Departement Informatica en Informatiekunde, Faculteit Bètawetenschappen, UU.
In elektronische vorm beschikbaar gemaakt door de TBC van A–Eskwadraat.
Het college INFOFP werd in 2007/2008 gegeven door Doaitse Swierstra.

Functioneel Programmeren (INFOFP)
18 april 2008

Opgave 1

Wat is het type van `concat.concat`?

a) `[[[a]]] -> [a]`

b) `[a] -> [[a]] -> [a]`

c) `[[a]] -> [[a]] -> [[a]]`

d) geen van a) t/m c)

Opgave 2

Welke van de volgende uitspraken is waar?

a) De functie `foldl` gebruikt altijd minder ruimte dan `foldr`.

b) De functie `foldr` kan al een stukje van een resultaat opleveren als het lijstargument nog niet helemaal bekend is.

c) Zoeken in een zoekboom is altijd sneller dan zoeken in een lijst.

d) Omdat lijstconcatenatie associatief is, moet functiecompositie dat ook zijn.

Opgave 3

Wat is het correcte type van de functie `(:=)` uit wxHaskell?

a) `a -> Attr w a -> Prop w`

b) `Attr w a -> a -> Prop w`

c) `Attr w -> a -> Prop w a`

d) `Prop w a -> a -> Attr w a`

Opgave 4

Welke van de volgende uitspraken is waar m.b.t. de ontleedmethodes zoals behandeld op college:

a) Een parser levert *altijd* een lijst op.

b) De combinator `<$>` is eenvoudig uit te drukken m.b.v. `pSucceed` en `<|>`.

c) De operator `<*>` is rechts-associatief.

d) De prioriteit van `<*>` is lager dan die van `<|>`.

Opgave 5: Paren

a) Schrijf een functie `splits` die een lijst op alle mogelijke manieren splitst in een element en de rest van de lijst. Voorbeeld:

```
demo >> splits [1,2,3,4]
[(1,[2,3,4]),(2,[1,3,4]),(3,[1,2,4]),(4,[1,2,3])]
```

b) Schrijf een functie `pairs :: (a -> a -> Bool) -> [a] -> [[(a, a)]]` die alle mogelijke manieren oplevert waarin alle elementen uit het argument paarsgewijs aan elkaar gekoppeld worden, mits ze bij elkaar passen. Of twee elementen bij elkaar passen wordt bepaald door het eerste argument; dus

```
demo>> pairs (\ x y -> abs (x - y) <6) [1,2,3,6,8,7]
[[(1,2),(3,6),(8,7)],[(1,2),(3,8),(6,7)],[(1,2),(3,7),(6,8)],[(1,3),(2,6),(8,7)],
[(1,3),(2,7),(6,8)],[(1,6),(2,3),(8,7)],[(1,6),(2,7),(3,8)]
]
```

Gebruik de list of successes methode. In onze oplossing bestaat de definitie uit een lijstcomprehensie.

Opgave 6

Bewijs met inductie dat `length . concat` $\equiv$ `sum . map length`.

Opgave 7: Finger Trees

We willen van een `FingerTree` tellen hoeveel waarden er in opgeslagen zijn. Een manier om dat te doen is als volgt:

```
data Node a = Node2 a a
            | Node3 a a a
type Digit a = [a]
data FingerTree a = Empty
                  | Single a
                  | Deep (Digit a) (FingerTree (Node a)) (Digit a)
count :: FingerTree a -> Int
count = length . toList
```

We kunnen echter ook proberen dit met behulp van het klassesysteem te doen:

```
class Countable t where
count :: t -> Int

instance Countable Int where
count i = 1

countint :: FingerTree Int -> Int
countint = count
...
```

Geef de instanties die nodig zijn om het programma te completeren.