

ECS 140A Programming Languages

SPRING 2018

Homework #4

Due 11:59pm Tuesday May 22nd, 2018

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. The assignment files can be downloaded from either Kodethon or Canvas. To use prolog on Kodethon, change your environment to 'prolog' and select 'gnu-1.4.4' as the version. This assignment should be worked on individually. Please turn in your solutions electronically via Kodethon or Canvas by the due date.

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 comma, e.g., `a(H) :-b(H).C(H).`, is quite different from `a(H) :-b(H), C(H).`
 - 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).`
- The test program can be executed by calling `./test.sh`.

- 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 (30 points)

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 (40 points)

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.

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 example, `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 example, `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 example, `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 example, `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 an unbound variable.

Part 3: Puzzle (30 points)

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. Each of them might initially be on either the left or right bank of the river. The farmer is tasked with moving them to a given bank.

Define the following predicates:

- Use terms, `left` and `right`, to denote the two banks.
- Define a term `state(F, W, G, C)` that denotes on which bank the farmer (F), the wolf (W), the goat (G), and the cabbage (C) are on.
or example, `state(left, left, right, left)` denotes 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 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.

The task is to define the rule `solve(F1, W1, G1, C1, F2, W2, G2, C2)` that prints the moves that should be taken to go from the initial state of the farmer (F1), wolf (W1), goat (G1), and cabbage (C1) to their respective final state F2, W2, G2 and C2.