02157 Functional programming

Michael R. Hansen
DTU COMPUTE
August 31, 2017

Exercises for Sept. 22^{nd}: Old exam questions

Many central concepts of functional programming are already covered during the first two weeks of the semester, and you should now be equipped to solve this exercise set, that is based on old exam questions in 02157 Functional programming.

Please note that the allowed aids at the exam are: "All written material". Thus, you will *not* have laptop, tablet, etc. available at the exam. You are, therefore, encouraged to solve the problems using paper and pencil before you test your solutions on the computer.

In order to focus on fundamental concepts and program construction techniques, your are strongly encouraged to solve these exercises without using libraries like the `String` and `List` libraries.

If a question requires you to define a particular function, then you may define as many helper functions as you want, but in any case you must define the required function so that it has exactly the type and effect that the question asks for.

You may use a function specified in earlier in the exercise set in a solution to a problem – even when you do not provide a declaration for the used function.

Problem 1 (approx. 30 minutes) From exam May 29th, 2015

1. Declare a function: `repeat: string -> int -> string`, so that `repeat` $s\,n$ builds a new string by repeating the string s altogether n times. For example: `repeat "ab" 4` = `"abababab"` and `repeat "ab" 0 = ""`.

2. Declare a function `f` $s_1\,s_2\,n$ that builds a string with n lines alternating between s_1 and s_2. For example: `f "ab" "cd" 4` = `"ab\ncd\nab\ncd"` and `f "XO" "OX" 3` = `"XO\nOX\nXO"`. Note that `\n` is the escape sequence for the newline character.
   Give the type of the function.

3. Consider now certain patterns generated from the strings `"XO"` and `"OX"`. Declare a function `viz` $m\ n$ that gives a string consisting of n lines, where

   - the first line contain m repetitions of the string `"XO"`,
   - the second line contain m repetitions of the string `"OX"`,
   - the third line contain m repetitions of the string `"XO"`,
   - and so on.

   For example, `printfn "%s" (viz 4 5)` should generate the following output

```
XOXOXOXO
OXOXOXOX
XOXOXOXO
OXOXOXOX
XOXOXOXO
```

Problem 2 (approx. 12 minutes) From exam Dec 18th, 2014

Consider the following declaration:

```
let rec f i = function
              | []  -> []
              | x::xs -> (x+i)::f (i*i) xs;;
```

1. Give the (most general) type of `f` and describe what `f` computes. Your description should focus on *what* it computes, rather than on individual computation steps. Hint: You may describe the value of `f` $i\ [x_0; x_1; x_2; \ldots; x_n]$.

Problem 3 (approx. 48 minutes) From exam Dec. 18th, 2014

We consider *relations* that are represented by lists of pairs: $[(x_0, ys_0); (x_1, ys_1); \ldots; (x_n, ys_n)]$. We say that x is related to y when there is a pair (x_i, ys_i) in the list where $x = x_i$ and y is an element of the list ys_i. The following type is used for relations:

```
type Rel<'a,'b when 'a: equality> = ('a * 'b list) list

let rel: Rel<int,string> = [(1, ["a"; "b"; "c"]); (4,["b"; "e"])];;
```

The value `rel` describes a relation where, for example, 1 and "b" and 4 and "e" are related, while 1 and "e" and 2 and "a" are not related.

We require that the x_i's in $[(x_0, ys_0); (x_1, ys_1); \ldots; (x_n, ys_n)]$ are all different; but we do not care about repetitions and the order of the elements in ys_i.

1. Declare a function: `apply: 'a -> Rel<'a,'b> -> 'b list`, where `apply` x rel finds the list of elements related to x in rel. For example: `apply 1 rel` = `["a"; "b"; "c"]` and `apply 0 rel = []`.
2. Declare a function `inRelation` x y rel that checks whether x and y are related in rel. For example, `inRelation 4 "e" rel = true` and `inRelation 1 "e" rel = false`.
3. Declare a function `insert` x y rel which returns the relation obtained from rel by adding that x is related to y. For example: `insert 2 "c" [(1,["a"]); (2,["b"])]` could give `[(1, ["a"]); (2, ["c"; "b"])]`.
4. Declare a function `toRel:('a*'b) list -> Rel<'a,'b>` that converts a list of pairs to a relation, e.g. `toRel[(2,"c");(1,"a");(2,"b")]` could give `[(2,["c";"b"]);(1,["a"])]`.

Problem 4 (approx. 24 minutes) From exam May 24th, 2017

1. Declare a function `repeatList: 'a list -> int -> 'a list`, so that

   `repeatList` $xs\, n = xs$ `@` xs `@` $\cdots$ `@` xs, with n occurrences of xs

   For example, `repeatList` $[1;2]$ $3 = [1;2;1;2;1;2]$ and `repeatList` $[1;2]$ $0 = [\,]$.

2. Declare a function `merge: 'a list * 'a list -> 'a list`, so that

$$\texttt{merge}([x_0; x_1; \ldots; x_m], [y_0; y_1; \ldots; y_n]) = \begin{cases} [x_0; y_0; x_1; y_1 \ldots; x_m; y_m; y_{m+1}; y_{m+2}; \ldots; y_n] & \text{when } m < n \\ [x_0; y_0; x_1; y_1 \ldots; x_m; y_m] & \text{when } m = n \\ [x_0; y_0; x_1; y_1 \ldots; x_n; y_n; x_{n+1}; x_{n+2}; \ldots; x_m] & \text{when } m > n \end{cases}$$

   That is, the function `merge` can merge the elements of two lists, where the lists need not have the same size. For example, `merge`$([1;2],[3;4]) = [1;3;2;4]$, `merge`$([1;2],[3;4;5]) = [1;3;2;4;5]$ and `merge`$([1;2;3;4],[5;6]) = [1;5;2;6;3;4]$.