examples:

```
?- entails([p, q or r], (p or q) and (p or r)).
Yes

?- entails([neg p or q, p], neg q).
No
```

That is, the first query confirms that $p,\ q \vee r \models (p \vee q) \wedge (p \vee r)$, while the second query establishes that $\neg p \vee q, p \not\models \neg q$. Show at least two examples.

Question 4 (4 points)

Recall that to prove that a given formula φ is a tautology you start off a semantic tableau with φ (labelled as *false*) in the root. Suppose your proof fails, i.e., φ turns out not to be a tautology. Then you will have produced at least one branch that (*i*) cannot be closed and (*ii*) has been fully expanded (i.e., it consists of labelled atomic formulas only). One of the beautiful features of semantic tableaux is that any such branch corresponds to a *model* of $\neg\varphi$, i.e., to a *countermodel* of the claim that φ supposedly is a tautology.

Unfortunately, the implementation of the method of semantic tableaux given in class does not give us direct access to this model. However, there's an easy hacker's solution. Add the following rule to (the end of) your program:

```
tableau([], Atoms) :-
  list_to_set(Atoms, Model),
  write('Countermodel: '),
  writeln(Model), !,
  fail.
```

That is, this rule applies when the list of non-atomic formulas is empty and in case the current branch could not be closed (it is important that this rule comes *after* the rule used to close branches). The built-in predicate `list_to_set/2` removes duplicates from the list of labelled atoms. Now we can obtain the countermodel to a false claim of a given formula being a tautology (note that the query still fails, as it should):

```
?- tautology(neg (neg p or q) or p or neg r).
Countermodel: [t::r, f::p]
No
```

That is, if you make r true and p false (and if you give any truth value you wish to q), then the formula $\neg(\neg p \vee q) \vee p \vee \neg r$ will *not* be true (so it certainly cannot be a tautology).

What is not pretty about this solution, in Prolog terms, is that the countermodel is simply written on the screen rather than being bound to an output variable. So we can only look at it, but we cannot do anything with it. The objective of this exercise is to find a better solution, using one of the *search algorithms* we have seen. Write a predicate `model/2` that, when given a list of formulas as the first argument, returns a model of those formulas in the second argument position. In case there is no model (i.e., if the formula is a contradiction), the predicate should fail. Example:

```
?- model([p or q, neg p or neg q], Model).
Model = [f::q, t::p]
Yes
```

Use tableau *branches* (not nodes in the tableau) as states of your search space. That is, a state will consist of two lists, the labelled complex formulas and the labelled atoms on the branch in question. The tableau elimination rules correspond to moves. You will not be able to use our implementation of `tableau/2` directly, but you will be able to follow it very closely when implementing `move/2`. A goal state is a branch with an empty list of complex formulas and a list of atoms that does *not* allow you to close the corresponding branch. Show at least two examples of your program in action.

Hints: To test your solution, compare the models it generates with those generated using the hacker's solution. If implemented correctly, your predicate will return one solution for every tableau branch that cannot be closed (while the hacker's solution only gives you one of them). In some cases this means that the same model will get generated more than once.