Logic Programming – Laboratory 3

Backtracking, The cut predicate !

Isabela Drămnesc

1 Questions

- What are accumulators? When and why do we use them?
- What is the difference between programs with accumulators and without it?
- Write the predicate that returns the length of a list:
  - without accumulator
  - with accumulator
- Simulate the Prolog's job for the predicate that uses accumulator (as we can see with the trace command)

2 Concepts

- Operations on lists
- Accumulators
- The difference between programs with accumulators and without it
- Backtracking
- The cut predicate !
  - the green cut
  - the red cut

3 Exercises

For each of the exercises below see what is the difference if we use the cut predicate or not, use the trace command and try to find out the answer of the question **"when do we have to use the cut predicate and why??"**

1. Modify the predicate member such that when we ask Prolog

$?-member(1,[2,1,1,3,1,4,1])$. will return one single solution.

```
member(X,[X|_]).
member(X,[_|B]):-member(X,B).
```

2. Verify if a list is a set or not (the predicate not)

Modify such that it works!

```
is_set([]).
is_set([X|T]):- not member(X,T), is_set(T).
```

3. The predicate which returns the intersection of two sets

```
inters([],X,[]).
inters([X|T],Y,[X|Z]):-member(X,Y),inters(T,Y,Z),!.
inters([X|T],Y,Z):-inters(T,Y,Z).
```

What happens if we do not use the cut predicate?

What happens if we use the predicate cut in other place? Like in the boundary condition?

4. Deletes the duplications and prints the reverse list (uses accumulator and the cut predicate)

```
deldup(L,X):-dupacc(L,[],X).
dupacc([],A,A).
dupacc([H|T],A,L):-member(H,A),!,dupacc(T,A,L).
dupacc([H|T],A,L):-dupacc(T,[H|A],L).
```

Modify the predicate such that it will delete the duplications from a list and returns the list (use accumulator and the cut !):

```
?-deldup2([1,1,1,2,2,3,4,5,4],L).
L=[1,2,3,4,5].
```

5. The minimum between two numbers using the cut predicate

```
min1(X, Y, X) :- X =< Y, !. /* green cut */
min1(X, Y, Y) :- X > Y.

min2(X, Y, X) :- X =< Y, !. /* red cut */
min2(X, Y, Y).
```

Explain why it **is** green cut and red cut?

6. Modify the program below such that the sorting by generate and test to print one single solution

```
is_permutation([], []).
is_permutation(List, [Prim | Remainder]) :- eliminate(Prim, List, L),
                                            is_permutation(L, Remainder).

eliminate(Elem, [Elem | Remainder], Remainder).
eliminate(Elem, [Prim | Remainder], [Prim | L]) :-
                                            eliminate(Elem, Remainder, L).

relation(X, Y) :- X =< Y.
```

```
ordered([_]).
ordered([Prim, Second | Remainder]) :- relation(Prim, Second),
                                       ordered([Second | Remainder]).

sorted(List, SortedList) :- is_permutation(List, SortedList),
                            ordered(SortedList).
```

7. Other exercises

a) Write a predicate for the union of two sets.

b) Given a list, split the list into two disjoint parts one part containing the elements less than a given element (e.g. the first element in the list) and one part containing the elements greater than or equal to this element. Then sort the two lists and append the results. (Quick-sort)

c) Given a list, find the minimum of the list, swap the minimum with the first position, repeat the steps for the remainder of the list. (Selection Sort)

4 Homework:

Implement operations on the queues using lists with differences.

Deadline: next lab.