EECS 3401 ASSIGNMENT 2 SOLUTIONS

Deadline: March. 7, 2016
Student: Skyler Layne (cse23170, 212166906)

PROBLEM 1

(1) X is a grandfather of Z, if X is the father of Y and Y is a parent of Z.

(2) The father of Y is a parent of Y.

(3) The mother of Y is a parent of Y.

Given the following facts:

(4) adam is the father of beth and bill.

(5) beth is the mother of chris.

(6) bill is the father of ann.

A. Convert (1)-(6) into clauses.

Use the following predicates:

gf(X,Y) - to denote the fact that X is a grandfather of Y;
f(X,Y) - to denote the fact that X is the father of Y;
m(X,Y) - to denote the fact that X is the mother of Y;
p(X,Y) - to denote the fact that X is a parent of Y;

A.sol

(1) C1. `gf(X,Y) -> (f(X,Y) & p(Y,Z)).`
`gf(X, Z) :- f(X, Y), p(Y, Z).`

(2) C2. `f(X, Y) :- p(X, Y).`

(3) C3. `m(X, Y) :- p(X, Y).`

(4) C4.1. `f(adam, beth).`
C4.2. `f(adam, bill).`

(5) C5. `m(beth, chris).`

(6) C6. `f(bill, ann).`

B. Formulate an appropriate query to solve the problem

B.sol: find all A's such that adam is the grandparent

Q. `gf(adam, X).`

C. Construct a complete SLD search tree

C.sol:

```
              :- gf(adam, X).
                   *
                   | C1.
```

```
                    *
          gf(X, Z) :- f(X, Y), p(Y, Z).
          :- f(adam, Y'), p(Y', X').
                    *
                    |
        **************************
       / C2, C4.1.                 \ C2, C4.2.
      *                             *
f(X, Y) :- p(X, Y).           f(X, Y) :- p(X, Y).
f(adam, beth).                f(adam, bill).
p(beth, X).                   f(bill, X).
      *                             *
      |  C5.                        | C6.
      *                             *
m(X, Y) :- p(X, Y).           f(X, Y) :- p(X, Y).
m(beth, chris).               f(bill, ann).
X = chris.                    X = ann.
```

Prolog Code for the Grandfather Problem

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Assignment 2: Problem 1 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Representing Grandfather in Prolog %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

p(adam, beth).
p(adam, bill).
p(beth, chris).
p(bill, ann).

% Definition of a Grandfather
gf(X, Z) :- f(X, Y), p(Y, Z).

% Definition of a parent
f(X, Y) :- p(X, Y).

% Definition of a mother
m(X, Y) :- p(X, Y).
```

PROBLEM 2

Specification of merge(L1,L2,L)

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%% Merge a sorted list %%
%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%
% CASE 1: If the empty lists.
```

```
%%%%
merge([], [], []).

%%%%
% CASE 2: The lists contain 1 element
%%%%
merge([], [Y], [Y|L]) :- merge([], [], L).
merge([X], [], [X|L]) :- merge([], [], L).

%%%%
% CASE 3: The first element, X, in L1 is  less than the first element, Y, in L2, add X to L
    and merge on L1', L2, and L' where L1' is `L1 \ X`, L2 stays the same and L' is `L u X`
%%%%
merge([X|L1], [Y|L2], [X|L]) :- lt(X, Y), merge(L1, [Y|L2], L).

%%%%
% CASE 4: The first element, X, in L1 is greater than or equal to the first element, Y, in
    L2, add Y to L and merge on L1, L2', and L' where L1 stays the same, L2' is `L2 \ Y` and
    L' is `L u Y`
%%%%
merge([X|L1], [Y|L2], [Y|L]) :- merge([X|L1], L2, L).

%%%%%%%%%%%%%%%%%%%%%%%
%% Helper Predicates %%
%%%%%%%%%%%%%%%%%%%%%%%
% True if X < Y is true.
lt(X, Y) :- X<Y.
```

Specification of delete(X,L,L1)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Delete all occurrences from a list %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CASE 1: Empty
delete(_, [], []).

% CASE 2: List of a list
delete(X, [[_|L1]|L2], [L1|L]) :- delete(X, L2, L).

% CASE 3: Recursive Case
delete(X, [X|L1], L) :- delete(X, L1, L).
delete(X, [H|L1], [H|L]) :- delete(X, L1, L).
```

PROBLEM 3

Specification of Binary Tree merge(T, T1, T2)

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Merge a two binary search trees %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% merge(T, T1, T2) true when T2 is the result of inserting every node from T1 into T. %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CASE 1: Empty, nothing to do
merge(nil, nil, nil).
merge(t(L, Root, R), nil, t(L, Root, R)).
merge(nil, t(L, Root, R), t(L, Root, R)).

% CASE 2: If the first element in T is less than the root of T1,
merge(t(L, X, R), t(L1, X1, R1), t(L2, X, R2)) :- X =< X1, merge(L, L1, L2), merge(R, t(nil,
    X1, R1), R2).

merge(t(L, X, R), t(L1, X1, R1), t(L2, X1, R2)) :- X > X1, merge(L, L1, L2), merge(t(nil, X,
    R), R1, R2).

%%%%%%%%%%%
%% Tests %%
%%%%%%%%%%%
% Test 1:  either tree contains only a single node.
merge(t(nil, 5, nil), t(nil, nil, nil), t(nil, 5, nil)).
merge(t(nil, nil, nil), t(nil, 5, nil), t(nil, 5, nil)).

% Test 2: Right trees
merge(t(nil, 4, nil), t(nil, 5, nil), t(nil, 4, t(nil, 5, nil))).
merge(t(nil, 4, t(nil, 6, nil)), t(nil, 5, nil), t(nil, 4, t(nil, 5, t(nil, 6, nil)))).

% Test 3: Even Trees
merge(t(t(nil, 2, nil), 4, nil), t(nil, 3, nil), t(t(nil, 2, nil), 3, t(nil, 4, nil))).
merge(t(t(nil, 3, nil), 4, nil), t(nil, 2, nil), t(t(nil, 3, nil), 2, t(nil, 4, nil))).

% Test 5: Large Tree
merge(t(t(nil, 4, t(nil, 5, nil)), 6, nil), t(t(t(nil, 5, nil), 6, t(nil, 7, nil)), 8, t(nil,
    9, nil)), t(t(t(nil, 5, nil), 4, t(nil, 5, t(nil, 6, t(nil, 7, nil)))), 6, t(nil, 8,
    t(nil, 9, nil)))).

% Test 6: Very Large Tree (Even 3 levels)
merge(t(t(t(t(nil, 4, nil), 5, t(nil, 6, nil)), 10, t(t(nil, 14, nil), 15, t(nil, 16, nil))),
    20, t(t(t(nil, 22, nil), 25, t(nil, 26, nil)), 30, t(t(nil, 34, nil), 35, t(nil, 36,
    nil)))), t(t(t(t(nil, 2, nil), 3, t(nil, 4, nil)), 5, t(t(nil, 7, nil), 8, t(nil, 9,
    nil))), 10, t(t(t(nil, 11, nil), 12, t(nil, 13, nil)), 15, t(t(nil, 16, nil), 18, t(nil,
    20, nil)))), t(t(t(t(nil, 2, t(nil, 4, nil)), 3, t(nil, 4, t(nil, 5, t(nil, 6, nil)))), 5,
    t(t(nil, 7, nil), 8, t(nil, 9, t(nil, 10, t(t(nil, 14, nil), 15, t(nil, 16, nil)))))), 10,
    t(t(t(nil, 11, nil), 12, t(nil, 13, nil)), 15, t(t(nil, 16, nil), 18, t(nil, 20,
    t(t(t(nil, 22, nil), 25, t(nil, 26, nil)), 20, t(nil, 30, t(t(nil, 34, nil), 35, t(nil,
    36, nil))))))))).
```

PROBLEM 4

Specification of resolve(C1,C2,P,Res)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% (a) resolve(C1,C2,P,Res) %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%resolve(C1,C2,P,Res).

% CASE 1: Base Case
resolve([],[],_,[]).

% CASE 2: If -P is in C1, AND P is in C2, remove -P from C1, P from C2 and append the left
    overs.
resolve(C1, C2, P, Res) :- not(member(P, C1)), Q is (0-P), member(Q, C1), r(Q, C1, R1), r(P,
    C2, R2), append(R1, R2, Res).

% CASE 3: If P is in C1, AND -P is in C2, remove P from C1, -P from C2 and append the left
    overs.
resolve(C1, C2, P, Res) :- member(P, C1), r(P, C1, R1), Q is (0-P), r(Q, C2, R2), append(R1,
    R2, Res).

%% Remove the first occurrence helper predicates.
r(X, [], []).
r(X, [X|T], T).
r(X, [H|T], [H|Res]) :- r(X, T, Res).

%%%%%%%%%%%
%% Tests %%
%%%%%%%%%%%

% resolve(C1,C2,P,Res) Tests
resolve([1], [-1], 1, []).
resolve([2,-1], [1,2], 1, [2, 2]).
resolve([-1,7,4], [2,-1,9,5,1,-1], 1, [7, 4, 2, -1, 9, 5, -1]).
```

Specification of sub(C1,C2)

```
%%%%%%%%%%%%%%%%%%%%
%% (b) sub(C1,C2) %%
%%%%%%%%%%%%%%%%%%%%
% CASE 1: Any clause subsumes the empty clause.
sub(C1, []).

% CASE 2: We must achieve all elements of C2 must be in C1, as we're modeling our clauses as
    lists.
sub(C1, [X|C2]) :- member(X, C1).

%%%%%%%%%%%
%% Tests %%
%%%%%%%%%%%

% sub(C1,C2) Tests
sub([],[]). % True
sub([1,2,3,4],[2,3]). % True
sub([3],[2,3]). % False
```

Specification of taut(C)

```
%%%%%%%%%%%%%%%%%
```

```
%% (c) taut(C) %%
%%%%%%%%%%%%%%%%%

% CASE 1: Check if the opposite literal exists in the Clause.
taut([H|T]) :- Q is (0-H), member(Q, T).

%%%%%%%%%%%
%% Tests %%
%%%%%%%%%%%

% taut(C) Tests
taut([1,2]).
taut([-1,2]).
taut([1,2,-1]).
taut([1,2,-1,-2]).
```