COMP90048 Declarative Programming
Semester 1, 2018
Peter J. Stuckey

QUESTION 1

This definition of sumlist/2 is not tail recursive:

```
sumlist([], 0).
sumlist([N|Ns], Sum) :-
    sumlist(Ns, Sum0),
    Sum is N + Sum0.
```

Rewrite it to be tail recursive.

ANSWER

```
sumlist(List, Sum) :-
    sumlist(List, 0, Sum).

sumlist([], Sum, Sum).
sumlist([N|Ns], Sum0, Sum) :-
    Sum1 is Sum0 + N,
    sumlist(Ns, Sum1, Sum).
```

QUESTION 2

Given a binary tree defined to satisfy the predicate tree/1

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

write a predicate tree_list(Tree, List) that holds when List is a list of all the elements of Tree, in left-to-right order. This code need only work in the mode where the tree is input.

ANSWER

```
tree_list(empty, []).
tree_list(node(Left,Elt,Right), List) :-
    tree_list(Left, List1),
```

```
    tree_list(Right, List2),
    append(List1, [Elt|List2], List).
```

QUESTION 3

Revise the definition from the previous question not to use append/3 to construct the list. That is, ensure the cost of the operation is proportional to the number of elements in the tree.

Hint: look at the approach taken to write a tail recursive reverse predicate for inspiration.

ANSWER

```
tree_list(Tree, List) :-
    tree_list(Tree, List, []).

tree_list(empty, List, List).
tree_list(node(Left,Elt,Right), List, List0) :-
    tree_list(Left, List, List1),
    List1 = [Elt|List2],
    tree_list(Right, List2, List0).
```

Note that we've swapped the order of the last two arguments from what you might expect. This is more natural, since

```
    tree_list(Tree, List, List0)
```

then says "List is the list of the elements of Tree with List0 appended to the end".

QUESTION 4

Write a predicate list_tree(List, Tree) that holds when Tree is a balanced tree whose elements are the elements of List, in the order they appear in List. This need only work in the mode where List is a proper list.

Hint: First divide the list into the first half, the middle element, and the last half, then recursively construct a tree from the first half and second half, and assemble these into the resulting tree.

Clarification: The main intended use of this predicate is to build a balanced tree from a list. It should, as far as

reasonable, work in other modes. One complication is that it's possible for there to be different trees, balanced in different ways, with the same nodes in order. You don't need to handle this possibility here (though it would make an interesting challenge exercise). You just need to handle balanced trees that could have been produced by this predicate.

ANSWER

```
list_tree([], empty).
list_tree([E|List], node(Left,Elt,Right)) :-
    length(List, Len),
    Len2 is Len // 2,
    length(Front, Len2),
    append(Front, [Elt|Back], [E|List]),
    list_tree(Front, Left),
    list_tree(Back, Right).
```