# ECS 140A Programming Languages

WINTER 2016

Homework #5

Due 11:59pm Thursday March 10th, 2016

## 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.

## Instructions

- Read Chapter 4 of your main textbook. It contains several examples.
- Read Sections 8.1-2 in the Prolog textbook 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!).
- The command to use is `gprolog`. The manual for gprolog is available at: `http://www.gprolog.org/manual/gprolog.html`. Check if you are working on the latest version of gprolog-1.4.4.
- Before executing consult all the files that are required for its execution. To consult file `abc.pl` use the command `consult('abc.pl')`.
- The test program can be executed by calling `./test.sh`. **Don't worry about the diff if your output prints in an order different than that's given in Output.correct but make sure the content is same though.**

- Individual test case can be executed as `test_isMember.`. Notice that the period is part of the command.
- You can either use \+ for not or, define a mynot function as follows:

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

- When developing your program, you might find it easier to first test your predicate interactively before using the test program. You might find trace predicate useful in debugging your predicate.

## Part 1: Simple Queries

**The order of the output doesn't matter refer to Output.correct just for the correctness of your output results.** You are given a set of facts of the following form:

1. `novel(name, year).`
   Here `name` is an atom denoting the name of the novel and `year` denotes the year that the novel has been published.
   For example, the fact `novel(the_kingkiller_chronicles, 2007).` says that the novel named `the_kingkiller_chronicles` was published in the year `2007`.
2. `fan(name, novels_liked).`
   Here `name` is an atom denoting the name of the character (in some imaginary world!) and `novels_liked` denotes the list of novels liked by that character.
   For example, the fact `fan(joey, [little_women]).` says that the character named `joey` is a fan of the novel named `little_women`.
3. `author(name, novels_written).`
   Here `name` is an atom denoting the name of the author and `novels_written` denotes the list of novels written by that author.
   For example, the fact `author(george_rr_martin, [a_song_of_ice_and_fire_series]).` says that the author named `george_rr_martin` has written the novel named `a_song_of_ice_and_fire_series`.

Write the following separate queries:

1. Find all the names of the novels that have been published in either 1953 or 1996.
2. Find all the names of the novels that have been published during the period 1800 to 1900.
3. Find all the names of the characters who are fans of `the_lord_of_the_rings`.
4. Find all the names of the authors whose novels `chandler` is a fan of.
5. Find all the names of the characters who are fans of the novels authored by `brandon_sanderson`.

6. Find all the names of the novels that are common between either of `phoebe, ross and monica`.

## 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.

In the following exercises you should implement sets as lists, where each element of a set appears exactly once on its list, but in no particular order. Do not assume you can sort the lists. Do assume that input lists have no duplicate elements, and do guarantee that output lists have no duplicate elements. **Choose any order for your predicate but to check the correctness of your output make sure that the elements in the list are same as the one provided in Output.correct.**

1. Define the `isMember` predicate so that `isMember(X,Y)` says that element `X` is a member of set `Y`. Do not use the predefined list predicates.
   For eg., `isMember(a,[b]).` returns `no`

2. Define the `isUnion` predicate so that `isUnion(X,Y,Z)` says that the union of `X` and `Y` is `Z`. Do not use the predefined list predicates. Your predicate may choose a fixed order for `Z`. If you query `isUnion([1,2],[3],Z)` it should find a binding for `Z`, but it need not succeed on both `isUnion([1],[2],[1,2])` and `isUnion([1],[2],[2,1])`. Your predicate need not work well when `X` or `Y` are unbound variables.
   For eg.,`isUnion([1,2],[3],Z).` returns `Z = [1,2,3]`

3. Define the `isIntersection` predicate so that `isIntersection(X,Y,Z)` says that the intersection of `X` and `Y` is `Z`. Do not use the predefined list predicates. Your predicate may choose a fixed order for `Z`. Your predicate need not work well when `X` or `Y` are unbound variables.
   For eg.,`isIntersection([1,2],[3],Z).` returns `Z = []`

4. Define the `isEqual` predicate so that `isEqual(X,Y)` says that the sets `X` and `Y` are equal. Two sets are equal if they have exactly the same elements, regardless of the order in which those elements are represented in the set. Your predicate need not work well when `X` or `Y` are unbound variables.
   For eg.,`isEqual([a,b],[b,a]).` returns `yes`

5. Define the `powerSet` predicate so that `powerSet(X,Y)` says that the powerset of `X` is `Y`. The powerset of a set is the set of all subsets of that set. For example, consider the set `A`=$\{$`1,2,3`$\}$. It has various subsets: $\{$`1`$\}$,$\{$`1,2`$\}$ and so on. And of course the empty set $\emptyset$ is a subset of every set. The powerset of A is the set of all subsets of A:
   $P(S) = \{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{2,3\},\{1,3\},\{1,2,3\}\}$
   For your `powerSet` predicate, if `X` is a list(representing the set), `Y` will be a list of lists(representing the set of all subsets of the original set). So `powerset([1,2],Y)` should produce the binding `Y =` $\{\{$`1,2`$\},\{$`1`$\},\{$`2`$\},\{\}\}$(in any order).Your predicate

may choose a fixed order for `Y`. Your predicate need not work well when `X` is unbound variable.

## 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

- You must develop your program in parts as outlined above. Grading will be divided as follows:

| Part # | Percentage |
|---|---|
| 1 | 30 |
| 2 | 40 |
| 3 | 30 |

- If your program does not fully work, hand in a listing of the last working part along with your attempt at the next part, and indicate clearly which is which. No credit will be given if the last working part is not turned in. Points will be deducted for not following instructions, such as the above.
- **Get started now to avoid the last minute rush.**