Assignment 5: Prolog Program
Due: Thursday, 3 December 2015 at 11:59pm

Overview

The purpose of this assignment is for you to gain some experience designing and implementing Prolog programs. This assignment is broken into three parts. The first part is a fairly straightforward Prolog warm-up. Part 2 involves writing a set of simple programs for manipulating lists and variables. Part 3 requires you to use Prolog's control mechanism to implement a predicate for scheduling.

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, courses_taken)`

   Here `name` is an atom denoting a student's name, 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, [ecs100, ecs110], 3).
                  .
                  .
                  .
   student(john, [ecs30, ecs40]).
                  .
                  .
                  .
   instructor(jim, [ecs30, ecs110]).
   ```

   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 ecs110 (`prereq_110`).
   (c) Find names of all students in ecs140a (`ecs140a_students`).
   (d) Find the names of all instructors who teach john's courses (`instructor_names`).

(e) Find the names of all students who are in jim's class (`students`).

(f) 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.)

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: List Manipulation Predicates

The goal of this part of the homework is to familiarize you with the notions of lists and predicate definitions in Prolog. This part requires you to define a number of simple predicates:

1. Write a predicate, `all_length`, that takes a list and counts the number of atoms that occur in the list at all levels.

```
?-all_length([a, b, c], X)
X = 3;
no
?-all_length([a, [b,c], [d,[e,f]]], Y)
Y = 6;
no
```

2. Define a predicate, `equal_a_b(L)`, which returns true if `L` contains an equal number of `a` and `b` terms.

3. Define a predicate, `swap_prefix_suffix(K,L,S)`, such that `swap_prefix_suffix(K,L,S)` is true if

- `K` is a sub-list of `L`, and
- `S` is the list obtained by appending the suffix of `L` following an occurrence of `K` in `L`, with `K` and with the prefix that precedes that same occurrence of `K` in `L`.

```
?-swap_prefix_suffix([c, d], [a, b, c, d, e], S).
yes. S=[e, c, d, a, b]
?-swap_prefix_suffix([c, e], [a, b, c, d, e], S).
no.
?-swap_prefix_suffix(K, [a, b, c, d, e], [b, c, d, e, a]).
yes. K=[a]
```

Look at the class notes for the definitions of prefix, suffix, and sublist.

4. Define a predicate, `palin(A)` that is true if the list `A` is a palindrome, that is, it reads the same backwards as forwards. For instance, `[1, 2, 3, 2, 1]` is a palindrome, but `[1, 2]` is not.

5. A good sequence consists either of the single number 0, or of the number 1 followed by two other good sequences: thus, `[1,0,1,0,0]` is a good sequence, but `[1,1,0,0]` is not. Define a relation `good(A)` that is true if `A` is a good sequence.

Part 3: Puzzle

Write a logic program to solve the following puzzle: A farmer must ferry a wolf, a goat, and a cabbage across a river using a boat that is too small to take more than one of the three across at once. If he leaves the wolf and the goat together, the wolf will eat the goat, and if he leaves the goat with the cabbage, the goat will eat the cabbage. How can he get all three across the river safely?

Hints: Define the following predicates:

- Use terms, `left` and `right`, to denote the two banks.

- Define a term `state(left,left,right,left)` to denote the state in which the farmer, the wolf, and the cabbage are on the `left` bank, and the goat is alone on the `right` bank.

- Define a term, `opposite(A,B)`, that is true if if A and B are different banks of the river.

- Define a term, `unsafe(A)`, to indicate if state A is unsafe.

- Similarly define a term, `safe(A)`.

- Define a term, `take (X,A,B)`, for moving object X from bank A to bank B.

- Define a term, `arc (N,X,Y)`, that is true if move N takes state X to state Y. Define the rule for the above terms. Now, the solution involves searching from a certain initial state to a final state. You may look at relevant examples in the textbook on how you can write your search algorithms.

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(_).
  ```

- Read Chapter 4 of your textbook. It contains several examples.

- Read Sections 8.1-2 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 $[A \mid X]$, 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 filename 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 is also provided. It exercises the predicates that you write; hence there is no test data. Details regarding the test program can be found in the attached materials for this assignment.

  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.

- Grading will be divided as follows:

Part	Percentage
1	30
2	45
3	25

- **Get started now to avoid the last minute rush.**