CMPUT 325 Wi17 - NON-PROCEDURAL PROG LANGUAGES Combined LAB LEC Wi17

Assignment 3 - Prolog

CMPUT 325 - Assignment 3

Due Mar 23 at 11:55pm.

Submit your program as a single file named <yourID.pl> via eClass using the **submission link at the end of this page**.

In this assignment you solve some problems using SWI Prolog.

Note that + and - in arguments of predicates mean input and output parameters, respectively, and furthermore, ? means either + or -.

Restrictions: You can use all standard Prolog functions (see lecture notes),

except: you **cannot use the cut operator ! or the operator not** in this assignment.

#1 xreverse (1 mark)

Define the predicate xreverse(+L, ?R) to reverse a list, where L is a given list and R is either a variable or another given list.

Examples:

xreverse([7,3,4],[4,3,7]) should return true,

xreverse([7,3,4],[4,3,5]) should return false,

xreverse([7,3,4], R) should return R = [4,3,7].

Your program should generate only one solution, so if the user presses ";" the next answer should be "false".

#2 xunique (2 marks)

Define the predicate xunique(+L, ?O) where L is a given list of atoms and O is a copy of L where all the duplicates have been removed. O can be either a variable or a given list. The elements of O should be in the order in which they first appear in L.

Examples:

xunique([a,c,a,d], O) should return O = [a,c,d],

xunique([a,c,a,d], [a,c,d]) should return true,

xunique([a,c,a,d], [c,a,d]) should return false (because of wrong order),

xunique([a,a,a,a,a,b,b,b,b,b,c,c,c,c,b,a], O) should return O = [a,b,c],

xunique([], O) should return O = [].

Your program should generate only one solution, so if the user presses ";" the next answer should be "false".

#3 xdiff (1 mark)

Define the predicate xdiff(+L1, +L2, -L) where L1 and L2 are given lists of atoms, and L contains the elements that are contained in L1 but not L2 (set difference of L1 and L2).

Examples:

xdiff([a,b,f,c,d],[e,b,a,c],L) should return L=[f,d],

xdiff([p,u,e,r,k,l,o,a,g],[n,k,e,a,b,u,t],L) should return L = [p,r,l,o,g],

xdiff([],[e,b,a,c],L) should return L = [].

Your program should generate only one solution, so if the user presses ";" the next answer should be "false".

#4 removeLast (1 mark)

Define the predicate removeLast(+L, ?L1, ?Last) where L is a given non-empty list, L1 is the result of removing the last element from L, and Last is that last element. L1 and Last can be either variables or given values.

Examples:

removeLast([a,c,a,d], L1, Last) should return L1 = [a,c,a], Last = d,

removeLast([a,c,a,d], L1, d) should return L1 = [a,c,a],

removeLast([a,c,a,d], L1, [d]) should return false (why?),

removeLast([a], L1, Last) should return L1 = [], Last = a,

removeLast([[a,b,c]], L1, Last) should return L1 = [], Last = [a,b,c].

Your program should generate only one solution, so if the user presses ";" the next answer should be "false".

#5 clique (5 marks)

The clique problem is a graph-theoretic problem of finding a subset of nodes where each node is connected to every other node in the subset. In this problem, a graph will be represented by a collection of predicates, node(A) and edge(A,B), where A and B are constants. Edges are undirected but only written once, so edge(A,B) also implies edge(B,A).

For example, the following set of predicates represents a graph

```
node(a).
node(b).
node(c).
node(d).
node(e).

edge(a,b).
edge(b,c).
edge(c,a).
edge(d,a).
edge(a,e).
```

The set of nodes [a,b,c] is a clique, and so is every subset of it such as [a,c] or [b]. The set [a,d] and its subsets [a] and [d] are also a clique, etc. The empty set [] is also a clique.

You don't need to worry about ill-defined graphs: If there is an edge between a and b, then there will always be two facts node(a) and node(b) in the graph representation as well.

To solve this problem, first, by using the built-in predicate findall (see lecture notes), one can find all nodes of a graph. Thus, the clique problem can be solved as follows.

```
clique(L) :- findall(X,node(X),Nodes), xsubset(L,Nodes), allConnected(L).
```

After findall/3 is executed, the variable Nodes is bound to the set of the terms q such that

node(q) is true in your program. In the example above, we would get: Nodes = [a,b,c,d,e].

The subset relation can be defined as follows:

```
xsubset([], _).
xsubset([X|Xs], Set) :- xappend(_, [X|Set1], Set), xsubset(Xs, Set1).
xappend([], L, L).
xappend([H|T], L, [H|R]) :- xappend(T, L, R).
```

#5.1 allConnected (2 marks)

Use the predicates clique, xsubset and xappend above. Your job is to define the

predicate allConnected(L) to test if each node in L is connected to each other node in L. A node

A is connected to another node B if either edge(A,B) or edge(B,A) is true.

This is a simple (and very slow) "generate and test" approach to solving the clique problem.

Upon backtracking, the subset predicate in your program will generate all the subsets, and

each subset will be tested by your allConnected predicate.

allConnected(L) is true for an empty list, L= []. The recursive case is:

allConnected([A|L]) if A is connected to every node in L and allConnected(L). Thus, you need to define a predicate, say connect(A,L), to test if A is connected to every node in L.

#5.2 maxclique (3 marks)

Write a predicate maxclique(+N, -Cliques) to compute all the maximal cliques of size N that are contained in a given graph. N is the given input, Cliques is the output you compute: a list of cliques. A clique is maximal if there is no larger clique that contains it. In the example above, cliques [a,b,c] and [a,d] are maximal, but [a,b] is not, since it is contained in [a,b,c].

Examples (using the graph above):

maxclique(2,Cliques) returns Cliques = [[a,d],[a,e]]

maxclique(3,Cliques) returns Cliques = [[a,b,c]]

maxclique(1,Cliques) returns Cliques = []

maxclique(0,Cliques) returns Cliques = []

Your program should generate only one solution (the list Cliques). If the user presses ";" the next answer should be "false".

Different sets of facts about node and edge will be provided by us for testing your code.
Do not include any facts about node and edge in your program! Keep them in a separate file that you load into Prolog for your testing only, and experiment with different graphs.

End of assignment 3

Submission status

This assignment will accept submissions from **Thursday, 16 March 2017, 12:00 AM**