Bryan Kline

CS326

Homework 7

LaTex (TeXstudio)

12/01/2016

1.
Write the rules for a predicate `reverse(L,L1)`, which succeeds if list `L1` is the list `L` reversed. The following query shows an example of using this predicate:

```
?- reverse([1,2,3], L1).
L1 = [3,2,1]
```

```
%reverse:
reverse([], []).
reverse([H|T], L1) :- reverse(T, L2), append( L2, [H], L1).
```

2.
Write the rules for a predicate `take(L,N,L1)`, which succeeds if list `L1` contains the first `N` elements of list `L`, in the same order. The following queries show examples of using this predicate:

```
?- take([5,1,2,7], 3, L1).
L1 = [5,1,2]
?- take([5,1,2,7], 10, L1).
L1 = [5,1,2,7]
```

NOTE: I made another function to return the number of elements in a list so that if the number given to `take` is larger than the size of the list it just returns the list.

```
%numElements:
numElements([], 0).
numElements([_|T], X) :- numElements(T, X1), X is X1 + 1.
```

```
%take:
take(_, 0, []).
take(L1, N, L1) :- numElements(L1, X), X =< N.
take([H|T], N, L1) :- X is N - 1, take(T, X, L2),
                                        append([H], L2, L1).
```

3.
Consider the following definition of a binary tree in Prolog, where a tree is either the constant `nil`, or a structure `node` with 3 elements, the second and third elements also being trees:

```
tree(nil).
tree(node(_,Left,Right)) :- tree(Left), tree(Right).
```

a)
Write the rules for a predicate `nleaves(T,N)`, which succeeds if `N` is the number of leaves in the tree `T`. The following query shows an example of using this predicate:

```
?- nleaves(node(1, node(2, node(3,nil,nil), node(4,nil,nil)),
node(5,nil,nil)), N).
N = 3
```

```
%nleaves:
nleaves(node(_, nil, nil), N) :- N is 1.
nleaves(node(_, Left, Right), N) :- nleaves(Left, N2),
                        nleaves(Right, N3), N is N2 + N3.
```

b)
Write the rules for a predicate `treeMember(E,T)`, which succeeds if `E` appears as an element in the tree `T`. The following query shows an example of using this predicate:

```
?- treeMember(3, node(1, node(2, node(3,nil,nil),
node(4,nil,nil)), node(5,nil,nil))).
Yes
```

```
%treeMember:
treeMember(E, node(E, _, _)).
treeMember(E, node(_, Left, _)) :- treeMember(E, Left).
treeMember(E, node(_, _, Right)) :- treeMember(E, Right).
```

c)
Write the rules for a predicate `preOrder(T,L)`, which succeeds if `L` is a list containing all elements in the tree `T` corresponding to a pre-order traversal of the tree. The following query shows an example of using this predicate:

```
?- preOrder(node(1, node(2, node(3,nil,nil), node(4,nil,nil)),
node(5,nil,nil)), L).
L = [1, 2, 3, 4, 5]
```

```
%preOrder:
preOrder(node(N, nil, nil), [N]).
preOrder(node(H, Left, nil), [H|L]) :-
                    preOrder(Left, L1), append(L1, [], L).
preOrder(node(H, nil, Right), [H|L]) :-
                    preOrder(Right, L1), append([], L1, L).
preOrder(node(H, Left, Right), [H|L]) :-
            preOrder(Left, L1), preOrder(Right, L2),
                                        append(L1, L2, L).
```

d)
The *height* of a tree is defined as the maximum number of nodes on a path from the root to a leaf. Write the rules for a predicate `height(T,N)`, which succeeds if `N` is the height of the tree `T`. The following query shows an example of using this predicate:

```
?- height(node(1, node(2, node(3,nil,nil), node(4,nil,nil)),
node(5,nil,nil)), N).
N = 3
```

```
%height:
height(node(_, nil, nil), N) :- N is 1.
height(node(_, Left, nil), N) :-
                              height(Left, N1), N is N1 + 1.
height(node(_, nil, Right), N) :-
                              height(Right, N1), N is N1 + 1.
preOrder(node(H, Left, Right), [H|L]) :-
            preOrder(Left, L1), preOrder(Right, L2),
                                        append(L1, L2, L).
```

You may want to use the predefined arithmetic function `max(X,Y)`.

4.
Write the rules for a predicate `insert(X,L,L1)`, which succeeds if list `L1` is identical to the sorted list `L` with `X` inserted at the correct place. Assume that `L` is already sorted. The following query shows an example of using this predicate:

```
?- insert(5, [1,3,4,7], L1).
L1 = [1,3,4,5,7]
```

```
%insert:
insert(N, [], [N]).
insert(N, [H|T], L) :- N < H, append([N], [H|T] , L).
insert(N, [H|T], L) :- N >= H, insert(N, T, L1),
                                        append([H], L1, L).
```

5. Extra Credit
Write the rules for a predicate `flatten(A,B)`, which succeeds if `A` is a list (possibly containing sublists), and `B` is a list containing all elements in `A` and its sublists, but all at the same level. The following query shows an example of using this predicate:

```
?- flatten([1, [2, [3, 4]], 5], L).
L = [1, 2, 3, 4, 5]
```

```
%flatten:
flatten([H|[]], [H]).
flatten([[H|T1]|T2], L) :- flatten([H], L4),
                flatten(T1, L1), flatten(T2, L2),
                    append(L1, L2, L3), append(L4, L3, L).
flatten([H|T], L) :- flatten([H], L1),
                        flatten(T, L2), append(L1, L2, L).
```

NOTE: This doesn't quite work, but it will flatten out most of a list. It needs more work, but I don't have time to finish it.