Tentamen Functioneel Programmeren—27 oktober 2009

De nagekeken tentamens zijn in te zien op kamer BB 366.

Opmerkingen:

- Schrijf **netjes** en duidelijk, met zwarte of blauwe pen.
- Zet op het eerste blad alle gegevens als naam, etc., en het totaal aantal ingeleverde bladen, en nummer de ingeleverde bladen.
- Lees de opgaven eerst goed door.
- Houd je programma's kort en helder, mede door verstandig gebruik te maken van standaardfuncties uit het boek (in het bijzonder uit het gedeelte over lijsten) en/of door listcomprehension.
- Motiveer je antwoorden.

1. a) Definieer een functie `matches :: Int -> [Int] -> [Int]` die alle voorkomens van de `Int` parameter uit de argument lijst oplevert.
   Dus `matches 2 [3,2,1,2,2,5,2] = [2,2,2,2]`.
   b) Gebruik nu de compositie operator om een functie te maken die het aantal voorkomens van de parameter in de lijst oplevert.

2. Gegeven is de implementatie van `reverse`:

   ```
   reverse [] = []
   reverse (x:xs) = reverse xs ++ [x]
   ```

   Bewijs met volledige inductie over alle eindige lijsten `xs` dat

   ```
   foldr (+) 0 (reverse xs) = foldr (+) 0 xs
   ```

   Hiervoor heb je wel een hulpbewijs nodig, over alle eindige lijsten `ys`.

   ```
   foldr (+) 0 (ys++[x]) = (foldr (+) 0 ys) + x
   ```

   Bewijs dit dus eerst!

   Let op een heldere presentatie van je bewijs.

3. Een getal n noemen we een *perfect* getal indien zijn delers (zonder n zelf uiteraard) opgeteld het getal n opleveren. Schrijf een functie die, gegeven parameter m, alle perfecte getallen $\leq m$ berekent.

1

4. Even opfrissen:

```
curry :: ((a,b) -> c) -> (a -> b -> c)
curry g x y = g (x,y)

uncurry :: (a -> b -> c) -> ((a,b) -> c)
uncurry f (x,y) = f x y
```

a) Laat zien wat het type is van `uncurry map`.
b) Laat zien waarom `curry uncurry` niet door de typering van Haskell heen komt.

5. Een **binary searchtree** is een boom met geordende elementen

```
data BsTree a = Nil | Node a (BsTree a) (BsTree a)
```

We kunnen een abstract data type maken middels een `module BsTree`:

```
module BsTree
  (BsTree,          -- constructor
   nil,             -- BsTree a
   isNil,           -- BsTree a -> Bool
   isNode,          -- BsTree a -> Bool
   leftSub,         -- BsTree a -> BsTree a
   rightSub,        -- BsTree a -> BsTree a
   insTree,         -- Ord a => a -> BsTree a -> BsTree a
   delTree          -- Ord a => a -> BsTree a -> BsTree a
  } where ...
```

a) Geef de implementatie van `insTree`.
b) Geef de implementatie van `delTree`. Voor de laatste heb je waarschijnlijk hulpfuncties nodig.

6. Onder een **taal** verstaan we in deze opgave: een verzameling van eindige strings. In het onderhavige verband zijn parsers voor talen bepaalde soorten Haskell-functies, zoals die aan de orde geweest zijn in de FP-stof.

   (a) Zij L een taal. Wat is het Haskell-type van een parser voor L, wanneer gewerkt wordt met een type T als type van representaties van de elementen van L. (Aanwijzing. Het gevraagde type is Parse Char T. Schrijf dit expliciet uit.)

2

(b) Beschouw nu in het bijzonder: de taal L bestaande uit alle eindige strings van cijfers. Construeer in Haskell een parser voor L. Neem daarbij als representatie van elke willekeurige string `xs` uit L: het paar `(n, l)` met `l` de lengte van `xs` en `n` de waarde van `xs` (gezien als decimale representatie van een getal, na weglating van overbodige 0-en aan het begin), waarbij het zo is dat de lege string per definitie waarde 0 heeft. Je mag daarbij gebruik maken van de elementaire parse functies uit §17.5 uit het boek.