FP 2008-2009, Eindtoets
15 april 2009, 13.00-16.00

Het tentamen bestaat uit 4 multiplechoicevragen (1 punt elk) en 3 open vragen (2 punten elk).
Voor elk syntactisch verkeerd patroon in de ingeleverde prpogrammacode wordt 0.5 punt afgetrokken; dit zijn dure punten, dus kijk je tentamen hierop nog een keer apart na voordat je inlevert.
Lever het aparte antwoordblad in. Vergeet niet je naam ook in te vullen.

1. Wat is het type van *map* . *foldr*?

   (a) $(a \to a \to a) \to [a] \to [[a] \to a]$
   (b) $(a \to a \to a) \to [b] \to [b \to a]$
   (c) $(b \to a \to a) \to [a] \to [[b] \to a]$
   (d) $(b \to a \to a) \to [b] \to [[a] \to a]$

   (c) De functie *foldr* heeft type $(b \to a \to a) \to a \to [b] \to a$. De expressie *map* . *foldr* is equivalent aan $\lambda f \to map\ (foldr\ f)$ en dat is weer gelijk aan $\lambda f \to (\lambda l \to map\ (foldr\ f)\ l)$. Als f nu, als eerste parameter van *foldr*, type $b \to a \to a$ heeft dat heeft *foldr f* type $a \to ([b] \to a)$. Deze waarde wordt losgelaten op de waarde l van type $[a]$ en levert dan een waarde van type $[([b] \to a)]$.

2. Welke van de volgende uitspraken is waar m.b.t. de behandelde Prolog interpretator?

   (a) De functie *unify* slaagt altijd als beide argumenttermen geen variabelen bevatten.
   (b) De functie *unify* faalt altijd als beide argumenttermen alleen variabelen bevatten.
   (c) De functie *unify* levert altijd een uitgebreidere substitutie op dan haar argumentsubstitutie, mits zij slaagt.
   (d) Als een unificatie slaagt worden de termen van de rechterkant van een regel altijd toegevoegd aan de verzameling nog op te lossen doelen.

   (d) Tegenvoorbeelden: (a) *Con* 3 unificeert niet met *Con* 5, (b) *Var x* unificeert met *Var y*, (c) de unificatie van *Con* 3 met *Con* 3 breidt de substitutie niet uit.

3. We kunnen de type constructor voor lijsten ([]) een instantie maken van de klasse *Monad*:

   ```
   instance Monad [] where
     ma >>= a2mb = concat (map a2mb ma)
     return a    = [a]
   ```

   Welk van de volgende expressie is nu gelijk aan $[f\ x\ y \mid x \leftarrow expr1, y \leftarrow expr2]$? Hint: herschrijf de **do**-notaties hieronder zonodig naar de vorm met expliciete $\gg=$ en *return* operaties.

   (a)
   ```
   do x <- expr1
      y <- expr2
      f x y
   ```

   (b)
   ```
   do x <- expr1
      y <- expr2
      return (f x y)
   ```

   (c)
   ```
   do y <- expr2
      x <- expr1
      return (f x y)
   ```

   (d)
   ```
   do return (f x y)
      where do x <- expr1
               y <- expr2
   ```

We ontsuikeren het tweede alternatief door de **do** uit te drukken in zijn onderliggende betekenis:

$$
\begin{aligned}
&\textbf{do } x \leftarrow \mathit{expr1}\\
&\quad\ y \leftarrow \mathit{expr2}\\
&\quad\ \mathit{return}\ (f\ x\ y)\\
&\mathit{expr1} \gg\!= (\lambda x \rightarrow \textbf{do } y \leftarrow \mathit{expr2}\\
&\qquad\qquad\qquad\qquad\ \mathit{return}\ (f\ x\ y)\\
&\qquad\quad)\\
&\mathit{expr1} \gg\!= (\lambda x \rightarrow \mathit{expr2} \gg\!= \lambda y \rightarrow \mathit{return}\ (f\ x\ y))\\
&\mathit{expr1} \gg\!= (\lambda x \rightarrow \mathit{expr2} \gg\!= \lambda y \rightarrow [f\ x\ y])\\
&\mathit{expr1} \gg\!= (\lambda x \rightarrow \mathit{concat}\ (\mathit{map}\ (\lambda y \rightarrow [f\ x\ y])\ \mathit{expr2})\\
&\mathit{concat}\ (\mathit{map}\ (\lambda x \rightarrow \mathit{concat}\ (\mathit{map}\ (\lambda y \rightarrow [f\ x\ y])\ \mathit{expr2}))\ \mathit{expr1})
\end{aligned}
$$

en dit is ook wat je krijgt als je de lijstdefinitie ontsuikert.

Alternatief (d) is helemaal geen correct Haskell, in (a) ontbreekt de *return*, en in (c) staan de x en de y verkeerd om.

4. Als we de GHCi gebruiken geeft de Haskell expression $2 + \mathit{True}$ als foutmelding?

```
No instance for (Num Bool)
  arising from use of ‘+’ at <interactive>:1:1
  ...
```

Hoe lossen we dit op?:

(a) We definiëren een functie *fromInteger* die *True* op een integer waarde afbeeldt.

(b) We definieren een functie (+) van type $\mathit{Integer} \rightarrow \mathit{Bool} \rightarrow \mathit{Integer}$.

(c) We definiëren een functie *fromInteger* van type $\mathit{Integer} \rightarrow \mathit{Bool}$.

(d) Geen van deze oplossingen.

Als we Bool een instantie maken van de klasse *Num* maken, dan komen daarmee o.a. de volgende twee functies beschikbaar:

$$
\begin{aligned}
(+) &:: \mathit{Bool} \rightarrow \mathit{Bool} \rightarrow \mathit{Bool}\\
\mathit{fromInteger} &:: \mathit{Int} \rightarrow \mathit{Bool}
\end{aligned}
$$

en in die context kan de gegeven expressie, die eigenlijk staat voor $\mathit{fromInteger}\ 2 + \mathit{True}$, uitgerekend worden. Dit is ook de hint die in de foutmelding gegeven wordt.

5. **Partities**
We noemen een lijst van lijsten p een partitie van een lijst l als geldt: $\mathit{concat}\ p \equiv l$. Een partitie bevat geen overbodige lege lijsten. Schijf m.b.v. *foldr* een functie $\mathit{parts} :: [a] \rightarrow [[[a]]]$ die alle verschillende partities van een lijst oplevert.

$$\mathit{parts} = \mathit{foldr}\ f\ [[]]\ \textbf{where}\ f\ x\ r = [(x : xs) : xss \mid (xs : xss) \leftarrow r] +\!\!+ [[x] : xss \mid xss \leftarrow r]$$

Elk element kan gebruikt worden om een nieuw stuk aan een partitie van de rest toe te voegen (tweede deel), en om aan het eerste stuk van een partitie van de rest toegevoegd worden (eerste deel).

6. **Inductiebewijs**
Bewijs met inductie dat $\mathit{sum}\ (\mathit{map}\ (+1)\ xs) = \mathit{length}\ xs + \mathit{sum}\ xs$.

Zie dictaat. Dit had de overgrote meerderheid goed!

7. **Ontleden**
We kunnen de functies *show* en *read* gebruiken om bijvoorbeeld een lijst af te drukken en weer in te lezen.

(a) Schijf nu zelf, m.b.v. de parsercombinatoren, een functie *readIntList* :: *String* → [*Int*] die het equivalent is van *read* voor lijsten van *Int*s. Je hoeft dus niet de hele input te consumeren. Je hoeft ook de functies voor sequentiële compositie (<∗>), keuze (<|>), voor de lege string (*pSucceed*) en voor een enkel symbool (*pSym*) niet zelf te definiëren. Andere hulpfuncties wel. Je mag ervan uitgaan dat de invoer geen spaties bevat. Ga even na dat je oplossing invoer zoals `"[]"`, `"[3]"` en `"[3,5,7]"` inderdaad aan kan.

(b) Geef nu de definitie van de instantie voor de klasse *Parseable* voor lijsten van een algemeen type; doe dit weer m.b.v. de parsercombinatoren. Dus vul de volgende code aan:

```
class Parseable a where
  parse :: String → [(a, String)]
instance... ⇒ Parseable [a] where
 ...
```

Bijvoorbeeld:

```
readIntList = readList pInt
  -- herken de vierkante haakjes en iets daartussen
readList a = pSym '[' *> pList a <* pSym ']'
pList a    = pSucceed []   -- het geval "[]"
             <|>
  -- en het eerste element van het niet lege geval
  -- gevolgd door een (mogelijk lege) rij komma-element paren
             (:) <$> a <*> pMany (pSym ',' *> a)
pMany p = pSucceed [] <|> (:) <$> p <*> pMany p
p (*>) q = flip const <$> p <*> q
p (<*) q =       const <$> p <*> q
```

Voor *pInt* zie dictaat. Afwezigheid hiervan is niet aangerekend. Merk op dat dit grote gelijknis toont met het inlzen van een rij argumenten voor een term, zoals in de Prolog interpretator.Ik heb voor de functie *readIntList* hier al gelijk maar een abstractie gebruikt (*readList*), zodat de volgende opgave nu een kort antwoord heeft:

```
instance Parseable a ⇒ Parseable [a] where
  parse = readList parse
```

Let op: de *parse* die aan *pList* meegegeven wordt is dus de *parse* die één enkel *a* element van de lijst van type [*a*] herkent.