DTU Compute

Lyngby 04-09-2023
Michael R. Hansen

Week 3: Exercises for the lecture slot

The exercises below are based on old exam questions that concern fundamental concepts introduced during the first two weeks.

Be aware that **no aids are allowed** the final exam. Therefore, try first to solve these exercises using paper and pencil only. This will give you some feedback concerning how well the concepts are understood.

Type in and test your solutions on a computer after you have solved the exercises.

Problem 1 (Approx 30 minutes)

All questions in this problem should be solved without using functions from the libraries `List`, `Seq`, `Set` and `Map`.

1. Declare a function `numberOf` x ys that returns the number of times x occurs in the list ys. For example, `numberOf 2 [0;2;3;3;0;2;4;2;1]` = 3.

2. Declare a function `positionsOf` x ys that returns the list containing the positions of occurrences of x in the list ys. For example, `positionsOf 2  [0;2;3;3;0;2;4;2;1]` = $[1;5;7]$. Notice that the position of the first element of a non-empty list is 0.

   Hint: You may consider introducing a helper function.

3. Declare a function `filterMap: ('a->bool) -> ('a->'b) -> 'a list -> 'b list`. The value of `filterMap` p f xs is the list obtained from xs by applying f to the elements that satisfy the predicate p.

   For example, `filterMap (fun x -> x>=2) string [0;2;3;3;0;2;4;2;1]` returns the list ["2";"3";"3";"2";"4";"2"].

Problem 2 (Approx 25 minutes)

Consider the declaration:

```
let rec splitAt i xs =
   if i<=0 then ([],xs)
   else match xs with
        | []      -> ([],[])
        | x::tail -> let (xs1,xs2) = splitAt (i-1) tail
                     (x::xs1,xs2);;
```

1. What are the values of
   - `splitAt -1 [1;2;3]`,
   - `splitAt 3 [1;2;3;4;5]` and
   - `splitAt 4 [1;2;3]`.
2. What is the type of `splitAt`? Justify your answer briefly.
3. Describe what `splitAt` is computing by stating the value of

$$\texttt{splitAt}\ k\ [x_0; x_1; \ldots; x_{n-1}],\ \text{where}\ n \geq 0.$$

Problem 3 (Approx 20 minutes)

Consider the following F# declarations:

```
let rec f(xs,rs) = match xs with
                   | []   -> rs
                   | [x]  -> x::rs
                   | x1::x2::xs -> x1::f(xs,x2::rs)
let g xs = f(xs,[]);;
```

1. Give a step-by-step evaluation (using $\rightsquigarrow$) for `g [1;2;3;4;5]` determining the value of the expression. There should at least be one step for every recursive call of `f`.
2. Give the types for `f` and `g`, and describe what `g` computes. Your description should focus on *what* it computes, rather than on individual computation steps.