CSCE 420, Howework 4 (HW4)
due: Tues, Nov 28, 11:59 pm

Turn-in answers as a Word document (HW4.docx or .pdf) and commit/push it to your class github repo (in your homework_4/ directory).

When pushing the final version of your HW1, also run the command:
git tag "HW4" && git push origin main "HW4"
This will be used to record time of submission for late penalty when applicable.

All homeworks must be typed, *not* hand-written and scanned as a photo.

This project gives some simple practice with programming in Prolog. You can use 'gprolog' on compute.cs.tamu.edu, or you can install gprolog or SWI-Prolog on your own machine, but the programs must still work with 'gprolog' on compute.cs.tamu.edu.

These questions are intended to show you several different paradigms or use-cases of Prolog.

Remember that Prolog works by back-chaining (e.g. putting antecedents on a goal stack), along with unification. The order of rules and facts is important; when trying to solve a predicate, it will try unifying it to facts or the head (consequent) of rules in the order that they appear in the program; and antecedents will be evaluated in left-to-right order (as if pushed onto stack in right-to-left order). Recall that 'is' can be used to compute variable binding by arithmetic, such as 'X is A+1' as an antecedent. See the documentation for arithmetical comparison operators like equal (=:=), not equal (=\=), greater than (>), and less-than-or-equal-to (=<, *not* <=). For more info, see the textbook, lecture slides, tutorial posted on the course web site, and online Prolog documentation. You might find 'trace.' and 'notrace.' to be helpful for debugging programs, which allows you to see the calls during back-chaining.

For more on Prolog, see the mini-tutorial: **https://people.engr.tamu.edu/ioerger/prolog.txt** as well as lecture slides and textbook.

What to Turn In:

- Put all your predicates in one file called '**solutions.pl**'. We will load this and test it by calling your predicates on other test inputs.
- Put examples of the input and output for each question in a file called '**transcript.txt**'
- Check these into your **homework_4/** directory in your github repository.

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(),` and `ancestor()`. 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 (typing ‘;’ after each solution to get the next):

```
?- brother(rod,X).

?- sister(marge,X).

?- aunt(X,patty).

?- uncle(bart,X).

?- grandfather(maggie,X).

?- granddaughter(jackie,D).
```

```
?- ancestor(bart,X).
```

- Hint 1: To utilize an existentially quantified variable, just put it in one of the antecedents. For example, in grandfather(X,Y), Z would be the parent inbetween.
- Hint 2: For brother(X,Y), you can include "X\=Y" (X≠Y) in the body of the rule (as an antecedent) to exclude someone from being their own brother.
- Hint 3: The rule for aunt() can use sister().
- Hint 4: ancestor() can be defined with 2 rules: a base case, and a recursive case.

2. Using the following database, write Prolog queries for:

- query2a: find all lawyers.
- query2b: find all surgeons who live in California.
- query2c: find all the surgeons who live in Texas and make over $100,000/yr.

(Hint: you can define a rule for each query and put it in your solutions.pl program file, like this: "query2a(X) :- ...". Then you can enter it as a query at the interactive Prolog prompt to get all values for X.)

You will have to add some additional facts or rules to define who is a surgeon (including different types of surgeons), who is a lawyer, and what state each person lives in.

```
occupation(joe,oral_surgeon).
occupation(sam,patent_lawyer).
occupation(bill,trial_lawyer).
occupation(cindy,investment_banker).
occupation(joan,civil_lawyer).
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. Consider the following database of PROLOG facts:

```
subject(algebra,math).
subject(calculus,math).
subject(dynamics,physics).
subject(electromagnetism,physics).
subject(nuclear,physics).
subject(organic,chemistry).
subject(inorganic,chemistry).

degree(bill,phd,chemistry).
degree(john,bs,math).
degree(chuck,ms,physics).
degree(susan,phd,math).

retired(bill).
```

Write a predicate `canTeach(X,Y)` that defines which persons X can teach a class Y a given subject, which requires that they have a phd in the relevant field Z. For example, Susan can teach calculus because she has a phd in math. Note that, since Z is not mentioned in the head of the clause, it will effectively be treated like an existentially quantified variable in the antecedents, which be matched to any academic field during the back-chaining.

Show all solutions that are generated for the query `canTeach(X,Y).`

Modify the *canTeach* rule (call it *canTeach2*(X,Y)) to exclude people who are retired. (hint: this requires *negation*). Show all solutions.

Modify the *canTeach2* rule (call it *canTeach3*(X,Y)) to allow persons with a PhD or an MS degree who are not retired to teach a class related to their degree. Show all solutions. (Hint: use 2 rules).

4. Prolog can be used to generate a set of combinations. First, write a FOL sentence describing **octal codes**. Octal codes are a set of 3 variables where each is a bit. (Of course, bits are 0 and 1). Then code it in Prolog, and use Prolog to generate all the octal codes.

Define the 3-argument predicate for '*octal_code(A,B,C)*' as a sentence in FOL.

Next, express this as a rule in Prolog (along with facts defining bits).

Finally, query '`octal_code(A,B,C)`' interactively, and show all solutions (by typing ';' after each).

Also, you can print things out as a side-effect by adding `format('~w~w~w~n',[A,B,C])` as the last antecedent in the rule.

Use back-chaining to **explain** why the solutions are generated in the order they are. Note that typing ';' after a solution is analogous to reaching a goal on the stack that fails, forcing back-tracking.

5. Solving the map-coloring CSP (from Ch. 6) using Prolog. Write a rule to solve the color of the 7 states of Australia. Use a "**generate and test**" approach, where the initial antecedents bind each of the variables to one of the 3 possible colors, and the remaining antecedents enforce the adjacency constraints (states sharing a border cannot be same color). Hint: if you want to say that Western Australia is not the same color as Northern Territories, you can use the inequality operator 'WT\=NT'.

Here is what *a* solution should look like: (of course, there can be others)

```
?- mapcolor(WA,NT,SA,Q,NSW,V,T).
WA = Q, Q = V, V = T, T = red,
NT = NSW, NSW = green,
SA = blue .
```

To force Western Australia to be green and get a different solution, one could do it this way:

```
?- WA=green, mapcolor(WA,NT,SA,Q,NSW,V,T).
WA = Q, Q = V, V = green,
NT = NSW, NSW = T, T = red,
SA = blue .
```

Note that WA and Q must be the same color, so trying to assign one to red and the other to green fails:

```
?- WA=red, Q=green, mapcolor(WA,NT,SA,Q,NSW,V,T).
false.
```

Show that your code gets the same answers for these queries.