Logic and Functional Programming

Labworks 10

May 26, 2021

1 Answers to review questions

1. (a) The program consists of 2 facts and 2 rules.

   (b) Function symbols: `john`, `bill`, `ann`
   Predicate symbols: `father`, `mother`, `parent`
   Variables: `X`, `Y`

   (c) `father`, `mother`, `parent`

2. (a) $\underline{\mathtt{f(X,Y,Z) = f(a,Z,h(a))}} \Rightarrow \mathtt{X = a, Y = Z, } \underline{\mathtt{Z = h(a)}} \Rightarrow$
   $\mathtt{X = a, Y = h(a), Z = h(a)}$
   We obtained the most general unifier $\{\mathtt{X \to a, Y \to h(a), Z \to h(a}\}$.

   (b) $\mathtt{f(g(X), g(c), Y) = f(g(g(Y)), X, a} \Rightarrow \underline{\mathtt{g(X) = g(g(Y))}}\mathtt{, g(c) = X, Y = a} \Rightarrow$
   $\underline{\mathtt{X = g(Y)}}\mathtt{, g(c) = X, Y = a} \Rightarrow \mathtt{X = g(Y), g(c) = g(Y), } \underline{\mathtt{Y = a}} \Rightarrow$
   $\mathtt{X = g(a), } \underline{\mathtt{g(c) = g(a)}}\mathtt{, Y = a} \Rightarrow \mathtt{X = g(a), } \underline{\mathtt{c = a}}\mathtt{, Y = a} \Rightarrow$failure.
   The two terms are not unifiable.

   (c) $\underline{\mathtt{f(h(b), X, X, Y) = f(h(b), g(Y), g(g(Z)), g(a))}} \Rightarrow$
   $\mathtt{h(b) = h(b), } \underline{\mathtt{X = g(Y)}}\mathtt{, X = g(g(Z)), Y = g(a)} \Rightarrow$
   $\mathtt{h(b) = h(b), X = g(Y), } \underline{\mathtt{g(Y) = g(g(Z))}}\mathtt{, Y = g(a)} \Rightarrow$
   $\underline{\mathtt{h(b) = h(b)}}\mathtt{, X = g(Y), Y = g(Z), Y = g(a)} \Rightarrow$
   $\underline{\mathtt{b = b}}\mathtt{, X = g(Y), Y = g(Z), Y = g(a)} \Rightarrow \underline{\mathtt{X = g(Y)}}\mathtt{, Y = g(Z), Y = g(a)} \Rightarrow$
   $\mathtt{X = g(Y), } \underline{\mathtt{g(Y) = g(Z)}}\mathtt{, Y = g(a)} \Rightarrow \mathtt{X = g(Y), Y = Z, } \underline{\mathtt{Y = g(a)}} \Rightarrow$
   $\mathtt{X = g(g(a)), } \underline{\mathtt{g(a) = Z}}\mathtt{, Y = g(a)} \Rightarrow \mathtt{X = g(g(a)), Z = g(a), Y = g(a)}.$
   We obtained the most general unifier
   $\{\mathtt{X \to g(g(a)), Z \to g(a), Y \to g(a)}\}$.

3. First, we define `sublist(S,L)` to hold if `S` is a sublist of `L`. We note that `S` is sublist of `L` if there exist subslists `S1`, `S2` such that `L` is the result of appending the lists `S1`, `S`, `S2`, in this order. Thus, we can use the predefined predicate `append` to define `sublist`:

```
sublist(S,L) :- append(S1S,S2,L),append(S1,S,S1S).
```

Then

```
niceList(L) :-
  L=[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
  sublist([1,_,1,_,1],L),
  sublist([2,_,_,2,_,_,2],L),
  sublist([3,_,_,_,3,_,_,_,3],L),
  sublist([4,_,_,_,_,4,_,_,_,_,4],L),
  sublist([5,_,_,_,_,_,5,_,_,_,_,_,5],L),
  sublist([6,_,_,_,_,_,_,6,_,_,_,_,_,_,6],L),
  sublist([7,_,_,_,_,_,_,_,7,_,_,_,_,_,_,_,7],L),
  sublist([8,_,_,_,_,_,_,_,_,8,_,_,_,_,_,_,_,_,8],L),
  sublist([9,_,_,_,_,_,_,_,_,_,9,_,_,_,_,_,_,_,_,_,9],L).
```

2 Homework (Deadline: May 11, 2021)

1. (a)
```
union([],B,B).
union([H|T],B,C):-member(H,B),!,union(T,B,C).
union([H|T],B,[H|C]):-union(T,B,C).
```
   (b)
```
diff([],_,[]).
diff([H|T],B,C):-member(H,B),!,diff(T,B,C).
diff([H|T],B,[H|C]):-diff(T,B,C).
```
   (c)
```
included([],B).
included([H|T],B):-member(H,B),included(T,B).
```

2.
```
% sum(-A,-B,+S) binds S to the representation of
% the sum of A with B
sum(0,B,B).
sum(s(A),B,s(S)):-sum(A,B,S).

% prod(-A,-B,+P) binds P to the representation of
% the product of A with B
prod(0,_,0).
prod(s(A),B,P):-prod(A,B,P1),sum(A,P1,P).

% pow(-A,-B,+P) binds P to the representation of
% the power of A to B
prod(A,0,s(0)).
prod(A,s(B),P):-pow(A,B,P1),prod(A,P1,P).
```

3. (a)
```
gcd(A,0,A).
gcd(A,B,D):-A<B,!,gcd(B,A,D).
gcd(A,B,D):-
  R is A mod B,
  gcd(B,R,D).
```

(b)
```
% move right
nextPair((X,Y),(X1,Y)) :- Y =< 0,Y =< X,X =< -Y,!,
                          X1 is X+1.
% move up
nextPair((X,Y),(X,Y1)) :- X>0, -X < Y, Y < X,
                          Y1 is Y+1.
% move left
nextPair((X,Y),(X1,Y)) :- Y>0, -Y < X,X =< Y,!,
                          X1 is X-1.
% move down
nextPair((X,Y),(X,Y1)) :- X<0, X < Y, Y =< -X,
                          Y1 is Y-1.
```

4.
```
subList([H|T1],[H|T2]) :- subList(T1,T2).
subList(([H|T],L2) :- subList(T,L2).
```

5.
```
countElems([],_,0).
countElems([X|T],[X,Y],N) :-
  !,countElems(T,[X,Y],N1),N is N1+1.
countElems([Y|T],[X,Y],N) :-
  !,countElems(T,[X,Y],N1),N is N1+1.
countElems([_|T],[X,Y],N) :-
  countElems(T,[X,Y],N).
```

6.
```
shift_left([],[]).
shift_left([H|T],L) :- append(T,[H],L).
```

7.
```
shift_right([],[]).
shift_right(Lst1,[H|T]) :- append(T,[H],Lst1).
```

8. (a)
```
fact(1,1):-!.
fact(N,R) :- N>1,factAcc(N,R,1).
factAcc(1,A,A) :-!.
factAcc(N,R,A) :- N1 is N-1,A1 is A*N,factAcc(N1,R,A1).
```

(b)
```
fib(1,1).
fib(2,1).
fib(N,R) :- N>2,fibAcc(N,R,1,1,1).
% fibAcc(+N,-R,+K,+A,+B) binds R to fib_N if A=fib_K and B=fib_{K+1}
fibAcc(N,R,N,R,_) :- !.
fibAcc(N,R,K,A,B) :-
  K1 is K+1,
  B1 is A+B,
  fibAcc(N,R,K1,B,B1).
```

9.
```
isSorted([]).
isSorted([_]).
isSorted([X,Y|T]) :- X =< Y, isSorted([Y|T]).
```

10.
```
elim(E,[E|Rest],Rest).
elim(E,[X|Lst],[X|Rest]) :- elim(E,Lst,Rest).
```

(a)
```
perm([],[]).
perm(Lst,[E|Perm1]) :- elim(E,Lst,Rest),perm(Rest,Perm1).
```

(b)
```
sortList(Lst,S) :- perm(Lst,S),isSorted(S).
```

11.
```
insert(E,[],[E]).
insert(E,[H|T],[E,H|T]) :- E =< H,!.
insert(E,[H|T],[H|T1]) :- insert(E,T,T1).
```

12.
```
insertionSort([],[]).
insertionSort([H|T],S):- insertionSort(T,S1),insert(H,S1,S).
```

13.
```
% split(+H,+L,-T1,-T2)
%   T1 are the elements in L smaller or equal to H
%   T2 are the elements in L larger than H
split(_,[],[],[]).
split(H,[X|T],[X|T1],T2) :- X=<H,!,split(H,T,T1,T2).
split(H,[X|T],T1,[X|T2]) :- split(H,T,T1,T2).

sortV2([],[]).
sortV2([H|T],S) :-
  split(H,T,T1,T2),
  sortV2(T1,S1),
  sortV2(T2,S2),
  append(S1,[H|S2],S).
```

14.
```
twoTimesLonger([],[]).
twoTimesLonger([_|L1],[_,_|L2]) :- twoTimesLonger(L1,L2).
```

15.
```
% sum_and_squareSum(+Lst,-S1,-S2)

% base case
sum_and_squareSum([],0,0).
% recursive case
sum_and_squareSum([H|T],S1,S2) :-
  sum_and_squareSum(T,St1,St2),
  S1 is H+St1, S2 is H*H+St2.
```

16.
```
isPalindrome([]).
isPalindrome([H|T]) :- append(P,[H],T),isPalindrome(P).
```

17.
```
rgb([r|T]) :- rgb(T).
rgb(Lst) :- gb(Lst).
gb([g|T]) :- gb(T).
gb(Lst) :- b(Lst).
b([b|T]) :- b(T).
b([]).
```