Functional Programming 1

Assignment 2

.

Instructions

- Start by reading the General Lab Instructions.
    - 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, and pro- vide specifications for all functions that you write.
    - Make sure that your solution passes the tests in `lab2 test.sml`.

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 ∩ 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 real value). For lemons (Lemon) there should be an associated number of units (given as a int value).

2. Define the 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 equality type '' a) and may have an arbitrary (non- negative) number of children. Different nodes may have different numbers of chil- dren. 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).

5 Testing

Use the file `lab2 test.sml` to test your solution.

1. Place the lab2 test.sml in the same directory as your solution file lab2.sml
2. Start Poly/ML in that directory: eg. cd lab2 dir; poly.
3. Enter use "lab2 test.sml"; at that Poly/ML prompt to run the tests.
4. Check the output for failing tests.
5. Quit Poly/ML (eg. by typing Ctrl+D)
6. Fix all failing tests by modifying your lab2.sml file.
7. Repeat steps 2 - 6 , until all tests succeed.

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!