Hertentamen Functioneel Programmeren—31 januari 2008

De nagekeken tentamens zijn in te zien bij de docent, J.H. Jongejan, Bernoulliborg kamer 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. (10 punten)

   a) Geef het type en de implementatie van `curry`.
   b) Geef het type van `map curry`

2. (15 punten)

   Gegeven is de functie `flatten :: [[a]] -> [a]` met de volgende eigenschappen

```
flatten []       = []                (* fl.1 *)
flatten (xs:xss) = xs ++ flatten xss (* fl.2 *)
```

   Bewijs met volledige inductie over alle eindige lijsten van eindige lijsten `xss` dat

```
(flatten xss) ++ (flatten yss) = flatten (xss ++ yss)
```

3. (5 punten)

   Gegeven is de volgende Java methode

```
int countZ (int[] a) {
  int i=0, n=0;
  while (i < a.length) {
    if (a[i] == 0) n++;
    i++;
  }
  return n;
}
```

Geef type en implementatie van een Haskell functie, die dezelfde berekening uitvoert als bovenstaande Java methode.

4. (20 punten)

Een **set** representeren we door middel van een geordende lijst elementen, waarbij ieder element hoogstens eenmaal voorkomt (dus een klassieke wiskundige verzameling):

```
newtype Set a = S [a]
```

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

```
module Set
  (Set,              -- constructor
   empty,            -- Set a
   singular,         -- a -> Set a
   member,           -- Eq a => a -> Set a -> Bool
   union, inter      -- Ord a => Set a -> Set a -> Set a
   subset,           -- Ord a => Set a -> Set a -> Bool
   makeSet,          -- Ord a => [a] -> Set a
   mapSet,           -- Ord b => (a ->b) -> Set a -> Set b
   card              -- Set a -> Int
  } where ...
```

a) Geef de implementatie van **singular**
b) Geef de implementatie van **union**
c) Geef de implementatie van **makeSet**.
d) Geef de implementatie van **mapSet**.

5. (15 punten)

Implementeer een Haskell functie `perms :: Int ->[[Int]]`, zodanig dat `perms n` ($n \geq 1$) alle permutaties van `1..n` oplevert, die starten met een even getal.

6. (25 punten)

Gegeven is een grammatica voor prefix expressies:

```
Pexp -> int | '+' Pexp Pexp | '*' Pexp Pexp
```

a) Geef het volledig type: `type Parse a b = [a] -> ???`
b) Schrijf een parser die een invoerstring kan parsen die aan deze grammaticaregels voldoet. Gebruik daarbij de primitieve parsers

```
succeed, token, spot, alt, build, >*>, list
```

c) Gegeven is het Haskell datatype voor postfix bomen:

```
data Ptree = Pdig Dig | Pnode Ptree Ptree Op
data Dig   = '0' | .. | '9'
data Op    = Add | Mul
```

Breid de parser uit b) uit, zodat zo'n postfix boom wordt opgebouwd tijdens het parsen.
d) Geef de implementatie van een functie `eval :: Ptree -> Int`, die zo'n postfix boom uitrekent.