FACULTEIT EWI
Datum: 20 oktober 2014

Tentamen Functioneel Programmeren (192112051)

28 oktober 2014
8:45 – 11:45 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.
- **Geef bij *elke* functie die u definieert het type.**
- Beoordeling: er zijn vier 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. Schrijf een functie *add35* die alle 3- en 5-vouden kleiner dan een getal n bepaalt en optelt. Als een getal tegelijk 3- en 5-voud is, moet het maar één keer meegeteld worden. Bijvoorbeeld voor n=20 is de som van alle 3- en 5-vouden:

$$3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 = 78$$

Generaliseer uw definitie voor een willekeurige lijst ks van getallen k waarvan de veelvouden (kleiner dan een gegeven getal n) meegeteld moeten worden.

b. Schrijf een functie die bepaalt op welke manieren een bedrag met munten van 1, 2, 5, 10, 20, 50 cent kan worden gemaakt. Bijvoorbeeld, een bedrag van 4 cent kan worden samengesteld als 4x1 cent, of als 2x1 cent en 1x2 cent, of als 2x2 cent:

[[1,1,1,1], [1,1,2], [2,2]]

De volgorde binnen een combinatie doet er niet toe: de rijtjes [1,1,2], [1,2,1], [2,1,1] zijn equivalent, en er moet slechts één worden opgeleverd.

c. 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. Schrijf een functie *bijnaGelijk* die test of twee lijsten "bijna" gelijk zijn. Dat wil zeggen dat beide lijsten op maximaal één positie mogen verschillen. De lijsten moeten dus dezelfde lengte hebben, en op overeenkomstige posities (op hoogstens één positie na) moeten dezelfde elementen staan.

b. Schrijf een functie *subset* die test of alle elementen van een gegeven lijst voorkomen in een andere lijst. De volgorde en het aantal voorkomens doet hierbij niet ter zake.

c. Neem aan dat alle elementen van een lijst verschillend zijn. Schrijf een functie `powerset` 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, maar waarbij de volgorde van de elementen er niet toe doet.

Opgave 3.

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

b. 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).

c. Schrijf een functie die een histogram maakt van de getallen in de boom. Dat wil zeggen, de functie moet een gesorteerde lijst van tupels (a, n) opleveren voor ieder getal a in de boom, waarbij n het aantal keren is dat a in de boom voorkomt.

d. Een boom van type *Tree* is gesorteerd, als voor elke node in de boom geldt dat alle getallen in de linker subboom bij die node kleiner of gelijk zijn dan het getal aan die node zelf, en alle getallen in de rechter subboom groter (dus alle getallen in een gesorteerde boom zijn verschillend).

Schrijf een functie *sorted* die test of een boom gesorteerd is of niet.

Opgave 4.

a. Definieer een datatype *Graph* voor ongerichte grafen, en waarbij de edges geen gewichten bevatten. De nodes moeten worden aangegeven door labels (u mag zelf een type uit de class *Eq* kiezen voor labels).

b. Laat *ns* de lijst van een deel van de nodes uit en graaf zijn. Schrijf een functie *cut* die een graaf in twee subgrafen verdeelt waarvan de nodes van de ene subgraaf precies de nodes bevat die in *ns* zitten, en de andere subgraaf de resterende nodes. De functie *cut* moet het minimale aantal edges verwijderen zodat de beide subgrafen niet meer met elkaar verbonden zijn.

c. Schrijf een functie die bepaalt of een graaf *samenhangend* is, dat wil zeggen dat iedere node vanuit iedere andere node bereikbaar is in een aantal stappen.

d. Een node is een *knoop* als na verwijdering van die node (samen met de met die node verboden edges) de graaf "uiteenvalt", d.w.z. meer subgrafen bevat die niet met elkaar verbonden zijn dan voordat die node werd verwijderd. Schrijf een functie *bevatKnoop* die test of een graaf een knoop bevat.