CSE505 – Spring 2018

Assignment 4 – Prolog

Due Date: **Weds, April 19 and 26 (11:59 pm)**

You may work in pairs for this assignment.

Problem 1 (due April 19): Four boys (Don, Ding, Daod, Deep) and four girls (Kate, Kari, Kim, Kejal) must pair up as one boy and one girl and each pair must program in one of four languages (Pascal, Perl, Python, Prolog). No two pairs will program in the same language. Furthermore, we must respect their individual likes and dislikes given by the following constraints:

1. Kejal programs only in Python but Kate does not program in Python.
2. Don programs only in Pascal and Daod does not program in Prolog.
3. Neither Kim nor Kari nor Deep programs in Perl.
4. Ding cannot be with Kim, Kari, or Kejal.
5. Kari cannot be with Don, and Kate cannot be with Don or Daod.

Write a Prolog program to determine the members of each team and their programming language. The outline of your solution is given below; your task is to define one Prolog rule for each of the five constraints given above. The answer to the problem is a Prolog list of four triples of the form `team(Boy,Girl,Language)`.

```
solve(Ans):-
      Ans = [team(_,_, pascal), team(_,_, perl),
             team(_,_, python), team(_,_, prolog)],
      constraint1(Ans),
      constraint2(Ans),
      constraint3(Ans),
      constraint4(Ans),
      constraint5(Ans).
```

For example, the definition of `constraint1` would be:

```
constraint1(Ans) :-
      member(team(_, kejal, python), Ans),
      member(team(_, kate, PL), Ans),  PL \== python.
```

Starter code is posted on Piazza in a file called `teams.pl`. Extend this file in developing your solution.

Problem 2 (due April 26): The function calls appearing in a program can be represented by a Prolog relation `calls(F,L)`, which states that function `F` makes a direct call to each function listed in `L`. For example, we might have the following facts.

```
calls(f, [g]).
calls(g, [h,k]).
calls(h, [f]).
calls(k, [m]).
```

Here, function `g` makes a direct call to `h` and `k`. Indirectly, all calls emanating from `g` are `h`, `k`, `f`, `g`, and `m`. The functions `f`, `g`, and `h` are all recursive, but `k` is not. Function `m` is called but is undefined because it does not appear in the first argument of any fact.
Assuming that a database of `calls` facts are given, define the following Prolog predicates:

a. `callers(F,L)`: Given function `F` as input, return in `L` the list of all functions that make a direct call to `F`.
b. `undefined(L)`: Return in `L` the list of all functions that are called but have no definition in the database.
c. `recursive(F)`: Given a function `F` as input, return `true`/`false` indicating whether `F` is recursive.
d. `all_calls(F,L)`: Given function `F` as input, return in `L` the list of all functions that `F` calls directly or indirectly.

The database of calls and starter code is posted on Piazza in a file called `analyzer.pl`. Extend this file in developing your solution.

Problem 3 (due April 26): Making use of the above predicates, implement a Call Graph Analyzer for Tiny PL programs by making some extensions to the Tiny PL parser which will be given to you. Details to be communicated in a few days. The Tiny PL program will be in a file called `defs.txt` and the parser in a file called `tinypl.pl`.

WHAT TO SUBMIT:

Problem 1 (submit by April 19): Make a directory called `A4_Prob1_UBITId` if working solo or make a directory called `A4_Prob1_UBITId1_UBITId2` if working as a pair (give UBITId's in alphabetic order). Put `teams.pl` in this directory, compress the directory, and submit it using the `submit_cse505` command.

Problems 2-3 (submit by April 26): Make a directory called `A4_Prob23_UBITId` if working solo or make a directory called `A4_Prob23_UBITId1_UBITId2` if working as a pair (give UBITId's in alphabetic order). Put `defs.txt`, `analyzer.pl`, and `tinypl.pl` in this directory, compress the directory, and submit it using the `submit_cse505` command.

End of Assignment 4