FACULTEIT EWI
Datum: 25 oktober 2010

Tentamen Functioneel Programmeren (211205)

27 oktober 2010
8.45 – 12.15 uur

Opmerkingen vooraf:
- U mag het dictaat bij dit tentamen gebruiken, verder niets.
- **Geef bij *elke* functie die u definieert het type.**
- Beoordeling: er zijn vier opgaven, de zwaarte is bij elke opgave aangegeven.
- Succes!

Opgave 1 (25 punten). Een *verzameling* is een lijst waarvan alle elementen verschillend zijn. De volgorde waarin de elementen voorkomen doet niet ter zake.

a. Schrijf een functie `isSet` die test of een lijst een verzameling is.

b. Schrijf een functie `setEqual` die test of twee verzamelingen gelijk zijn.

c. De *vereniging* van twee verzamelingen `xs` en `ys` is de verzameling die alle elementen bevat die in `xs` en/of `ys` zitten. Schrijf een functie `union` die de vereniging van twee verzamelingen bepaalt.

d. Een verzameling `xs` is een *deelverzameling* van een verzameling `ys` als alle elementen van `xs` ook in `ys` zitten. Schrijf een functie `subsets` die de verzameling van alle deelverzamelingen van een gegeven verzameling oplevert.

Opgave 2 (15 punten). Een matrix is een lijst van lijsten van gelijke lengte, waarbij elke lijst een rij in de matrix is. Schrijf een functie `kolomtotalen` die de lijst van totalen van elke *kolom* in de matrix oplevert. Bijvoorbeeld, de uitkomst van deze functie voor de matrix:

$$\begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{bmatrix}$$

is de lijst:

$$\begin{bmatrix} 63, & 66, & 69, & 72 \end{bmatrix}$$

Opgave 3 (30 punten). Gegeven is het volgende type voor expressies:

```
opA   ::= Add | Mul | ...

opB   ::= EQ | GT | LT | ...

exprA ::= Num num
        | OpA opA exprA exprA
        | If exprB exprA exprA

exprB ::= OpB opB exprA exprA
```

Het type `opA` ("A" voor "Arithmetical") bevat rekenkundige operaties *optelling*, *vermenigivuldiging*, etc. Het type `opB` ("B" voor "Boolean") bevat operaties voor *gelijkheid*, *groter-dan*, *kleiner-dan*, etc. U hoeft onderstaande deelopgaven alleen te beantwoorden voor de hierboven genoemde operaties.

Rekenkundige expressies kunnen een enkele constante (bijv. `Num 3`) zijn, of een rekenkundige samenvoeging van twee expressies, bijvoorbeeld

```
OpA Add (Num 3) (Num 5)
```

of een "if-then-else"-expressie (die een boolean expressie bevat), bijvoorbeeld

```
If (OpB LT (Num 3) (Num 5)) (Num 1) (Num 0)
```

De laatste expressie representeert de volgende expressie

if 3<5 *then* 1 *else* 0

in "standaardnotatie".

a. Schrijf functies `evalA`, `evalB` die de waarde van rekenkundige en boolean expressies uitrekenen.

b. Schrijf een *generic* functie `toString` die rekenkundige en boolean expressies omzet in standaardnotatie.

c. Beschouw de volgende eigenschappen van rekenkundige expressies:

associativiteit: $(a + b) + c = a + (b + c)$
$(a \times b) \times c = a \times (b \times c)$

distributiviteit: $a \times (b + c) = (a \times b) + (a \times c)$

Een rekenkundige expressie is in *normaalvorm* als alle haakjes zoveel mogelijk naar rechts zijn verschoven en alle vermenigvuldigingen zoveel mogelijk "naar binnen" zijn geschoven. Preciezer gezegd: een expressie is in normaalvorm als bovenstaande eigenschappen niet meer van links naar rechts kunnen worden toegepast. Merk op: a, b, c kunnen zelf ook weer samengestelde expressies zijn die nog in normaalvorm moeten worden gebracht).

Schrijf een functie `toNF` die rekenkundige expressies omzet naar normaalvorm ("NF" staat voor "Normal Form").

Opgave 4 (30 punten). Deze opgave gaat over gerichte grafen, waarvan de nodes aangegeven zijn door natuurlijke getallen (elke node uiteraard door een ander getal). Gegeven zijn de types

```
node  == num
graph == [ (node,[node]) ]
```

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`.

a. Schrijf een functie `bereikbaar` die, gegeven een gerichte graaf van type `graph` en een startnode, de lijst van alle nodes oplevert die vanuit die startnode bereikbaar zijn. Daarbij mogen edges alleen in de goede richting worden doorlopen.

b. De *ingraad* van een node is het aantal binnenkomende edges van die node. Schrijf een functie `ingraad` die de ingraad van een node bepaalt.

c. Een *cycle* $[a_0, a_1, \ldots, a_{n-1}]$ is een lijst van nodes zodanig dat twee opeenvolgende nodes steeds door een edge verbonden zijn (in de goeie richting), en dat de laatste node weer (ook in de goeie richting) door een edge met de eerste node is verbonden. Bovendien mag iedere node slechts één keer in de cykel voorkomen.

Schrijf een functie `isCycle` die test of een lijst van nodes een cycle vormt.