Tentamen Functioneel Programmeren, 21 november 2000

Tijdsduur 3 uur.

Houd de programma's kort door zoveel mogelijk standaardfuncties te gebruiken (uit de prelude van Hugs en het boek van Bird). Je hoeft geen definities te geven van: id, head, tail, take, drop, map, filter, and, or, zip, curry, uncurry. Geef van andere standaardfuncties wel steeds definities (gelijk of equivalent met de standaarddefinities).

Geef van elke hulpfunctie die je gebruikt een informele specificatie.

Het begrip stijgend betekent steeds strikt stijgend.

Opgave 1 (2 punt). Maak een functie

```
ontbind :: [Int] -> [[Int]]
```

zodanig dat `(ontbind xs)` de zo kort mogelijke lijst `yss` is van stijgende deellijsten van `xs` waarvoor `(concat yss)` gelijk is aan `xs`. Voorbeeld:

```
ontbind [1,7,5,5,7,8] = [[1,7],[5],[5,7,8]]
```

Opgave 2 (2 punt). De functie

```
inv :: (Int -> Int) -> Int -> Int
```

is zodanig dat `(inv f x)` altijd een getal y oplevert dat voldoet aan `f y` $\leq$ `x` $<$ `f(y+1)`, tenminste als de functie `f` voldoet aan $\lim_{x\to\infty} \mathtt{f}(x) = +\infty$ en $\lim_{x\to-\infty} \mathtt{f}(x) = -\infty$.

Gevraagd wordt een implementatie van `inv` met logaritmische complexiteit, dwz als voor alle $n \geq 2^k$ geldt dat $\mathtt{f}(-n) \leq \mathtt{x} < \mathtt{f}(n)$, dan dient `(inv f x)` berekend te worden in $\mathcal{O}(k)$ stappen. Je mag hierbij aannemen dat `(f x)` berekend wordt in één stap.

Opgave 3 (2 punt). De functie

```
genereer :: (Int -> [Int]) -> [Int] -> [Int]
```

is zodanig dat, als `xs` een stijgende lijst van getallen is en als `(f x)` voor elke `x` een stijgende lijst van getallen $>$ `x` is, dan is `(genereer f xs)` een stijgende lijst die `xs` als deellijst bevat en die met elk element `x` ook de lijst `(f x)` als deelllijst bevat, en die geen verdere elementen bevat. Voorbeeld:

```
genereer (\x->[2*x]) [1,7] = [1,2,4,7,8,14,16,28,32,56, enz
```

Implementeer `genereer`. De lijsten `xs` en `(f x)` moeten leeg, eindig, of oneindig kunnen zijn.

Opgave 4 (2 punt). Beschouw de definities

```
data Htree a = Nil | Node a (Htree a) (Htree a)
fl Nil = []
fl (Node x yh zh) = x: fl yh ++ fl zh
```

We beschouwen nu tevens een functie `flac` gespecificeerd door

```
flac us xh = fl xh ++ us
```

(a) Leid uit deze specificatie een definitie voor `flac` af die efficienter is dan de specificatie omdat het gebruik van de operator `++` eruit geëlimineerd is.

(b) We definieren `afl = flac []`. We schrijven $T(\mathtt{fl})(h)$ en $T(\mathtt{afl})(h)$ voor het aantal stappen nodig voor de berekening van `(fl xh)` of `(afl xh)` als `xh` een volledige (gebalanceerde) boom met hoogte h is. Leid recurrente betrekkingen voor deze twee functies af. Je hoeft deze recurrente betrekkingen niet op te lossen.

Z.O.Z.

Opgave 5 (2 punt). We beschouwen eindige lijsten als een abstract datatype `(List a)` met de operaties

```
nil :: List a
null :: List a -> Bool
cons :: a -> List a -> List a
head :: List a -> a
tail :: List a -> List a
```

en de axioma's

```
null nil = ...
null (cons x xl) = ...
head (cons x xl) = ...
tail (cons x xl) = ...
```

(a) Vul de rechter leden van de axioma's goed in.
(b) We representeren (implementeren) het abstracte datatype met

```
type FList a = (Int -> a, Int)
```

waarbij (f, n) de lijst representeert met lengte n en met $f(k)$ als k–de element.

Geef geschikte operaties `fnil`, `fnull`, `fcons`, `fhead`, `ftail` die de abstracte operatoren `nil`, `null`, `cons`, `head`, `tail` implementeren met type `(List a)` vervangen door `(FList a)`. Geef tevens een abstractiefunctie

```
abstr :: FList a -> List a
```