FP 2011-2012, Eindtoets
2011, Nov 8 , 17.00-20(21).00, EDUC Beta

Only hand in both answer sheets (**with your name on both sheets!!**), while showing your legitimation. Do not forget necessary parentheses! In case you want to withdraw your submission since your guess is that it will, in combination of your mark for the first test, lead to a final mark < 4 you may do so by sending an email to doaitse@swierstra.net before tomorrow 9.30. In this case you do not run the risk to loose the right to take part in the reseating. Good luck!

1 The function *foldl′* (1)

Give the type and the definition of the function *foldl′*. Give an example where its use is profitable and an example where its use is not giving the desired effect at all.

2 Fair enumeration(1)

Define a value $enumInts :: [(Int, Int)]$ in which the distance from an occurrence of any value from the set $\{(x, y) \mid x \in Int, y \in Int\}$ to the beginning of the list is a finite value (note that *Int*'s can also be negative).

3 Permutations(1)

Write a function $permutations :: [a] \to [[a]]$ which returns all permutations of its parameter.

4 Side effects(1)

Someone writes the following program and does not get any output.

```
import System.Random
createRandomValues   = sequence (repeat randomIO)
printRandomValues n = do randomValues ← createRandomValues
                         print (take n randomValues)
main = printRandomValues 10
```

Rewrite the program such that it does what the code suggests, i.e. printing 10 random numbers

5 Is *tja* correct?(1)

Given the data type

```
data Tree a = Leaf a
            | Node (Tree a) (Tree a)
```

we define the function *tja*:

```
tja t = let tja′ (Leaf a)   0 ls = (0, a : ls)
            tja′ (Leaf _)   n ls = (0, ls)
            tja′ (Node l r) n ls = let (lm, ll) = tja′ l (n − 1) rl
                                       (rm, rl) = tja′ r (n − 1) ls
                                   in ((lm ‘min‘ rm) + 1, ll)
            (m, r) = tja′ t m []
        in r
```

If this code computes something explain what it computes (small example?); if it does not compute anything explain why this is the case.

6 The function *enumBf* (1)

Write a function $enumBf :: Tree\ a \to [a]$ which returns a list which contains the a-values from the leaves resulting from a breadth-first enumeration (i.e. leaves at a lower depth occur earlier in the list). Hint: use a helper function $enumBf' :: [Tree\ a] \to [a]$.

7 Parsing(2)

We can define a somewhat simplistic data type *XML* and a parser for it:

$$\begin{aligned}
&\textbf{type}\ Tag = String\\
&\textbf{data}\ XML = Tag\ Tag\ [XML]\\
&\qquad\qquad \mid\ Content\ String\\
&pXML\ =\ (pOpenTag \gg\!= (\lambda t \to Tag\ t <\$> pMany\ pXML <*\ \ pCloseTag\ t))\\
&\qquad <|> Content <\$> pString
\end{aligned}$$

Write the functions *pOpenTag* and *pCloseTag*. Write a parser $pXML'$ which also recognises attributes, and returns the result as a value of type XML'. You may assume that *pString* takes care of escaping special characters. Assume also that *pString* and *pSym* remove any trailing whitespace (i.e. you do not have to worry about spaces, newlines, tabs, etc).

$$\begin{aligned}
&\textbf{data}\ XML' = Tag'\ Tag\ Attrs\ [XML']\\
&\qquad\qquad \mid\ Content'\ String\\
&\textbf{type}\ Attrs = [(String, String)]
\end{aligned}$$

An example input might be:

```
<TABLE COLS="1" BORDER="0" CELLSPACING="4" CELLPADDING="5">
        <TR>
          <TD COLSPAN="2" WIDTH="100%" BGCOLOR="#99CCFF" ALIGN="CENTER">
            <B>Functional programming in the Netherlands</B>
          </TD>
        </TR>
</TABLE>
```

8 Heaps (2)

A *heap* is a data structure described by a data type quite similar to a search tree:

$$\begin{aligned}
&\textbf{data}\ Heap\ a = Top\ a\ (Heap\ a)\ (Heap\ a)\\
&\qquad\qquad \mid\ Empty
\end{aligned}$$

with the property that the a value in a *Top* node dominates ($\geqslant$) all the values contained in its two children, which have this property themselves too.

1. Write a function $checkHeap :: Ord\ a \Rightarrow Heap\ a \to Bool$ which returns *True* if its argument is a *heap*, and *False* otherwise. Hint: you may want to write a helper function $checkHeap' :: Ord\ a \Rightarrow a \to Heap\ a \to Bool$.

2. Write a function $mergeHeaps :: Ord\ a \Rightarrow Heap\ a \to Heap\ a \to Heap\ a$ which combines its two arguments into a *heap*

3. Write the function $enumHeap :: Ord\ a \Rightarrow Heap\ a \to [a]$ such that the value r:

   $$v = enumHeap\ .\ foldr\ mergeHeaps\ Empty\ \$\ [\,Top\ x\ Empty\ Empty \mid x \leftarrow [1\,..\,10]]$$

   evaluates to $[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$, i.e. the elements stored in the heap come out in reversed sorted order.

1

Name: **Student nr:** **Program: Inf/CKI/...**

1	2	3	4	5	6	7	8

Leave the above boxes blank.

QUESTION 1: The function $foldl'$.

$$
\begin{aligned}
&foldl'\ op\ e\ [\,] = e \\
&foldl'\ op\ e\ (x : xs) = \mathbf{let}\ newe = e\ `op`\ x \\
&\qquad\qquad\qquad\qquad\quad \mathbf{in}\ newe\ `seq`\ foldl'\ op\ newe\ xs \\
&sum = foldl'\ (+)\ 0 \qquad \text{-- good use} \\
&and = foldl'\ (\wedge)\ \ True \quad \text{-- bad use}
\end{aligned}
$$

QUESTION 2: Fair enumeration:

$$
\begin{aligned}
pairs = concat\ \ & [\ [(x, y), (-x, y), (-x, -y), (x, -y)] \\
& \mid sum \leftarrow [0\,..\,] \\
& ,\ x \quad\ \leftarrow [0\,..\,sum] \\
& ,\ \mathbf{let}\ y = sum - x \\
&]
\end{aligned}
$$

QUESTION 3: *permutations*:

$$
\begin{aligned}
&split :: [\,a\,] \to [(a, [\,a\,])] \\
&split\ [\,] \qquad\qquad = [\,] \\
&split\ (x : xs) \qquad = (x, xs) : [(y, x : zs) \mid (y, zs) \leftarrow split\ xs\,] \\
&permutations\ [\,] = [[\,]] \\
&permutations\ l\ \ = [h : p \mid (h, t) \leftarrow split\ l, p \leftarrow permutations\ t\,]
\end{aligned}
$$

QUESTION 4: Print random numbers

```
createRandomValues 0 = return []
createRandomValues n = do r  <- (randomIO :: IO Int)
                          rr <- createRandomValues (n - 1)
                          return (r : rr)
  -- or
createRandomValues' n = sequence [randomIO | _ <- [1 .. n]]
printRandomValues n = do nr <- createRandomValues n
                         print nr
```

QUESTION 5: *tja*?

The function *tja* returns a list of leaf values which "live" at the greatest depth where the tree structure is still complete. If we define *the minimal depth* of a tree as the minimal length of a path from the root to some leaf (which is computed as the first component ogf the result), then *tja* returns the list of leaves at the end of the paths of which the length equals this minimal depth. This is computed as the second component of the result (as an accumulating parameter)

QUESTION 6: *enumBf*:

```
enumBf :: Tree a -> [a]
enumBf t = enumBf' [t]
  where enumBf' ts = [a | Leaf a <- ts]
                     ++
                     (enumBf' . concat $ [[l, r] | Node l r <- ts])
```

2

Name: **Student nr:** **Program: Inf/CKI/...**

QUESTION 7: Parsing XML:

$$
\begin{aligned}
&pOpenTag \quad = pSym\ \texttt{'<'}\ \ *\!> pString <\!* \ \ pSym\ \texttt{'>'} \\
&pCloseTag\ t = pToken\ (\texttt{"<\textbackslash\textbackslash"} \mathbin{++} t \mathbin{++} \texttt{">"}) \\
&pOpenTag' = (,) <\$\ pSym\ \texttt{'<'} \\
&\qquad\qquad <\!*\!> pString <\!*\!> pMany\ ((,) <\$\!> pString <\!* \ \ pSym\ \texttt{'='} <\!*\!> pString) \\
&\qquad\qquad <\!* \ \ pSym\ \texttt{'>'} \\
&pXML' = (pOpenTag' \gg\!= (\lambda(t, as) \to Tag'\ t\ as <\$\!> pMany\ pXML' <\!* \ \ pCloseTag\ t)) \\
&\quad <\!|\!> \quad Content' <\$\!> pString
\end{aligned}
$$

QUESTION 8: Heaps

```
checkHeap Empty = True
checkHeap h@(Top a _ _)
 = checkHeap' a h
   where checkHeap' f (Top c l r) =      f ⩾ c
                                       ∧ checkHeap' c l
                                       ∧ checkHeap' c r
         checkHeap' _ Empty       = True

mergeHeaps Empty r       = r
mergeHeaps l       Empty = l
mergeHeaps l@(Top lv ll lr) r@(Top rv rl rr)
  | lv ⩾ rv = Top lv (mergeHeaps ll lr) r
  | lv ⩽ rv = Top rv l (mergeHeaps rl rr)

enumHeap Empty       = []
enumHeap (Top v l r) = v : enumHeap (mergeHeaps l r)

v = enumHeap . foldr mergeHeaps Empty $ [Top x Empty Empty | x ← [1 .. 10]]
```