Project 4

Due July 30, 2017 at 11:59 PM

Overview

This project specification is subject to change at any time for clarification. For this programming assignment you will be programming in Prolog with a partner. This assignment is broken into four parts. The first part is a fairly straight forward warmup. Part 2 requires you to use Prolog's control mechanism to find a solution to a card game. Part 3 requires that you use Prolog to implement a predicate to color a map. Part 4 is extra credit.

Notes

The command to use is gprolog. The manual for gprolog is available at:
http://www.gprolog.org/manual/gprolog.html

- You can either use \+ for not or, define a mynot function as follows:

  mynot(A) :- A, !, fail.

  mynot( ).

- You may not use assertz or related predicates to modify the database.
- Read Chapter 4 of your textbook. It contains several examples.

- Read Sections 8.1 and 8.2 in the Prolog text for common mistakes to avoid. If your program does not work as you think it should, go through the checklists before doing anything else. In particular, beware of typos such as:
  - space between a predicate name and '('.
  - [A, X] instead of [A | X], or vice versa.
  - uppercase instead of lower case, or vice versa.
  - period instead of command, e.g., a(H) :- b(H).C(H). is quite different from a(H) :- b(H), C(H).
  - use quotes within consult — e.g., consult('multi.p'). — if your file name contains anything other than letters.
  - use is for arithmetic assignment and = for binding (make sure that you understand this important difference!).

Also, do not forget to handle base cases (e.g., empty list) and to specify cases in the right order.

- The test program will be provided. It exercises the predicates that you write; hence there is no test data. Details regarding the test program will appear on Canvas or Piazza. You may define additional helper predicates that your main predicates use. Be sure, though, to name the main predicates as specified since the test program uses those names.

- When developing your program, you might find it easier to test your predicates first interactively before using the test program.

Part 1: Simple Queries

You will be given sets of facts of the following form:

1. course(number, prereq, units) Here number is an atom denoting the course number (for instance, ecs140a), prereq a list of course numbers, and units a number indicating the number of units associated with a course.

2. student(name, current_courses, courses_taken). Here name is an atom denoting a student's name, current_courses is a list of courses the student is currently taking, and courses_taken is a list of courses that the student has taken.

3. instructor(name, course_list). Here name specifies the name of the instructor who teaches the set of courses in the list course_list.

An example database of facts is shown below:

course(ecs40, [ecs30], 4).

course(ecs122a, [ecs20, ecs60], 3).

...

student(john, [ecs60, ecs120], [ecs30, ecs40]).

...

instructor(jim, [ecs30, ecs60]).

Write the following separate queries:

a) Find all courses with 3 or 4 credits (fc_course).

b) Find all courses whose immediate pre-requisite is a course (imprereq).

c) Find names of all students in a course (students).

d) Find the names of all students who have not met the perquisites for the courses they are currently taking (students_prereq). (This will involve finding not only the immediate prerequisites of a course, but pre-requisite courses of pre-requisites and so on.)

e) Find all pre-requisites of a course (allprereq). (This will involve finding not only the immediate prerequisites of a course, but pre-requisite courses of pre-requisites and so on.)

f) Find all teachers the students of a course are currently taking (student_teach).

When you are through testing your queries interactively, prepare them for use by the testing program by defining predicates with the names shown above.

Part 2: Boo-ray (or Bourrée)

Boo-ray (or Bourrée) is a trick based card game. Rules vary from location to location, but for the purposes of this part the following rules will apply:

1. There are two players each with five cards.
2. There is a trump card (this will specify the trump suit unless it is a 2 – in the case of a 2 there is no trump suit).
3. The player leading the round can play any card but a trump suit unless a trump card has been played, or the player only has trump suit left in hand.
4. The player countering in the round must play a card of the suit that was led, unless the player does not have one in hand (in that case the player can play any card in hand).
5. If the suits are the same for the cards in the round, the player that played the highest card wins (A or 1 is high, then K or 13 is next and so on). If the suits are different and no trump suit is played, then the leading player wins the round. If only one trump suit is played, the player that played it wins the round.
6. The player that wins the previous round leads for the current round.
7. Player wins if they take three or more tricks (win three or more rounds).

Write a predicate that is true if it is possible for the initial leader can possibly win. Provide the order of the cards that were played and who was the lead each round. The first parameter should be the trump card, the second should be leader hand, then counter hand, and return the play. The cards will be a list of two values (the card number 1 – 13 for A, 2, …, J, Q, and K and the suit c – clubs, d – diamonds, h – hearts, and s – spades). The following is an example and the potential successful play:

```
bourree([3,h],[[5,c],[8,c],[1,d],[4,h],[12,s]],[[6,c],[11,c],[12,d],[1,s],[2,s]],X).
X = [[t,[5,c],[6,c]],[f,[8,c],[11,c]],[f,[12,s],[2,s]],[t,[1,d],[12,d]],[t,[4,h],[1,s]],[3,2]]

yes
```

Part 3: Map Coloring

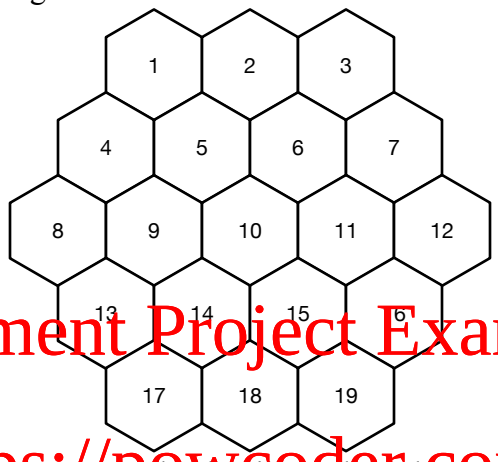
Write a logic program to solve map coloring of map. Part of a Europe map is provided in the europe.p file. The map is described as adjacency pairs, with the predicate is_adjacent(a,b) if a and b are adjacent. Your predicate to color the map should be called mapcolor and should build a

sorted list where each element is a two-item list of the location name followed by the color. The colors that are allowed are red, white, blue, and gold. An example of the return for mapcolor(X) should be something like:

```
X = [[countrya, red], [countryb, white], [countryc, blue]]
```

Part 4: Lo Shu Hexagon (Extra Credit)

The Lo Shu square requires that the numbers from 1 – 9 be put in a 3 x 3 square so that all rows/columns and diagonals add up to 15. For this problem you will be doing the Lo Shu hexagon, you will be placing the numbers from 1 – 19 in a hexagon so that every row, diagonal, etc. will add up to 38. The hexagon looks as follows:



Each cell has three possible components that will need to add up to 38. For example cell 1 would need to add up to 38 with cells 2 and 3, with 4 and 8, and with 5, 10, 15, and 19. Your solution will be compared against the professor's in how fast if finds solutions (the first and all solutions).

You must submit the source file(s), and README.txt file in a tgz archive.

You should avoid using existing source code as a primer that is currently available on the Internet. You **must** specify in your readme file any sources of code that you have viewed to help you complete this project. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.