Hertentamen Functioneel Programmeren—8 februari 2011

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)

   Geef het type en de implementatie van `foldr1`.

2. (20 punten)

   a) Geef het type en de implementatie van `zip`.
   b) Gegeven is de volgende definitie van `unzip`:

```
unzip :: [(a,b)] -> ([a],[b])
unzip [] = ([],[])
unzip ((a,b):abs) = (a:as,b:bs)
      where (as,bs) = unzip abs
```

   Bewijs met volledige inductie over alle eindige lijsten `xs` dat

```
zip (fst (unzip xs)) (snd (unzip xs)) = xs
```

3. (15 punten)

   Even opfrissen:

```
curry :: ((a,b) -> c) -> (a -> b -> c)

uncurry :: (a -> b -> c) -> ((a,b) -> c)
```

   a) Laat zien waarom `curry uncurry` niet door de typering van Haskell heen komt.
   b) Wat is het type van `uncurry zip`?

4. (15 punten)

Een **set** is een geordende lijst elementen, waarbij ieder element hoogstens eenmaal voorkomt (dus een klassieke wiskundige verzameling).

```
data 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
   subset,           -- Ord a => Set a -> Set a -> Bool
   makeSet,          -- Ord a => [a] -> Set a
   mapSet            -- Ord b => (a ->b) -> Set a -> Set b
  } where ...
```

a) Geef de implementatie van `singular`.
b) Geef de implementatie van `subset`.
c) Geef de implementatie van `makeSet`.
d) Geef de implementatie van `mapSet`.

5. (15 punten)

Gegeven is dat de invoer voldoet aan de grammaticaregels:

```
   E  = ('I' D | 'C' L) E'
   E' = E |
-- 2e alternatief is leeg!
   D  = '0'|'1'|..|'9'
   L  = 'a'|'b'|..|'z'
```

a) Geef een data definitie Exp om E's te representeren.
b) Bouw een parser `pExp :: Parse Char Exp`, die de input omzet naar een `Exp`. Je mag hierbij gebruik maken van:

```
type Parse a b = [a] -> [(b,[a])]

succeed :: b -> Parse a b
spot    :: (a -> Bool) -> Parse a a
token t  = spot (==t)
alt     :: Parse a b -> Parse a b -> Parse a b
build   :: Parse a b -> (b -> c) -> Parse a c
(>*>)   :: Parse a b -> Parse a c -> Parse a (b,c)
```


2