

Functional Programming 1 — 1DL330

Assignment 2

Lab: Thursday, 14 September

Submission Deadline: 18:00, Tuesday, 19 September, 2017

This assignment should be solved in **groups of two** students. Please join a group in the *Assignment 2* 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 our lab assistant, albert.yang@it.uu.se.

Instructions

- Start by reading the General Lab Instructions (on the Student Portal).
- Name the file containing your solutions `lab2.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 `lab2_test.sml` before you submit. A submission that does not pass these tests will get grade K.

1 Iota

Define a function `iota` so that `iota n` returns the list `[0, 1, 2, 3, ..., n-1]`. For example, `iota 5` should return `[0, 1, 2, 3, 4]`.

2 Intersection

Finite sets of integers can be represented by lists of integers. For this exercise we only consider lists without duplicates, i.e., every element is contained in the list at most once. Sometimes it may be useful to also require that the lists are ordered.

- Examples of ordered lists: `[6, 17, 23, 89]`, `[]`, `[~3,~2,~1]`
- Example of a list that is not ordered: `[17, 23, 6, 89]`

1. First, do not assume that the input lists are ordered. Define a function `inter s1 s2` that returns a list representing the set $s1 \cap s2$, i.e., the set containing the elements that occur in both argument lists. This is also known as the intersection of the two sets.

Hint: First define a helper function `member` that checks if a value occurs in a list.

2. Now, assume that the input lists are ordered. Naturally, to compute the intersection of two ordered lists, the previous solution will still work. However, it is possible to write a more efficient function `inter' s1 s2` that assumes that the lists are ordered, and only visits each element of either list once (by recursing over both `s1` and `s2` simultaneously). Define `inter'`.
3. Verify that `inter'` is indeed faster than `inter` by testing on some long lists. Using the function `iota` it is easy to create lists with, say, 100,000 or 1,000,000 elements. You can use the following code to measure how fast your function is:

```
(* real_time f
   TYPE: (unit -> 'a) -> Time.time * 'a
   PRE : true
   POST: (the amount of (real) time spent evaluating f (), f ())
   SIDE-EFFECTS: any side-effects caused by evaluating f ())
*)
fun real_time f =
  let
    val rt = Timer.startRealTimer()
    val result = f ()
    val time = Timer.checkRealTimer rt
  in
    (time, result)
  end

val result =
  let
    val s1 = iota 100000
    val s2 = iota 1000000
    val slow_time_and_res = real_time (fn () => inter s1 s2)
    val fast_time_and_res = real_time (fn () => inter' s1 s2)
  in
    (slow_time_and_res, fast_time_and_res)
  end
```

If your implementation of `iota` is well-written, it should be able to create lists this long quickly. Report the results of your measurements.

3 Fruit

We consider three kinds of fruit: apples, bananas and lemons. Bananas and apples are sold by the kilogram, but a lemon always has the same price, regardless of weight.

1. Give a datatype declaration for the type `fruit`. The datatype should reflect that there are three kinds of fruit, so the datatype definition should have three cases.

For apples (**Apple**) and bananas (**Banana**) there should be an associated weight (given as a value of type `real`). For lemons (**Lemon**) there should be an associated number of units (given as a value of type `int`).

2. Define a function `sumPrice : fruit list -> real -> real -> real -> real`. This function should take a list of fruit, the price of apples (per kilogram), bananas (per kilogram) and the price of lemons (per unit), and return the total cost of the items in the list. Give specification and variant for `sumPrice`.

4 Trees

1. Define a datatype `ltree` of (finitely branching) labeled trees. Each node (**Node**) carries a label (of polymorphic type `'a`) and may have an arbitrary (non-negative) number of children. Different nodes may have different numbers of children. Each tree has at least one node (so there should be no `Void` case in your datatype definition).

See https://en.wikipedia.org/wiki/Tree_%28data_structure%29 for further definitions and some hints.

2. Define functions to

- (a) compute the number of nodes in such a tree (`count : 'a ltree -> int`),
- (b) compute the list of all node labels in such a tree (`labels : 'a ltree -> 'a list`),
- (c) check if a given value is present in such a tree, i.e., if some node is labeled with this value (`is_present : 'a ltree -> 'a -> bool`),
- (d) compute the height of such a tree (`height : 'a ltree -> int`).

Testing

Use the file `lab2_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!