CPSC 449: Sample Questions for the Prolog Lab. Exam!

30 March, 2021

Unification

You must know how to unify terms to a parallelized substitution! Give the most general (parallelized) unifier for the following pairs of terms:

1. $f(x, y)$ and $g(y, z)$
2. $f(x, y, z)$ and $f(x, y)$
3. $f(x, y, y)$ and $f(x, x, z)$
4. $f(g(x, y), z)$ and $f(w, g(w, z))$
5. $f(h(x, y), w, h(y, x))$ and $f(x, x, y)$
6. $f(x, g(y, z))$ and $f(g(z, y), x)$
7. $h(u, g(v, w), u)$ and $h(z, z, f(x, y))$
8. $f(g(x, y), z)$ and $f(z, h(x, y))$

Hand evalutions:

1. Hand evaluate `insert(a,[1,2,3],X)` and `insert(H,Y,[a,b,c])`, exploring the backtracking behaviour, where

```
insert(H,T,[H|T]).
insert(H,[X|T],[X|Z]):-
    insert(H,T,Z).
```

2. Show by hand evaluating how Prolog (with all the backtracking behaviour) evaluates:

```
remove([a,b], X,Y)
```

Given the code:

```
remove([H|T],H,T).
remove([H|T],X,[H|Y]):- remove(T,X,Y).
```

3. Show by hand evaluating how Prolog (with all the backtracking behaviour) evaluates:

```
append([a,b], X,Y).
append(X,Y,[a,b]).
```

where

```
append([],X,X).
append([H|T],X,[H|S]):-
      append(T,X,S).
```

4. Consider the following "naive" version of reverse:

```
reverse([],[]).
reverse([H|T],Y):-
     reverse(T,W),
     append(W,[H],Y).
```

Hand evaluate `reverse(X,[a,b,c])`. Why does forcing backtracking of this call cause the computation to fail to terminate. What can be done to correct this (disastrous) behaviour?

5. Consider the following version of "fast" reverse:

```
frev(X,Y):-
    shunt(X,[],Y).

shunt([],X,X).
shunt([H|T],S,Z):-
    shunt(T,[H|S],Z).
```

Hand evaluate `frev(X,[a,b])`. Why does forcing backtracking cause non-termination? What can be done to correct this (distastrous behaviour).

6. Consider the following version for finding permutations of a list:

```
perm([],[]).
perm([H|T],Z):-
    perm(T,S),
    insert(H,S,Z).
```

Hand evaluate it for `perm([a,b],X)` and `perm(X,[a,b])`. Why does backtracking force non-termination in the latter case but, in the former, generate all the permutations? What can you do about this?

Programming in prolog:

1. Explain what a "cut" does. Explain how the predicate `not`/1 (below) works:

   ```
   not(X):-  X,!,false.
   not(X).
   ```

2. Write the grow function in Prolog, `grow(+,-)` where `grow([a,b,c],X)` produces `X = [a,b,b,c,c,c]`.

3. Given a structure write a program `leaves`/2 to collect the leaves of the structure into a list, thus:

   ```
   leaves(a, X)     X=[a]
   leaves(f(a,b,c),X)    X=[a,b,c]
   leaves(f(a,g(b,c),h(d,e,f)),X)     X=[a,b,c,d,e,f]
   ```

   Hint use the built in predicate =../2.

4. What does the following program do? Explain how it works!

   ```
   printer(L):-
        member(X,L),
        write(X),
        nl,
        false.
   printer(_).
   ```

5. Write a predicate, `equal_set(+,+)`, to determine whether two list are equal as sets (order and repetitions do not matter!).

6. Write a predicate to sum every other (first, third, fifth, seventh, ...) element of a list, `odd_sum(+,-)`.

7. Write a predicate which, from a list of integers, extracts all the maximal sublists which are in ascending order, `group(+,-)`.

8. Given an *undirected* graph determined by edges:

   ```
   edge(a,b).
   edge(c,d).
   edge(d,a).
   edge(d,f).
   edge(f,k).
   edge(k,b).
   ```

   Write a predicate `findpath(+,+,-)` which finds a path from a specified start node to a specified end node, `findpath(Start,End,Path)`. The path is should be a sequence of nodes (with no repetitions) in which neighbours have edges between them and which starts at `Start` and ends at `End`.