Alexander Grzesiak

April 12, 2017

CS 470: Intelligent Systems

FOPL in Prolog

* 1. **Peano Problems**
     1. natNum
        1. FOL
           1. NatNum(0).
           2. ∀n natNum(n)⇒NatNum(S(n)).
        2. 0 is a natural number and for all n, n is a natural number if n is the successor of a number that is also a natural number.
     2. successor
        1. FOL
           1. ∀n successor(n) ≠ 0.
           2. ∀m, n m ≠ n ⇒ S(m) ≠ S(n).
           3. ∀m, n successor(m, n) ⇔ m + 1 = n
        2. Explanations
           1. For all n, the value whose successor is n does not equal 0.
           2. For all m and n, m does not equal is the successor of m does not equal the successor of n.
           3. For all m and n, m’s successor is n if and only if m plus one is equal to n.
        3. This was a little weird because only the constraints for the successor function were outlined in the book. The assignment details said that the FOL statements were in the book so it took a second to determine that one more had to be added to give the successor function a logic statement that said what it did.
     3. plus
        1. FOL
           1. ∀m natNum(m) ⇒ +(0, m) = 0.
           2. ∀m, n natNum(m) ∧ natNum(n) ⇔ +(S(m), n) = S(+(m, n)).
        2. Explanations
           1. For all m, if m is a natural number it implies that 0 plus m is equal to 0.
           2. For all m and n, if m is a natural number and n is a natural number it implies that m’s successor plus n is equal to m plus n’s successor.
     4. multiplication
        1. FOL
           1. ∀m natNum(m) ⇒ \*(m, 0) = 0
           2. ∀m, n natNum(m) ∧ natNum(n) ⇔ \*(S(m), n) = +(\*(m, n), m).
        2. Explanations
           1. For all m, if m is a natural number it implies that m multiplied by 0 equals 0.
           2. For all m and n, if m is a natural number and n is a natural number it implies that m’s successor multiplied by n equals m plus m multiplied by n (observing order of operations).
     5. exponentiation
        1. FOL
           1. ∀m natNum(m) ⇒ ^(m, 0) = 1
           2. ∀m, n natNum(m) ∧ natNum(n) ⇒ ^(m, S(n)) = \*(^(m, n), m).
        2. Explanations
           1. For all m, if m is a natural number it implies that m raised to the 0 power equals 1.
           2. For all m and n, if m is a natural number and n is a natural number it implies that m raised to n’s successors power equals m raised to the n power multiplied by m (observing order of operations).
  2. **Kinship Problems**
     1. grandchild
        1. ∀(x, y) grandchild(x,y) ⇔ ∃ p ∋ child(x, p) ∧ child(p, y)
        2. For all x and y, x is a grandchild of y if and only if there exists a p such that x is a child of p and p is a child of y.
     2. greatGrandParent
        1. ∀(x, y) greatGrandParent(x, y) ⇔ ∃ g, p ∋ child(g, x) ∧ child(p, g) ∧ child(y, p)
        2. For all x and y, x is a greatGrandParent of y if and only if there exists g and p such that g is a child of x, p is a child of g, and y is a child of p.
     3. ancestor
        1. ∀(x, y) ancestor(x,y) ⇔ ∃ c ∋ (child(y, x)) ∨ (child(c, x) ∧ ancestor(c, y))
        2. For all x and y, x is an ancestor of y if and only if there exists a c such that either y is a child of x, or c is a child of x and c is an ancestor of y.
        3. This one was somewhat difficult because of the recursion but there are only the conditions where the ancestor is the parent as well as the condition in which you have to traverse down the ‘tree’ to find all possible ancestors.
     4. brother
        1. ∀(x, y) brother(x, y) ⇔ ∃ z ∋ male(x) ∧ child(x, z) ∧ child(y, z)
        2. For all x and y, x is a brother of y if and only if there exists a z such that x is a male, x is a child of z, and y is a child of z.
     5. sister
        1. ∀(x, y) sister(x, y) ⇔ ∃ z ∋ female(x) ∧ child(x, z) ∧ child(y, z)
        2. For all x and y, x is a sister of y if and only if there exists a z such that x is a female, x is a child of z, and y is a child of z.
     6. daughter
        1. ∀(x, y) daughter(x, y) ⇔ female(x) ∧ child(x, y)
        2. For all x and y, x is a daughter of y if and only if x is a female and x is a child of y
     7. son
        1. ∀(x, y) son(x, y) ⇔ male(x) ∧ child(x, y)
        2. For all x and y, x is a son of y if and only if x is a male and x is a child of y
     8. sibling
        1. ∀(x, y) sibling(x, y) ⇔ ∃ z ∋ child(x, z) ∧ child(y, z)
        2. For all x and y, x is a sibling of y if and only if there exists a z such that x is a child of z and y is a child of z.
        3. In a real implementation, there would have to be checks to ensure x ≠ y as well as only one parent is queried for kids as to avoid duplicates.
     9. firstCousin
        1. ∀(x, y) firstCousin(x, y) ⇔ ∃ p1, p2, g ∋ child(x, p1) ∧ child(y, p2) ∧ child(p1, g) ∧ child(p2, g)
        2. For all x and y, x is a first cousin of y if and only if there exists a p1, p2, and g such that x is a child of p1, y is a child of p2, p1 is a child of g, and 02 is a child of g.
     10. brotherinlaw
         1. ∀(x, y) brotherinlaw(x, y) ⇔ ∃ s, p ∋ male(x) ∧ (spouse(s, x) ∧ child(s, p) ∧ child(y, p)) ∨   
            (child(x, p) ∧ child(s, p) ∧ spouse(s, y))
         2. For all x and y, x is a brother in law of y if and only if there exists an s and p such that x is male and either s is a spouse of x, s is a child of p, and y is a child of p, or x is a child of p, s is a child of p, and s is a spouse of y
         3. This one was tricky because it must consider if person1 is married into the family or if person2 is married into the family.
     11. sisterinlaw
         1. ∀(x, y) sisterinlaw(x, y) ⇔ ∃ s, p ∋ female(x) ∧ (spouse(s, x) ∧ child(s, p) ∧ child(y, p)) ∨   
            (child(x, p) ∧ child(s, p) ∧ spouse(s, y))
         2. For all x and y, x is a sister in law of y if and only if there exists an s and p such that x is female and either s is a spouse of x, s is a child of p, and y is a child of p, or x is a child of p, s is a child of p, and s is a spouse of y
         3. This one was tricky because it must consider if person1 is married into the family or if person2 is married into the family.
     12. aunt
         1. ∀(x, y) aunt(x, y) ⇔ ∃ p, c ∋ female(x) ∧ (child(x, p) ∧ child(c, p) ∧ child(y, c)) ∨ (spouse(x, s) ∧ child(s, p) ∧ child(c, p) ∧ child(y, c))
         2. For all x and y, x is an aunt of y if and only if there exists a p and c such that x is a female and either x is a child of p, c is a child of p, and y is a child of c, or x is a spouse of s, x is a child of p, c is a child of p, and y is a child of c.
         3. The tricky part for aunt is that the aunt can either be an immediate family member or married to an immediate family member but in either case the woman is an aunt.
     13. uncle
         1. ∀(x, y) uncle(x, y) ⇔ ∃ p, c ∋ male(x) ∧ (child(x, p) ∧ child(c, p) ∧ child(y, c)) ∨ (spouse(x, s) ∧ child(s, p) ∧ child(c, p) ∧ child(y, c))
         2. For all x and y, x is an uncle of y if and only if there exists a p and c such that x is a male and either x is a child of p, c is a child of p, and y is a child of c, or x is a spouse of s, x is a child of p, c is a child of p, and y is a child of c.
         3. The tricky part for uncle is that the uncle can either be an immediate family member or married to an immediate family member but in either case the man is an uncle.

1. Explanation of ancestor(Who, eugenie)
   1. Once this query is entered into prolog, the interpreter recognizes that Who is obviously the thing we want to solve for meaning that we want to know who is the ancestor of eugenie. So, since the first definition of ancestor is the base case that just checks if the ancestor is the parent of eugine, we will hit this base case and we will find that one of eugenie’s parents will get bound to Who…our first result. Next, if we want more results, the base case will get hit again and since eugenie has one more parent, the other parent will get bound to Who…our second result. Now with the two trivial results out of the way we can get into the recursive definition of ancestor to find the other ancestors up the ‘tree’. Now, we start working our way up the tree starting at the dependent, eugenie. We start by taking a parent of eugenie; we will call this parent p1. Now that we have a value for p1, we want to find who is the ancestor of this newly found p1. So we ‘call’ ancestor with p1 as the dependent (i.e. ancestor(Who, p1)). This new query will hit the base case and just like before when eugenie was our dependent this new value, we’ll call it g1, will be bound to Who and ‘bubbled’ up through the recursive layers…our third. Just like when eugenie was our dependent, p1 has two ancestors with one of these being g1 and the second, g2, being found in quite the same way as g1…our fourth result. Each successive call to ancestor will find the two parents of the dependent, get bound to Who, and get bubbled up to give us every ancestor of eugenie. In our case where only one side of the family tree is looked, there are only two results for each layer of the tree. If each member in the tree had their own tree, the results would increase 2n where n is the layer in which we are searching for ancestors.
2. Explanation of duplicate results
   1. If the query brother(charles, X) was fed into Prolog without any duplicate checking, there would be two values for each of charles’s brothers. This is because we check for brother-ness by checking to see if there is anyone with the same parent as charles. Since charles has two parents, his brother will be found as a child of both parents which will be returned as the same brother found twice, once for each parent. The way I got around this was only looking at children for one of the parents so that there were no duplicates of children. My implementation was to only look at the female parent but in the case where there is no female parent for some reason, my code would break and produce a result of no siblings at all let along any brothers. This is only prevalent in queries where there are multiple paths to get from input to result. In another case, such as first cousin, you check if x1’s parent’s parent is the same as x2’s parent’s parent. So in each of these cases where duplicates are found, the query has to be narrowed.

%%%%%%%%%%% PART 1 %%%%%%%%%%%%%%%%%%

%%% SUCCESSOR %%%  
% ∀n successor(n) ≠ 0.  
% ∀m, n m ≠ n ⇒ S(m) ≠ S(n).  
% ∀m, n successor(m, n) ⇔ m + 1 = n  
%Ns successor is Y

successor(N,Y) :-  
 var(N) ->  
 N is Y - 1;  
 Y is N + 1.

%%% NATURAL NUMBER %%%  
% NatNum(0).   
% ∀n natNum(n)⇒NatNum(S(n)).

natNum(0).  
natNum(0.0).  
natNum(X) :-  
 successor(Y, X),  
 Y >= 0,  
 natNum(Y).

%%% ADDITION %%%  
% ∀m natNum(m) ⇒ +(0, m) = 0.  
% ∀m, n natNum(m) ∧ natNum(n) ⇔ +(S(m), n) = S(+(m, n)).

plus(X, 0, X).  
plus(X, Y, OUT) :-  
 successor(X , XX),  
 successor(Yy, Y),  
 plus(XX, Yy, OUT).

%%% MULTIPLICATION %%%  
% ∀m natNum(m) ⇒ \*(m, 0) = 0  
% ∀m, n natNum(m) ∧ natNum(n) ⇔ \*(S(m), n) = +(\*(m, n), m).

mult(X, Y, OUT) :-  
 multHelper(X, X, Y, OUT).

multHelper(X,\_,1,X).  
multHelper(X, ConstantX, Y, OUT):-  
 plus(X, ConstantX, AddOut),  
 successor(Yy, Y),  
 multHelper(AddOut, ConstantX, Yy, OUT).

%%% EXPONENTIATION %%%  
% ∀m natNum(m) ⇒ ^(m, 0) = 1  
% ∀m, n natNum(m) ∧ natNum(n) ⇒ ^(m, S(n)) = \*(^(m, n), m).

exp(X, Y, OUT):-  
 expHelper(X, X, Y, OUT).

expHelper(X, \_, 1, X).  
expHelper(X, ConstantX, Y, OUT):-  
 mult(X, ConstantX, MultOut),  
 successor(Yy, Y),  
 expHelper(MultOut, ConstantX, Yy, OUT).

%%%%%%%%%%% PART 2 %%%%%%%%%%%%%%%%%%

%%%% GRANDCHILD %%%  
% FOPL: ∀(x, y) grandchild(x,y) ⇔ E p child(x, p) ∧ child(p, y)

grandchild(C, G) :-  
 child(C, P),  
 child(P, G).

%%% GREATGRANDPARENT %%%  
% FOPL: ∀(x, y) greatGrandParent(x, y) ⇔ E g, p child(g, x) ∧ child(p, g) ∧ child(y, p)

greatGrandParent(A, C) :-  
 child(G, A),  
 child(P, G),  
 child(C, P).

%%% ANCESTOR %%%  
% FOPL: ∀(x, y) ancestor(x,y) ⇔ E c (child(y, x)) ∨ (child(c, x) ∧ ancestor(c, y))

ancestor(A, X) :-  
 child(X, A).

ancestor(A, X) :-  
 child(X, C),  
 ancestor(A, C).

%%% BROTHER %%%  
% FOPL: ∀(x, y) brother(x, y) ⇔ E z male(x) ∧ child(x, z) ∧ child(y, z)

brother(X, Y):-  
 male(X),  
 child(X, Z),  
 child(Y, Z),  
 female(Z).

%%% SISTER %%%  
% FOPL: ∀(x, y) sister(x, y) ⇔ E z female(x) ∧ child(x, z) ∧ child(y, z)

sister(X, Y) :-  
 female(X),  
 child(X, Z),  
 child(Y, Z),  
 female(Z).

%%% DAUGHTER %%%  
% FOPL: ∀(x, y) daughter(x, y) ⇔ female(x) ∧ child(x, y)

daughter(X, Y) :-  
 female(X),  
 child(X, Y).

%%% SON %%%  
% FOPL: ∀(x, y) son(x, y) ⇔ male(x) ∧ child(x, y)

son(X, Y):-  
 male(X),  
 child(X, Y).

%%% SIBLING %%%  
% FOPL: ∀(x, y) sibling(x, y) ⇔ E z child(x, z) ∧ child(y, z)

sibling(X, Y):-  
 child(X, Z),  
 child(Y, Z),   
 Y \= X,  
 female(Z).

%% FIRSTCOUSIN %%%  
% FOPL: ∀(x, y) firstCousin(x, y) ⇔ E p1, p2, g child(x, p1) ∧ child(y, p2) ∧ child(p1, g) ∧ child(p2, g)

firstCousin(X, Y) :-  
 child(X, P1),  
 child(Y, P2),  
 P1 \= P2,  
 child(P1, G),  
 child(P2, G),  
 female(G),  
 X \= Y.

%%% BROTHERINLAW%%%  
% FOPL: ∀(x, y) brotherinlaw(x, y) ⇔ E s, p male(x) ∧ (spouse(s, x) ∧ child(s, p) ∧ child(y, p)) ∨ (child(x, p) ∧ child(s, p) ∧ spouse(s, y))

brotherinlaw(X, Y) :-  
 male(X),  
 (  
 (spouse(X, S) ; spouse(S, X)),  
 child(S, P),  
 female(P),  
 child(Y, P),  
 Y \= S  
 );(  
 child(X, P),  
 female(P),  
 child(S, P),  
 S \= X,  
 (spouse(S, Y) ; spouse(Y, S))  
 ).

%% SISTERINLAW %%%  
% FOPL: ∀(x, y) sisterinlaw(x, y) ⇔ E s, p female(x) ∧ (spouse(s, x) ∧ child(s, p) ∧ child(y, p)) ∨ (child(x, p) ∧ child(s, p) ∧ spouse(s, y))

sisterinlaw(X, Y) :-  
 female(X),  
 (  
 (spouse(X, S) ; spouse(S, X)),  
 child(S, P),  
 female(P),  
 child(Y, P),  
 Y \= S  
 );(  
 child(X, P),  
 female(P),  
 child(S, P),  
 S \= X,  
 (spouse(S, Y) ; spouse(Y, S))  
 ).

%%% AUNT %%%  
% FOPL ∀(x, y) aunt(x, y) ⇔ E p, c female(x) ∧ (child(x, p) ∧ child(c, p) ∧ child(y, c)) ∨ (spouse(x, s) ∧ child(s, p) ∧ child(c, p) ∧ child(y, c))

aunt(X, Y) :-  
 female(X),  
 (  
 child(X, P),  
 female(P),  
 child(C, P),  
 X \= C,  
 child(Y, C)  
 );(  
 (spouse(X, S);spouse(S, X)),  
 child(S, P),  
 female(P),  
 child(C, P),  
 S \= C,  
 child(Y, C)  
 ).

%% UNCLE %%%  
% FOPL ∀(x, y) uncle(x, y) ⇔ E p, c male(x) ∧ (child(x, p) ∧ child(c, p) ∧ child(y, c)) ∨ (spouse(x, s) ∧ child(s, p) ∧ child(c, p) ∧ child(y, c))

uncle(X, Y) :-  
 male(X),  
 (  
 child(X, P),  
 female(P),  
 child(C, P),  
 X \= C,  
 child(Y, C)  
 );(  
 (spouse(X, S);spouse(S, X)),  
 child(S, P),  
 female(P),  
 child(C, P),  
 S \= C,  
 child(Y, C)  
 ).