Functional Programming 1 — 1DL330

Assignment 3

Lab: Friday, 22 September

Submission Deadline: 18:00, Tuesday, 26 September, 2017

This assignment should be solved in **groups of two** students. Please join a group in the *Assignment 3* group division on the Student Portal. If you have not found a group partner by the start of the lab for this assignment, please inform the lab assistant. If you would prefer to solve this assignment individually due to special circumstances, please contact the assistant teacher, svenolof@it.uu.se.

Instructions

- Start by reading the General Lab Instructions (on the Student Portal).
- Name the file containing your solutions `lab3.sml`. The file that you submit must contain valid SML code, so the answers to some of the questions need to be placed in comments: *(∗ ... ∗)*.
- Remember to follow our Coding Convention (also on the Student Portal), and provide specifications for all functions that you write.
- Make sure that your solution passes the tests in `lab3_test.sml` before you submit. A submission that does not pass these tests will get grade K.

1 Tail Recursion

Write a function average that computes the average of a list of real numbers. Example: average [1.0,4.0,10.0] = 5.0.

Requirement: Use exactly one helper function. This helper function must be tail-recursive. (Do *not* use library functions, such as length, that recurse over a list.)

2 Specification

Give specifications and variant for the function average and its helper function.

See the Coding Convention (on the Student Portal) about specifications and variants.

3 Use of Higher-Order Functions

You have already seen definitions for the functions append, member, last, reverse and filter. Now your task is to give non-recursive definitions for these functions, using the higher-order functions foldl or foldr. Define

1. append : 'a list -> 'a list -> 'a list
2. member : ''a -> ''a list -> bool
3. last : 'a list -> 'a
4. reverse : 'a list -> 'a list
5. filter : ('a -> bool) -> 'a list -> 'a list

In all cases, the definition using foldl or foldr should be quite simple.

Obviously, the function last cannot yield a meaningful result for the empty list. Make sure that applying last to the empty list gives an error.

4 Binary Search Trees

The nodes of a binary search tree are ordered from left to right according to their key value. Consider the following datatype definition of binary search trees:

```
datatype tree = Void | Node of tree * int * tree
```

Define a function sub_tree: int -> int -> tree -> tree where sub_tree a b t yields a binary search tree containing the keys in t that are greater than or equal to a but smaller than b.

Example: Assume that ex1 is defined as follows:

```
val ex1 = Node(Node(Node(Void, 0, Node(Void,2,Void)),
                    3,
                    Node(Void, 5, Void)),
               6,
               Node(Void,
                    7,
                    Node(Void, 8, Node(Void,9,Void))));
```

We now have

```
sub_tree 5 8 ex1 = Node(Node(Void,5,Void),6,Node(Void,7,Void))
```

(The tree produced by your solution might be shaped differently, but it should also contain the keys 5, 6 and 7, and no other keys.) We also have

```
sub_tree 10 20 ex1 = Void
```

This question can (and should) be solved without introducing intermediate data structures (such as lists).

5 Complexity

Assume that `a` and `b` are fixed. What is the (worst-case) time complexity of your function `sub_tree a b t`? (The size of the input tree `t` is given by the number of nodes in `t`.)

Testing

Use the file `lab3_test.sml` (from the Student Portal) to test your solution. See the instructions for Assignment 1 for further information on running the tests.

Some tests might fail because they assume other datatype definitions or function signatures than what you are using. If necessary, *change your solutions* to make all tests compile and run successfully.

Good luck!