CSCE 625
Programing Assignment #2 - PROLOG
due: Thurs, Dec 7, 2017, 1:00pm

This assignment is intended to give you practice programming in Prolog, which will rely on your understanding of back-chaining and unification. The questions represent different types of use cases for Prolog, from database-like queries, to knowledge representation (concept definitions), to numerical computations, to decision-making and problem-solving.

I recommend you use SWI Prolog (`www.swi-prolog.org`). You can run Prolog on the departmental linux servers like compute.cs.tamu.edu by running '/home/faculty/ioerger/bin/swipl' at the command line, or you can download and install it yourself. For a brief intro, see http://faculty.cs.tamu.edu/ioerger/prolog.txt

Note: for this project, you should not use any special Prolog operators, like ';', '|', or '->'. The only operator you will need is negation, '\+'.

1. Write rules in Prolog to infer various kinship relationships in terms of basic predicates like parent(X,Y) and female(X) and male(Y). Input the following facts about people on The Simpsons:

```
parent(bart,homer).
parent(bart,marge).
parent(lisa,homer).
parent(lisa,marge).
parent(maggie,homer).
parent(maggie,marge).
parent(homer,abraham).
parent(herb,abraham).
parent(tod,ned).
parent(rod,ned).
parent(marge,jackie).
parent(patty,jackie).
parent(selma,jackie).

female(maggie).
female(lisa).
female(marge).
female(patty).
female(selma).
female(jackie).

male(bart).
male(homer).
male(herb).
male(burns).
male(smithers).
male(tod).
male(rod).
male(ned).
male(abraham).
```

Write rules to define the following relationships: brother(), sister(), aunt(), uncle(), grandfather(), granddaughter(), ancestor(), descendant(), and unrelated(). Use the convention that relation(X,Y) means "the relation of X is Y)". For example, uncle(bart,herb) means the uncle of bart is herb.

Use your rules to answer the following queries:

```
?- brother(rod,X).
X = tod ;
?- sister(marge,X).
X = selma ;
X = patty ;
?- aunt(X,patty).
X = bart ;
X = lisa ;
X = maggie ;
?- uncle(bart,X).
X = herb ;
?- grandfather(maggie,X).
X = abraham ;
?- granddaughter(jackie,lisa).
true
?- ancestor(bart,X).
X = homer ;
X = marge ;
X = abraham ;
X = jackie ;
?- unrelated(tod,bart).
true
?- unrelated(maggie,smithers).
true
?- unrelated(maggie,selma).
false
```

2. Using the following database, write a Prolog query to find all the surgeons who live in Texas and make over $100,000/yr. You will have to add some additional data, such as about different types of surgeons, or city-state relationships.

```
occupation(joe,oral_surgeon).
occupation(sam,patent_laywer).
occupation(bill,trial_laywer).
occupation(cindy,investment_banker).
occupation(joan,civil_laywer).
occupation(len,plastic_surgeon).
occupation(lance,heart_surgeon).
occupation(frank,brain_surgeon).
occupation(charlie,plastic_surgeon).
occupation(lisa,oral_surgeon).

address(joe,houston).
address(sam,pittsburgh).
address(bill,dallas).
address(cindy,omaha).
address(joan,chicago).
address(len,college_station).
address(lance,los_angeles).
address(frank,dallas).
address(charlie,houston).
address(lisa,san_antonio).

salary(joe,50000).
salary(sam,150000).
salary(bill,200000).
salary(cindy,140000).
salary(joan,80000).
salary(len,70000).
salary(lance,650000).
salary(frank,85000).
salary(charlie,120000).
salary(lisa,190000).
```

3. Write a prolog function to remove duplicates from a list:

```
  ?- remdups([1,3,4,2,4,3,6,8,6,5,4,2,3,4,9],X).
  X = [1, 8, 6, 5, 2, 3, 4, 9]
```

For this problem, you will need to know how lists are represented in Prolog, and how unification of [X|Y] with a list binds X to the head and Y to the tail. see: https://en.wikibooks.org/wiki/Prolog/Lists

hint: define remdups() recursively; use member(.,.)

4. Implement prime factorization in Prolog. Here is an example:

```
  ?- factor(120,M).
  M = [5, 3, 2, 2, 2]

  ?- factor(7,P).
  P = [7]
```

Note: you can use a simple sieve algorithm that iterates from 2 up to N (or sqrt(N)) and tests for divisibility using mod. for example:

```
  divisible(N,X) :- M is N mod X,M=0.
```

You will probably want to write an auxilliary function that determines the smallest factor for a given number, and then recursively builds up a list of factors as you go.

5. Write a predicate to generate all bit vectors of a specified length:

```
  ?- bitvec(3,K).
  K = [0, 0, 0] ;
  K = [0, 0, 1] ;
  K = [0, 1, 0] ;
  K = [0, 1, 1] ;
  K = [1, 0, 0] ;
  K = [1, 0, 1] ;
  K = [1, 1, 0] ;
  K = [1, 1, 1] ;
  No
```

Use this to generate a system of bit vectors as 'codes', such as all those bit vectors of length 5 with exactly 2 bits on. (This is a common system used in bar-coding systems, such as on store products or postal mail, where each code is represented by a pattern of bars of alternating width or height and represents a different digit).

```
  ?- code(5,2,X).
  X = [0, 0, 0, 1, 1] ;
  X = [0, 0, 1, 0, 1] ;
  X = [0, 0, 1, 1, 0] ;
  X = [0, 1, 0, 0, 1] ;
  X = [0, 1, 0, 1, 0] ;
  X = [0, 1, 1, 0, 0] ;
  X = [1, 0, 0, 0, 1] ;
  X = [1, 0, 0, 1, 0] ;
  X = [1, 0, 1, 0, 0] ;
  X = [1, 1, 0, 0, 0] ;
  No
```

6. Write a logic program to compute the zeros of sin(x), that is the values of x such that sin(x)=0, using Newton's method.

Newton's method says that, given a function f(x) and an initial guess or starting point x0, it may be iterated to get closer and closer to a zero by using this update equation: x_i+1 = x_i - f(x_i)/f'(x_i), where f'(x) is the derivative.

You may assume a fixed value for f(x) and f'(x), i.e. sin(x) and cos(x). Write your predicate as **sin_zero(X,Y)**, where X is an input guess, and Y is the output value, iterated to be within some threshold of zero, e.g. -0.0001<sin(Y)<0.0001. Of course, the zeros of this function are expected to be 0, pi, 2*pi, etc. So the query sin_zero(3,Y) should return Y=3.1415, and sin_zero(10,Y) should return Y=9.4248.

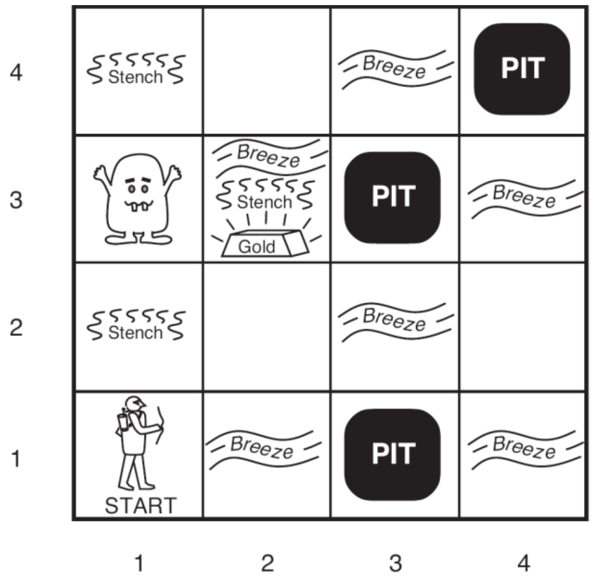
```
7. SEND + MORE = MONEY is a classical "cryptarithmetic" puzzle: the
variables S, E, N, D, M, O, R, Y represent digits between 0 and 9, and
the task is finding values for them such that the following arithmetic
operation is correct:

     S  E  N  D
+    M  O  R  E
---------------
  M  O  N  E  Y

Moreover, all variables must take unique values, and all the numbers
must be well-formed (which implies that M > 0 and S > 0).

By the way, if you want to print something in Prolog, you can use the
format(A,B) predicate, where A is a format string and B is a list of values.
For example: format('the square of ~s is ~s~n',[3,9]) prints "the square of 3
is 9" followed by a carriage return.  See the documentation for more details.
```

8. Write rules in Prolog to determine the best move in Tic-Tac-Toe for any given board configuration. Assume that the position of pieces is given by a predicate 'p'. For example, consider the following board state:

```
x  .  x
.  .  .
o  .  o

% assert these facts as the state description
p(x,1,1).
p(x,1,3).
p(o,3,1).
p(o,1,3).

?- ttt_move(x,R,C). % query
go for win!     % printed as a side-effect
R = 1, C = 2 ;  % solution

Here is another example...

x  .  .
.  .  x
o  .  o

p(x,1,1).
p(x,2,3).
p(o,1,3).
p(o,3,3).

?- ttt_move(x,R,C).
move to block opponent!
```

```
R = 3, C = 2 ;

?- ttt_move(o,R,C).
go for win!
R = 3, C = 2 ;
```

9. Write decision-making rules in Prolog for the Wumpus World. Assume the world is described in terms of facts about sense inputs (*stench* and *breeze*), along with a predicate called *visited*. Regardless of the current location of the agent, the available actions are to go to any room adjacent to one that has been visited. Primarily, the agent would prefer to explore rooms it can infer to be safe. (also assume that a wumpus, pit, and gold will never be in the same room together). There might be more than one room with gold or pits, but only one wumpus (which doesn't move). Note that your rules should work for any 4x4 cave, regardless of the location of the pits and wumpus and gold (don't just assume they are like below).



```
As shown in the book, assume the coordinate system is (C,R) = (column,row).
For example, the gold in in room (2,3).
```

example scenario.

```
visited(1,1).
visited(2,1).
visited(1,2).
stench(2,1).
breeze(1,2).

?- candidate(X,Y). % just list unvisited rooms adjacent to visited ones
X = 1, Y = 3 ;
X = 2, Y = 2 ;
X = 3, Y = 1 ;
```

```
?- move(X,Y). % the best choice, because it can be inferred to be safe
X = 2, Y = 2 ;
```

(Note, your implementation might give solutions in different order, or the same solution multiple times; that is OK).

Now suppose we go to (2,2) and observe no stench or breeze.

```
visited(1,1).
visited(2,1).
visited(1,2).
stench(2,1).
breeze(1,2).
visited(2,2).

?- candidate(X,Y).
X = 1, Y = 3 ;
X = 3, Y = 1 ;
X = 2, Y = 3 ;
X = 3, Y = 2 ;
?- move(X,Y).
X = 3, Y = 2 ; % because it is inferred to be safe
X = 2, Y = 3 ; % another solution - both are safe
```

Here is another scenario, showing location of wumpus, gold, pits, and visited rooms (W,G,P,V):

```
. . . V
. . W V
. G P V
. . P V

visited(4,1).
visited(4,2).
visited(4,3).
visited(4,4).
stench(4,2).
breeze(4,3).
breeze(4,4).

?- ww_move(X,Y).
X = 3, Y = 1 ;
```

<u>What to Turn in</u>

- You will submit your Prolog code for testing using the web-based CSCE *turnin* facility, which is described here: **https://wiki.cse.tamu.edu/index.php/Turning_in_Assignments_on_CSNet**