Logic Programming Alan Smaill
14/10/2015

Tutorial for week 5 (19–23 Oct)

Non-logical features

This tutorial relates to concepts covered in LPN chapters 10–11.

1. **Cut**

   Consider the following facts:

   ```
   r(1). r(2).
   s(1). s(3).
   ```

   Draw the depth-first proof search trees for the following queries, showing each solution as well as each failing branch, and indicating which branches are discarded by cuts. You should ensure that your answer makes clear in which order the nodes of the tree are visited.

   (a) `r(X), !, s(Y)`

   (b) `r(X), s(Y), !`

   (c) `r(X), \+ s(X)`

   (d) `r(X), !, \+ s(X)`

   (e) `r(X), \+ r(X)`

   (f) `\+ \+ r(X)`

   (g) `\+ \+ r(3)`

2. **Input/Output**

   Here is a Sicstus Prolog version of `rot13/2` which converts a ground atom as first argument into its translation under the rot13 cipher (also available from web page):

   ```
   % rot13:  only for mode (+,?)

   rot13(Str, SR) :-
   atom_codes(Str,SL),
   maprot(SL, SL1),
   atom_codes(SR,SL1).

   maprot([],[]).
   ```

```
maprot([H|T],[HH|TT]) :- rot(H,HH),
                maprot(T,TT).

rot(C, C1) :-
      ( member(C, "abcdefghijklmABCDEFGHIJKLM"), C1 is C+13, ! )
    ; ( member(C, "nopqrstuvwxyzNOPQRSTUVWXYZ"), C1 is C-13, ! )
    ; C1 = C.
```

Define a predicate `translate/0` that reads in an atom, encodes it using `rot13/2`, writes it to the output stream, and then asks for another string. You might want to use the goal `nl/0` to insert newlines to separate the input and output.

3. **Using Cut**

   Without using cut, write a predicate split/3 that splits a list of integers into two lists: one containing the positive ones (and zero), the other containing the negative ones. For example:

   ```
   ?-  split([3,4,-5,-1,0,4,-9],P,N).
   ```

   should return:

   ```
   P  =  [3,4,0,4]
   N  =  [-5,-1,-9].
   ```

   Then make this program more efficient, without changing its meaning, with the help of the cut.

4. (**) **Assert/retract**

   In this problem we will use the `assert/1` predicate. This is covered in Lecture 5, and LPN chapter 11. Essentially, `assert/1` adds a fact or clause to the program dynamically. Many Prolog implementations use first-argument indexing, meaning that it is often a lot faster at finding the next rule to apply if the first argument of the predicate is known.

   Use `assert/1` to define a goal `buildrot/0` that always succeeds by building a dynamic predicate `rotA/2` that and use this relation instead of `rot/2` in the code above. This should lead to an improvement in the space/time efficiency of rot13 for long strings compared to a version that re-computes values for every character.

   Note: You will need to add a line

```
:- dynamic rotA/2.
```

to your Prolog file to declare `rotA/2` as a dynamic predicate.

5. (*) **Collecting Solutions**

   This problem uses the `findall`, `setof/3` and `bagof/3` and predicates (discussed in LPN chapter 11).

   Suppose we have the following statements of people and age information.

```
person(fred).
person(peter).
person(ann).
person(beth).
person(tom).
person(talullah).

age(peter,10).
age(ann,5).
age(beth,10).
age(tom,8).
```

   (a) Use `bagof` to work out the list of people whose age is not known.

   (b) Use `setof` to define a predicate that calculates:

      i. the set of all people with known ages

      ii. the set of all known ages.

   (c) Using `findall`, define a predicate `flatten/2` that takes a list of lists and flattens it to a list, so that on the success of `flatten(Xs,Ys)` each element of `Ys` is an element of an element of `Xs`.

   Experiment with `bagof` and `setof` instead of `findall`, with different inputs (and quantifiers) to see the differences in behaviour. (There are a couple of natural variants for each predicate.)