Departement Informatica en Informatiekunde, Faculteit Bètawetenschappen, UU.
In elektronische vorm beschikbaar gemaakt door de TBC van A–Eskwadraat.
Het college INFOFP werd in 2008/2009 gegeven door Doaitse Swierstra.

Functioneel Programmeren (INFOFP)
15 april 2009

Het tentamen bestaat uit 4 multiplechoicevragen (1 punt elk) en 3 open vragen (2 punten elk).

Opgave 1

Wat is het type van `map . foldr`?

a) `(a -> a -> a) -> [a] -> [[a] -> a]`

b) `(a -> a -> a) -> [b] -> [b -> a]`

c) `(a -> a -> a) -> [a] -> [[b] -> a]`

d) `(a -> a -> a) -> [b] -> [[a] -> a]`

Opgave 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.

Opgave 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 | x <- expr1, y <- expr2]`? Hint: herschrijf de do-notaties hieronder zonodig naar de vorm met expliciete `>>=` en return operaties.

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

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

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

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

Opgave 4

Als we de GHCi gebruiken geeft de Haskell expression `2 + 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 definiëren een functie `(+)` van type `Integer -> Bool -> Integer`.

c) We definiëren een functie `fromInteger` van type `Integer -> Bool`.

d) Geen van deze oplossingen.

Opgave 5: Partities

We noemen een lijst van lijsten p een partitie van een lijst l als geldt: `concat p == l`. Een partitie bevat geen overbodige lege lijsten. Schijf m.b.v. `foldr` een functie `parts :: [a] -> [[[a]]]` die alle verschillende partities van een lijst oplevert.

Opgave 6: Inductiebewijs

Bewijs met inductie dat `sum (map (+1) xs) = length xs + sum xs`.

Opgave 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 Ints. 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
   ...
```