Fall 2017
Programming Languages
Homework 4

- This homework is a combination programming and "paper & pencil" assignment.
- Due via NYU Classes on Wednesday, December 6 at 5:00 PM Eastern Time. Due to timing considerations relating to the end of the semester, late submissions will not be accepted.
- For the Prolog questions, you should use SWI Prolog. A link is available on the course page. For the Prolog Adventure question, please see the attachment `adv.pl`. You will start with the code in this file and modify it as you answer the questions.
- You may collaborate and use any reference materials necessary to the extent required to understand the material. All solutions must be your own, except that you may rely upon and use Prolog code from the lecture if you wish. Homework submissions containing any answers or code copied from any other source, in part or in whole, will receive a zero score.
- Please submit your homework as a zip file, `hw4-<netid>.zip`. The zip file should contain:
  - For the Prolog Adventure question (Q4), `hw4-<netid>-advrules.pl` which should contain all *rules* necessary to make the solution work. That is, it should contain the contents of the code from `adv.pl`, plus your modifications. Use comments to mark your modifications.
  - For the Prolog Adventure question (Q4), `hw4-<netid>-advqueries.txt` containing the adventure game commands which the question asks for.
  - For the Prolog Rules question (Q2), `hw4-<netid>-prules.pl` which should contain your implementation of all of the rules. You do not need to submit queries, although we will run our queries on your solutions.
  - For Q3, Q5, and sub-questions 1-3 of Q1 (Investigating Prolog), a PDF file `hw4-sols.pdf` containing the answers to all of the "paper & pencil" questions.
- Make sure your Prolog code compiles before submitting. *If your code does not compile for any reason, it may not be graded.*

1. [10 points] **Investigating Prolog**

   Consider the following:

```
male(brian).
male(kevin).
male(zhane).
male(fred).
male(jake).
male(bob).
male(stephen).
male(tom).
male(paul).

parent(melissa,brian).
parent(mary,sarah).
parent(stephen,jennifer).
parent(bob,jane).
parent(paul,kevin).
parent(tom,mary).
parent(jake,bob).
parent(zhane,melissa).
parent(tom,stephen).
parent(stephen,paul).
parent(emily,bob).
parent(zhane,mary).

grandfather(X,Y) :- male(X), parent(X,Z),parent(Z,Y).
```

   1. Reorder the facts above to provide faster execution time when querying `grandfather(tom,jennifer).` List the re-ordered facts and briefly explain what you changed.
   2. Explain in your own words why the change above affects total execution time. Show evidence of the faster execution time (provide a trace for each).
   3. Can we define a new rule `grandmother` that calls into the goals presented above to correctly arrive at an answer? Why or why not? (See slide 32 of the Prolog lecture. This will also be covered in class on November 27.)
   4. Define new rules `aunt` and `uncle` that work with the rules above. If you need to define other rules (including facts) in order to correctly define these, go ahead. Assume that the universe of facts is **not** complete.

2. [25 points] **Prolog Rules**

Write the Prolog rules described below in a file `hw4-<netid>-prules.pl`. All rules must be written using the subset of the Prolog language discussed in class. You may not, for example, call built-in library rules unless specifically covered in the slides. You may call your own rules while formulating other rules. You do not need to turn in queries, although you should certainly test your rules using your own queries.

1. Write a rule `remove_item(I,L,O)` in which O is the output list obtained by removing every occurrence of an item I from list L. The items in O should appear in the same order as L, only without I. Optional hint: try first defining `remove_item/4` and then define `remove_item/3` in terms of that.
2. Write a rule `remove_items(I,L,O)` which is the same as `remove_item/3` above except that I is a *list* of items to be removed from L instead of just a single item.
3. Write a rule `intersection(L1,L2,F)` in which F is a set containing only items that appear in both L1 and L2. You should not assume that L1 or L2 are sets[1] and, therefore, may contain repeated items. Optional hint: first defining `intersection/4` and then define `intersection/3` in terms of that.
4. Write a rule `is_set(L)` which is true if L is a set.
5. Write a rule `disjunct_union(L1,L2,U)` in which U is the disjunctive union of the items in L1 and L2. That is, items in L1 or L2 that are not in the intersection of L1 and L2. You should not assume that L1 or L2 are sets.
6. Write a rule `remove_dups(L1,L2)` in which L2 is L1 with duplicates removed. The order of the output list does not matter. This can also be viewed as a rule that creates a set from a list.
7. Write a rule `union(L1,L2,U)` in which U is the union (i.e., a set) of the items in L1 and L2. Do not assume that L1 or L2 are sets.

[1] A *set* is a collection of unique, unordered items.

3. [10 points] **Unification**

For each pair below that unifies, show the bindings. Circle any pair that doesn't unify and explain why it doesn't.

1. d(15) & c(X)
2. 42 & 23
3. a(X, b(3, 1, Y)) & a(4, Y)
4. a(X, c(2, B, D)) & a(4, c(A, 7, C))
5. a(X, c(2, A, X)) & a(4, c(A, 7, C))
6. e(c(2, D)) & e(c(8, D))
7. X & e(f(6, 2), g(8, 1))
8. b(X, g(8, X)) & b(f(6, 2), g(8, f(6, 2)))
9. a(1, b(X, Y)) & a(Y, b(2, c(6, Z), 10))
10. d(c(1, 2, 1)) & d(c(X, Y, X))

4. [45 points] **Prolog Adventure**

In 1976, Will Crowther created a game called **Colossal Cave Adventure**, the first of the interactive fiction and text adventure genres. Text adventures required players to input text in order to perform actions. For example, typing LOOK allows a player examine the surrounding environment, and inputting INVENTORY lists items in the player character's inventory. **Colossal Cave Adventure** is available to play online.

It turns out that Prolog is an excellent tool for prototyping and implementing basic text adventure games since these games rely on logic. This assignment is based on the adventure game implemented at `http://www.amzi.com/AdventureInProlog/`, where the player controls a young girl who is exploring her house in search of her security blanket, nani. To run the game, type **main.**

We have implemented a basic prototype of this game for you in **adv.pl**. Your task is to examine that file and implement the following functionality in Prolog. That is, make use of the functions given to you in **adv.pl** and modify them as necessary to answer each question. These modifications should be placed in `hw4-<netid>-advrules.pl`. To make grading easier, please **clearly identify your changes** using comments.

1. Create a room called **attic**.
2. Create two items, **doll** and **plate**, in the attic.
3. Create an item called **teleporter** in the desk.
4. Right now it's possible to carry heavy items like a desk to another room. That's not very realistic for a 3 year old. Add logic to prevent heavy items from being taken to different rooms. You get to designate which items are or aren't heavy.
5. Write a procedure called **teleport** that allows the player to move themselves to any other room in the house if the player has the **teleporter** in their inventory. Objects can be teleported along with the girl if they are in her possession. Heavy objects cannot be teleported, since the girl cannot take them in the first place.
6. The action **turn off** is not currently implemented. Implement it.
7. If you take an item and then try to take it again, the program currently says the item doesn't exist. This isn't quite right. Instead make it say that you already have the item. Of course, the system should continue to say that the item doesn't exist if it actually doesn't.
8. Implement a new action called **toss** which will toss an item out the window, provided the item is **tossable**. Tossing an item out the window should result in the item being removed from inventory. Attempting to throw a non-tossable item should result in an appropriate error. Note that many steps are required for this, so study the existing program carefully. You get to decide what is and isn't tossable.
9. Implement one more feature of your own. Explain how it works.
10. Create a separate file called `hw4-<netid>-advqueries.txt` that consists of actions in the game that results in the girl having broccoli, an apple, a key, the doll, and crackers in her inventory, and having the girl located at the cellar. You are forbidden from writing a procedure that allows the girl to take arbitrary items from arbitrary locations; for example, the player must be at the cellar to access the washing machine.

5. [10 points] **Prototype OOLs**

The following question should be completed after the November 27 lecture. Consider the following code below, written in a prototype OOL:

```
var obj1;
obj1.x = 20;
var obj2 = clone(obj1);
var obj3 = clone(obj2);
var obj4 = clone(obj1);
obj2.y = 5;
obj4.x = 10;
obj3.z = 30;
```

Assume that the program fragment above has executed. Answer the following:

1. What fields are contained locally to `obj1`?
2. What fields are contained locally to `obj2`?
3. What fields are contained locally to `obj3`?
4. What fields are contained locally to `obj4`?
5. To what value would `obj1.x` evaluate, if any?
6. To what value would `obj2.x` evaluate, if any?
7. To what value would `obj3.x` evaluate, if any?
8. To what value would `obj4.x` evaluate, if any?
9. To what value would `obj4.y` evaluate, if any?
10. To what value would `obj2.y` evaluate, if any?
11. To what value would `obj3.y` evaluate, if any?
12. To what value would `obj3.z` evaluate, if any?