CSC 351 – Comparative Programming Languages

Additional Prolog Problems #2 – Prolog Continued

Due: Tuesday Feb. 17 11:59pm

What: Two files containing information detailed in "Deliverables" section of each part of lab – both should be word processed documents – the first section file should be called XXXrelatives (where XXX are your initials) and the second section file should be called XXXnqueens

Relatives – Section I - Purpose:

The objectives of these exercises are for you to become familiar with the Prolog system you will be using, including its trace/debug features, and for you to practice writing Prolog rules and querying the Prolog database. By tracing Prolog's execution of your queries, you will be able to see backtracking in action.

Description:

The first part of this exercise requires that you look through a set of Prolog rules and find the syntax errors in them. You will be provided with a file of facts and rules that have some errors. In lab, you will look through the file and identify as many of the errors as possible. You will also look at a set of queries relative to this file and decide on what you think the answer should be. You will later test your expectations on the computer. It is important that you think about the answers first, because a difference in the computer results might be due to a subtle error that you missed.

The second part requires you to write and test some simple rules.

Steps:

1) Look though the first print out of facts and rules and identify/correct as many errors as possible. Some errors in Prolog can be rather subtle because of the structure of the language. NOTE: these facts and rules are provided on the last pages of this lab writeup.
2) Determine what you think the answers are for the set of queries below. You should do this with the expectation that all errors have been removed.
3) Copy the file `~mcconnel/CSC351/Prolog/Lab1/sample.pl` into your own directory and use a text editor to make the corrections you identified in step 1.
4) Ask the following set of queries and compare the results with your expectations from step 2. If there are differences determine whether your expectations or the Prolog is wrong.
   Queries:
   - Is Tom happy?
   - Who is happy?
   - Who is playing what game/sport?
   - Who has supplies?
   - Who has pop?
   - Who has what?
5) Look through the second print out of facts and rules. You will find facts with the following names and structure: `father(Father, Child)`, `male(Person)`, `mother(Mother, Child)`, `female(Person)`, `citizen_of(Person, Country)`.
6) Write Prolog statements to represent the following English sentences (you will have to complete some of these definitions yourself):
   - A person is a citizen of a country if the person's father is a citizen of the country.

One person is the sibling of another if they have the same parents.
One person is the sister of another if …
One person is the brother of another if …
One person is the aunt of another if …
One person is the uncle of another if …

7) Copy the file `~mcconnel/CSC351/Prolog/Lab1/people.pl` into your own directory and use a text editor to enter the rules you developed in step 6.
8) Query the extended database to get Prolog to determine who are sisters, brothers, aunts, and uncles. Also, find out from the database the names of persons who are citizens of Norway.

Deliverables:

For the first part you should turn in your corrected facts and rules, and a write up showing what your expected and actual results were. Where there is a difference between your expectation and first computer answer briefly indicate either what gave you the wrong expected answer or what error you missed. For the second part, print out your complete set of facts and rules and the answers you got from step 8. (Note: you will have to take screen shots for some of the deliverables.)

{Section 1 of lab created by Jeffrey McConnell, based on a lab by Tom Hankins, West Virginia Graduate College, South Charleston, WV)

****

nQueens – Section 2 - Purpose:

To explore the "generate and test" paradigm in Prolog.

Description:

The eight queens problem is a classic in computer science. It is frequently used to illustrate recursion and backtracking. In general, this problem is known as the N queens problem where the task is to place N queens on an N by N chessboard so that no two queens attack each other. This problem is solvable for values of N greater than 3.

For this lab exercise, you will create a Prolog program to solve the N queens problem. Let's begin by exploring this problem in a little more detail. A chess queen attacks all squares that are on the horizontal, vertical, and diagonal lines from where it sits. Because of the horizontal and vertical attacks, we know that there can be at most one queen per row and column of the chessboard. This means we can represent the position of the queens as a permutation of the numbers from 1 to N to indicate the columns where the queens are positioned. For example, the list [3, 1, 4, 2] represents the solution:

		Q	
Q			
			Q
	Q		

We only need to check for diagonal attacks either above or below each queen, but not both ways. If we checked both ways, we would find a pair of attacking queens twice. You should also see that for a queen at (X, Y), the diagonal elements below it are at (X+1, Y-1), (X+1, Y+1), (X+2, Y-2), (X+2, Y+2), etc.

The generate portion of this project will create permutations of the numbers between 1 and N, and the test portion will check to see if these permutations are a solution. In class, we looked at permutation sort, which you can use as the basis for generating the permutation. To complete this lab, you need to write predicates that will check to see if a board is safe (in other words, no queen is attacked) and write

predicates that will output the solution. Your output can just be the queens in properly aligned positions with vertical bars and horizontal dashes to make the spacing clearer.

Steps:

1) Copy the file (it contains the permutation code and the code to generate a list of numbers in the proper range) found at:

`~mcconnel/CSC351/Prolog/Lab2/queens.pl`

2) Write the definition of the `safe` predicate, which will probably be easiest if you also create and use another predicated called `attacked`.

3) Write the definition of the predicates necessary to output the queens on a "chessboard."

Deliverables:

Paper: You should turn in a print out of your facts and rules as well as a script showing that your program works.

In addition, you must provide a detailed algorithm that describes your strategy and how and why it works

FILES AND RULES for correction/inspection, Section 1 problem:

```
:- discontiguous(has/2).
:- discontiguous(playing/2).


happy(tom):- watching(tom,bills), has(Tom,supplies).
happy(mary) :- can_pay(mary,bills).
happy(mary) :- \+ (owes_for(mary,bills)).
happy(dave) :- weather(sunny), temperature(hot).
happy(anne) :- happy(dave), \+ (happy(tom)), happy(mary).
happy(anne) - \+ (happy(dave)), happy(tom), \+ (happy(mary)).
happy(anne) :- \+ (happy(dave)), happy(tom), happy(mary).
happy(steve) :- competing(steve,jim,tennis).
happy(steve) :- competing(steve,sue,golf).
happy(steve) :- competing(steve,Anyone,baseball).
happy(carol) := watching(carol,sabres), has(carol,supplies).
has(X,supplies):- has(X,pop), has(X,pretzels).
has(X,supplies):- has(X,pop), has(X,chips).
watching(tom,bills):- is_on(toms_tv), playing(bills,football).
watching(carol,sabres):- is_on(carols_tv), playing(sabres,hockey).
can_pay(X,Y) :- has(X,money), owes_for(X,Y).
competing(X,Y,Sport) :-
                playing(X,Sport),playing(Y,Sport),opponents(x,y).
playing(steve baseball).
playing(bob, baseball).
opponents(steve, bob)
```

```
playing(jim, tennis).
playing(sue, golf).
is_on[toms_tv].
is_on(carols-tv).
playing(bills,football).
has(tom,pop).
has(mary, money).
weather(Sunny).
temperature(mild).
has(tom,pretzels).
playing(sabres,hockey).
has(carol, chips).
has(carol,pop).
owes_for(dave,dinner).
father(haakon,olav).
father(olav,harald).
father(olav,ragnhild).
father(olav,astrid).
father(harald,christian).
mother(maud,olav).
mother(martha,harald).
mother(martha,ragnhild).
mother(martha,astrid).
mother(ragnhild,edelhart).
mother(astrid,dagne).
male(haakon).
male(olav).
male(harald).
male(christian).
male(edelhart).
female(maud).
female(martha).
female(ragnhild).
female(astrid).
female(dagne).
citizen_of(olav,norway).
```