CS 4337

Programming Assignment 2

Prolog

Due: March 13 11:59 p.m.

Submit a zip file called netid.zip containing the answers to the questions below and your prolog file.

1. What is it called for the variable matching process in Prolog?

Unification, which process variable matching with the variables start with capital letters.

|  |  |  |
| --- | --- | --- |
| What would it be the result of the following prolog query? | | |
| 2. | ?- p(X, f(Y), a) = p(a, f(a), Y). | X=a,Y=a |
| 3. | ?- p(X, f(Y), a) = p(a, f(b), Y). | false |
| 4. | ?- [a, b, c] = [X | Y]. | X=a,  Y=[b,c]. |
| 5. | ?- [a, b, c] = [X, Y | Z]. | X=a,  Y=b,  Z=[c]. |
| 6. | ?- [a, b, c] = [X, Y, Z | T]. | X=a,  Y=b,  Z=c,  T=[]. |
| 7. | ?- transpose([[1,2,3],[4,5,6],[7,8,9]], Ts). | Ts=[[1,4,7],[2,5,8],[3,6,9]] |
| 8. | ?- (31 is (36-5)). | true |

9. Consider the following query. How many answers will it have? \_1\_

?- member(X,[1,2,3,4]), Y = X\*X, Y<10.

It will have one answer. Lack of ‘is’ statement.

Result: X=1,

Y=1\*1

10. What is a correct definition of negation in Prolog?

Negation in Prolog is implemented based on the use of cut. Actually, negation in Prolog is the so-called *negation as failure*, which means that to negate p one tries to prove p (just executing it), and if p is proved, then its negation, not(p), fails.

11. Given the following facts and predicates

mystery(A,B) :- mystery(A,[],B).  
mystery([X|Y],Z,W) :- mystery(Y,[X|Z],W).  
mystery([],X,X).

What would be the result of the query below?  
?- mystery([1,2,3], A).

The output will be 3.

12. Given the following facts and predicates

mystery(X,[X|R],R).  
 mystery(X,[F|R],[F|S]) :- mystery(X,R,S).

What is the result L of the following query?  
?- mystery(1,[1,2,3], L).

The result is L = [2,3]

13. What is the result of the following Prolog statements?  
  
?- assert(test(N,R) :- R is N\*N). ---- true.  
?- maplist(test,[1,2,3,4],Rss). ---- Rss = [1,4,9,16].

14. Explain the behavior or goal of the following program xyz/3.  
  
xyz(X,[X|R],R).  
xyz(X,[F|R],[F|S]) :- xyz(X,R,S).  
  
?- xyz(X,[1,2,3],L).

Output: X=1,

L=[2,3].

The program takes an element X out of a list [X|R] and resulting in a list R.

15. What is the arity of the following compound term?

love(2,loves(richard, sarah), X).

arity = 3.

Define and test the Prolog predicates described below. **Each of your predicates must have the same name and signature as the examples below**. Your predicates must behave properly on all instances of valid input types. Your submission should consist of a single source code text file that includes all facts, predicate definitions, and propositions.

Your file should be named <your\_net\_id>.pl

You may find additional Prolog language help at the following links:

* SWI-Prolog manual: <http://www.swi-prolog.org/pldoc/refman/>
* SWI-Prolog documentation: <http://www.swi-prolog.org/pldoc/index.html>
* Learn Prolog Now: <http://www.learnprolognow.org/>

**1) Odd Multiple of 3**

Define a predicate **oddMultOf3/1** that determines whether an integer is an odd multiple

of 3. A user should be able to enter the predicate with an integer, e.g. **oddMultOf3(42)**

and evaluate to either **true** or **false**. If the given parameter is not an integer, your

predicate should display the message “**ERROR: The given parameter is not an**

**integer**”.

Input/Output:

?- oddMultOf3(171).

true.

?- oddMultOf3(100).

false.

?- oddMultOf3(12).

false.

?- oddMultOf3(4.2).

ERROR: The given parameter is not an integer

?- oddMultOf3(-9).

true.

**2) List Product**

Define a predicate **list\_prod/2** that takes a list of numbers as a first parameter and

determines the product of all of the list elements in the second parameter. Your

predicate should have the signature **list\_prod(List, Number)**. The product of an

empty list should be zero.

Examples:

?- list\_prod([4,3], Product).

Product = 12.

?- list\_prod([5,5,5], Product).

Product = 125.

?- list\_prod([7,8,0,13], Product).

Product = 0.

?- list\_prod([6,2,5,10], Product).

Product = 600.

?- list\_prod([], Product).

Product = 0.

**3) Segregate**

Define a predicate **segregate/3** that takes a list of integers as an argument and

generates two lists, the first containing the even numbers from the original

list and the second sublist containing the odd numbers from the original list. Your

predicate should have the signature **segregate(List, Even, Odd)**.

Examples:

?- segregate([8,7,6,5,4,3], Even, Odd).

Even = [8,6,4]

Odd = [7,5,3]

?- segregate([7,2,3,5,8], Even, Odd).

Even = [2,8]

Odd = [7,3,5]

?- segregate([-4,11,-7,9,0], Even, Odd).

Even = [-4,0]

Odd = [11,-7,9]

?- segregate([5,13,29], Even, Odd).

Even = []

Odd = [5,13,29]

?- segregate([], Even, Odd).

Even = []

Odd = []

Even = Odd, Odd = []

**4) Route**

Given the following graph of possible flights between seven US cities:



(graph taken from the web site of the American Mathematical Society, <http://www.ams.org>)

route

Define a predicate **route/3** that takes two cities as arguments and finds the routes to get from city A to a city B. Your predicate should have the signature **route(cityA, cityB, Route)**.

Examples:

?- route(seattle, boston, X).

X = [seattle, omaha, atlanta, boston] ;

false.

?- route(fresno, atlanta, X).

X = [fresno, seattle, omaha, atlanta] ;

X = [fresno, albany, seattle, omaha, atlanta] ;

X = [fresno, albany, dallas, seattle, omaha, atlanta] ;

false.

?- route(albany, atlanta, X).

X = [albany, seattle, omaha, atlanta] ;

X = [albany, dallas, seattle, omaha, atlanta] ;

false.

?- route(boston, atlanta, X).

false.

**5) Genealogy**

Design a set of predicates that encode genealogical relationships.

• **male(X) - X is male.**

• **female(X) - X is female.**

• **parent(X,Y) - X is the parent of Y.**

• **mother(X,Y) - X is the mother of Y.**

• **father(X,Y) - X is the father of Y.**

• **child(X,Y) - X is the child of Y.**

• **sibling/2 (reflexive)**

• **grandparent(X,Y) - X is the grandparent of Y.**

• **grandmother(X,Y) - X is the grandmother of Y.**

• **grandfather(X,Y) - X is the grandfather of Y.**

• **grandchild(X,Y) - X is the grandchild of Y.**

• **grandson(X,Y) - X is the grandson of Y.**

• **granddaughter(X,Y) - X is the granddaughter of Y.**

Note: Your definitions should avoid infinite recursion and return a single result set. For

example, **siblings(X,Y)** should queries should return a single result set, i.e. *not* **X=bob**,

**Y=joe**; **X=joe**, **Y=bob**.

Note: The Knowledge Base of people below is for example only. You are just responsible for the definitions of predicate rules. The Knowledge Base used for grading will be different.

% Knowledge Base

male(adam).

male(bob).

male(brett).

male(charles).

male(chris).

male(clay).

female(ava).

female(barbara).

female(betty).

female(colette).

female(carrie).

parent(adam,bob).

parent(adam,barbara).

parent(ava,bob).

parent(ava,barbara).

parent(bob,clay).

parent(barbara,colette).

Input:

?- mother(ava,Kid).

Kid = bob;

Kid = barbara.

?- father(bob, Kid).

Kid = clay.

?- parent(X, colette).

X = barbara.

?- sibling(X,Y).

X = bob,

Y = barbara;

?- grandparent(GParent,colette).

GParent = adam;

GParent = ava.

?- grandmother(X, clay).

X = ava ;

?- grandfather(X, clay).

X = adam .

?- grandchild(GChild, adam).

GChild = clay;

GChild = colette.

?- grandson(Grandson, adam).

Grandson = clay.

?- granddaughter(X, ava).

X = colette .