INFOAFP Assignments

AFP Assignment 1

Deadline: Friday, November 23, 2012, 24:00

General remarks

- Mail your solution as a single file `doaitse@swierstra.net`, with in the subject "Assign-1: *name1* and *name2*" and as filename "afp2012-name1-name2.zip"
- Team size: preferably 2, but 1 is possible.
- Programming style and quality of Haddock comments influences the grade.
- Put Haddock information in your code, so as not to make it a puzzel to mark it.
- Gathering information on the internet is okay, but copying entire solutions from the internet (or elsewhere) is not allowed.

1 (10%)**.** Consider a function of type

$$runIO :: IO\ a \to a$$

Why is such a function dangerous? (There are several reasons. Try to give example programs that are dangerous or even demonstrate that something strange and unexpected is going on.)

2 (15%)**.** Consider the datatype

$$\textbf{data}\ Tree\ a = Leaf\ a \mid Node\ (Tree\ a)\ (Tree\ a)$$

The function *splitleft* splits off the leftmost entry of the tree and returns that entry as well as the remaining tree:

$$\begin{aligned} &splitleft :: Tree\ a \to (a, Maybe\ (Tree\ a)) \\ &splitleft\ (Leaf\ a) \quad = (a, Nothing) \\ &splitleft\ (Node\ l\ r) = \textbf{case}\ splitleft\ l\ \textbf{of} \\ &\qquad\qquad\qquad\qquad (a, Nothing) \to (a, Just\ r) \\ &\qquad\qquad\qquad\qquad (a, Just\ l') \quad \to (a, Just\ (Node\ l'\ r)) \end{aligned}$$

Write a *tail-recursive* variant of *splitleft*.

Hint. Generalize *splitleft* by introducing an additional auxiliary parameter. Recall that functions can be used as parameters. If you do not know what "tail-recursive" means, look up the definition somewhere.

3 (25%)**.** In some FP exam I have asked to write a function

$$smooth\_perms :: Int \to [Int] \to [[Int]]$$

which returns all permutations of its second argument for which the distance between each two successive elements is at most the first argument.

```
module Perms where
split [] = []
split (x:xs) = (x,xs) : [(y,x:ys) | (y,ys) <- split xs]
perms [] = [[]]
perms xs = [(v:p) | (v,vs) <- split id xs, p <- perms vs]
smooth n (x:y:ys) = abs (y-x) <= n && smooth n (y:ys)
smooth _ _ =        True
CC
smooth_perms :: Int -> [Int] -> [[Int]]
smooth_perms n xs = filter (smooth n) (perms xs)
```

A straightforward solution is to generate all permutations and then to filter out the smooth ones. This however is expensive. A better approach is to build a tree, for which it holds that each path from the root to a leaf correspond to one of the possible permutations, next to prune this tree such that only smooth paths are represented, and finally to use this tree to generate all the smooth permutations from.

Now define this tree data type, a function which maps a list onto this tree, the function which prunes the tree, and finally the function which generates all permutations.

Give a *quickCheck* specification and check, by defining a function *allSmoothPerms*.

4 (25%)**.** Consider the following definitions:

```
data Tree a = Leaf a
            | Node (Tree a) (Tree a)
  deriving Show
size :: Tree a -> Int
size (Leaf a)   = 1
size (Node l r) = size l + size r
length :: [a] -> Int
length []     = 0
length (x:xs) = 1 + length xs
```

```
flatten :: Tree a -> [a]
flatten (Leaf a)   = [a]
flatten (Node l r) = flatten l ++ flatten r
(++) :: [a] -> [a] -> [a]
[]     ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

Prove the following Theorem using equational reasoning:

$$\forall(t :: \mathit{Tree}\ a).\mathit{length}\ (\mathit{flatten}\ t) \equiv \mathit{size}\ t$$

Note that using the induction principle on trees, it is sufficient to show the following two cases:

$$\forall(x :: a).\mathit{length}\ (\mathit{flatten}\ (\mathit{Leaf}\ x)) \equiv \mathit{size}\ (\mathit{Leaf}\ x)$$

and

$$\begin{array}{l}\forall(l :: \mathit{Tree}\ a)\ (r :: \mathit{Tree}\ a).\\ \quad (\ \mathit{length}\ (\mathit{flatten}\ l) \equiv \mathit{size}\ l\\ \quad \wedge\ \mathit{length}\ (\mathit{flatten}\ r) \equiv \mathit{size}\ r\\ \quad)\ \rightarrow \mathit{length}\ (\mathit{flatten}\ (\mathit{Node}\ l\ r)) \equiv \mathit{size}\ (\mathit{Node}\ l\ r)\end{array}$$

To prove the Theorem, you will need to prove the following Lemma first:

$$\forall(xs :: [a])\ (ys :: [a]).\mathit{length}\ (xs +\!\!+ ys) = \mathit{length}\ xs + \mathit{length}\ ys$$

You may use facts about $(+)$ such that $(+)$ is associative or that 0 is the neutral element of addition.

5 (25%)**.** Submit your solutions to the assignments as a Cabal package. Turn the code (for the *perms_smooth* task) into a library. Include the solutions to the theoretical packages as extra documentation files. Write a suitable package description, and include a Setup script so that the package can easily be built and installed using Cabal. As a package name, choose a name that includes your names. Produce the file using the command "cabal dist".