FACULTEIT EWI
Datum: 26 juni 2015

Tentamen Programming Paradigms – Functional Programming

29 juni 2015
13:45 – 16:45 uur

Opmerkingen vooraf:
- U mag het dictaat *Functional Programming – an overview* bij dit tentamen gebruiken, verder niets.
- U mag alleen gebruik maken van standaard Haskell libraries: *Prelude*, *Data.List*, *Data.Char*, *Data.Maybe*.
- **Geef bij *elke* functie die u definieert het type.**
- Beoordeling: er zijn drie opgaven die allemaal even zwaar wegen.
- De elegantie van de oplossing zal ook een rol spelen, dus gebruik geen onnodige hulpfuncties en zo weinig mogelijk tellertjes en indices.
- Succes!

Opgave 1.

a. Een lijst *xs* heet een *jolly jumper* als de absolute waarde van de verschillen tussen alle tweetallen opeenvolgende elementen precies alle getallen in de range $1, \ldots, n-1$ aannemen (waarbij n de lengte van de lijst *xs* is). Bijvoorbeeld: $[1, 5, 8, 7, 9]$ is een jolly jumper, want de lijst van absolute waardes van de opeenvolgende verschillen is $[4, 3, 1, 2]$, en deze lijst bevat alle getallen in de range 1–4.

Schrijf twee varianten van een functie die test of een gegeven lijst een jolly jumper is: één maal met recursie, één maal met hogere orde functies.

b. Neem aan dat alle elementen van een lijst verschillend zijn. Schrijf een functie *powerList* die de lijst van alle mogelijke sublijsten van de gegeven lijst bepaalt, zodanig dat alle sublijsten verschillend zijn wat betreft de elementen die er in zitten (dus bovenstaande functie *setEqual* moet *False* opleveren bij elk tweetal sublijsten).

c. Schrijf een functie de bepaalt of een getal (van type *Int*) een palindroom is, dus van links naar rechts dezelfde cijferreeks oplevert als van rechts naar links. Bijvoorbeeld: 13531 is een palindroom.

Opmerking: u mag *geen* gebruik maken van een tussenrepresentatie met strings of characters, zoals bijvoorbeeld door de functie *show* wordt opgeleverd.

d. De *driehoek van Pascal* kent vele toepassingen. De eerste vijf regels zien er als volgt uit:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

Elke regel wordt gemaakt door de aangrenzende getallen in de regel erboven bij elkaar op te tellen, aangevuld met een 1 aan de beide uiteinden. Dus de zesde regel wordt als volgt uit de vijfde regel bepaald:

```
1+4   4+6   6+4   4+1
```

aangevuld met een 1 aan het begin en eind. Het resultaat daarvan is dus:

```
1 5 10 10 5 1
```

Schrijf een functie die de eerste n regels van de driehoek van Pascal oplevert. Het resultaat moet de vorm hebben van een lijst van lijsten.

Opgave 2.

a. Definieer een type *Tree* van binaire bomen die aan elke node en aan elk blad een getal bevat.

b. Schrijf een functie *count* die telt hoe vaak een gegeven getal in een gegeven boom voorkomt.

c. Schrijf een functie *zoekMax* die van een gegeven boom het grootste getal oplevert (dat grootste getal kan eventueel vaker dan één keer in de boom voorkomen).

d. De *mediaan* van een verzameling getallen is het middelste getal qua grootte, dat wil zeggen: de helft van de getallen is kleiner (of gelijk) aan de mediaan, en de andere helft is groter (of gelijk). Als de verzameling een even aantal getallen bevat, mag u zelf kiezen welke van de twee mogelijkheden u neemt als mediaan.

Schrijf een functie *minstensMed* die alle getallen in de boom die kleiner zijn dan de mediaan vervangt door nul.

e. Een *pad* naar een node in een boom is een string die bestaat uit l en r, respectievelijk voor "links" en "rechts", en die aangeeft hoe je vanuit de root van de boom in die node komt.

Schrijf een functie *subboom* die die subboom uit een boom oplevert waarvan de root wordt aangewezen door een pad. Als het pad te lang is, moet een foutmelding worden gegeven.

Opgave 3.
Gegeven zijn de types

$$\begin{array}{lcl} \textbf{type } \mathit{Node} & = & \mathit{Int} \\ \textbf{type } \mathit{Graph} & = & [(\mathit{Node},\ [\mathit{Node}])] \end{array}$$

Het type *Graph* is een lijst van geordende paren (n, ms), waarbij node n uitgaande edges heeft naar precies alle nodes in de lijst ms. Een graaf is volgens dit type dus een *gerichte* graaf.

a. Schrijf een functie *bereikbaar* die test of een node m bereikbaar is vanuit node n in nul of meer stappen. Daarbij mogen edges alleen in de juiste richting worden doorlopen.

b. Schrijf een functie *verbonden* die nagaat of twee nodes n en n' verbonden zijn, dat wil zeggen dat er een rij nodes $m_0, m_1, \ldots, m_k$ is met $m_0{=}n$ en $m_k{=}n'$, zodanig, dat alle opeenvolgende paren m_i, m_{i+1} is de rij buren zijn. Anders gezegd, n en n' zijn verbonden als er n' "bereikbaar" is vanuit n zonder op de richting van de edges te letten.

c. Schrijf een functie die bepaalt of een graaf *samenhangend* is, dat wil zeggen dat ieder tweetal nodes met elkaar verbonden zijn.

d. Een edge is een *brug* als na verwijdering van die edge de graaf "uiteenvalt", d.w.z. meer subgrafen bevat die niet met elkaar verbonden zijn dan voordat die [redacted] edge werd verwijderd. Schrijf een functie *bevatBrug* die test of een graaf een brug bevat.