# Haskell: programmeren in een luie, puur functionele taal

Jan van Eijck

jve@cwi.nl

5 Talen Symposium, 12 juli 2010

**Samenvatting**

In deze mini-cursus laten we zien hoe je met eindige en oneindige lijsten kunt programmeren in de functionele programmeertaal Haskell.

## Korte Inhoud

- Een programma puzzle
- Puzzelen met steentjes
- Functies en functioneel programmeren
- Functies maken met lambda abstractie
- Eigenschappen van dingen en karakteristieke functies
- De ‘filter’ functie
- Oneindige lijsten
- Priemgetallen herkennen en genereren
- Opdrachten

## Programma-puzzel: Wat doet dit programma?

```
main = putStrLn (s ++ show s)
  where s = "main = putStrLn (s ++ show s) \n  where s = "
```

## Puzzelen met steentjes

In een vaas zitten 35 witte en 35 zwarte steentjes. Je gaat, zolang dat mogelijk is, als volgt te werk. Je haalt steeds twee steentjes uit de vaas.

- Als ze dezelfde kleur hebben stop je een zwart steentje terug in de vaas (er zijn voldoende extra zwarte steentjes),
- als ze verschillende kleur hebben stop je het witte steentje terug in de vaas.

Omdat er bij elke stap een steentje verwijderd wordt is er na 69 stappen nog maar één steentje over. Welke kleur heeft dat steentje? Waarom?

## Functies en functioneel programmeren

Functioneel programmeren is programmeren met functies. Een bekende functionele programmeertaal is Haskell, genoemd naar de logicus Haskell B. Curry.


http://www.haskell.org

Hugs is de implementatie van Haskell die we zullen gebruiken. Zie http://www.haskell.org/hugs.

## Functies en Typen-Declaraties

Een functie van een verzameling $A$ naar een verzameling $B$ is een voorschrift om elk element van $A$ te koppelen aan een element van $B$.

Notatie:

$$f : A \to B.$$

Bij functioneel programmeren heten de verzamelingen typen, en is de notatie als volgt:

```
f :: a -> b
```

Dit heet: de type-declaratie van het programma `f`.

De instructie voor de functie zelf is het programma voor `f`.

## Voorbeelden van Type-declaraties

Voorbeeld: als gehele getallen het type `Int` hebben, dan geldt:

optellen heeft het type `Int -> Int -> Int`.

met 1 vermeerderen heeft type `Int -> Int`.

kwadrateren heeft type `Int -> Int`.

Deze informatie kan worden gebruikt om te kijken of een programma welgetypeerd is.

Dit is een handige manier om vaak gemaakte slordigheidsfouten bij het programmeren te voorkomen.

## Een programma met type-declaratie

Voorbeelden uit de praktijk van het programmeren in Haskell.

```
kwadraat :: Int -> Int
kwadraat x = x * x
```

```
Main> kwadraat 7
49
Main> kwadraat (-3)
9
Main> kwadraat (kwadraat 7)
2401
Main> kwadraat (kwadraat (kwadraat 7))
5764801
```

## Het steentjes-programma in Haskell

Representeer een wit steentje als 0, een zwart steentje als 1. Een vaas met steentjes wordt nu een lijst van nullen en enen. Het type van zo'n lijst is `[Int]`. Een steentje trekken verandert een lijst in een lijst met een steentje minder. Type: `[Int] -> [Int]`.

```
trekSteentje :: [Int] -> [Int]
trekSteentje [x] = [x]
trekSteentje (0:0:xs) = trekSteentje (1:xs)
trekSteentje (1:1:xs) = trekSteentje (1:xs)
trekSteentje (0:1:xs) = trekSteentje (0:xs)
trekSteentje (1:0:xs) = trekSteentje (0:xs)
```

```
Main> trekSteentje [0,1,1,1,1,1,1,0,0,0,0,0,1,0]
[0]
```

## Lambda abstractie

Uit ‘Jan kust Heleen’ kunnen we door abstractie allerlei eigenschappen en relaties halen:

- ‘Heleen kussen’
- ‘door Jan gekust worden’
- ‘kussen’
- ‘gekust worden’

Dat gaat zo: We vervangen het element waarvan we abstraheren door een variabele, en we binden die variabele met een lambda operator.

Dus:

- '$\lambda x.x$ kust Heleen' staat voor 'Heleen kussen'.
- '$\lambda x$. Jan kust $x$' staat voor 'door Jan worden gekust'.
- '$\lambda y.x$ kust $y$' staat voor 'door x worden gekust'
- '$\lambda x \lambda y.x$ kust $y$' staat voor 'kussen'
- '$\lambda y \lambda x.x$ kust $y$' staat voor 'gekust worden'

## Lambda abstractie in Haskell

Een andere manier om de `kwadraat` functie te schrijven is met behulp van lambda abstractie. In Haskell staat `\ x` voor lambda abstractie over variabele `x`.

```
kwadr :: Int -> Int
kwadr = \ x -> x * x
```

De bedoeling is dat variabele `x` staat voor een getal, van type `Int`. Het resultaat, het gekwadrateerde getal, is ook van type `Int`. De functie `kwadr` is een functie die samen met een argument van type `Int` een waarde van type `Int` oplevert. Dat is precies wat de type-aanduiding `Int -> Int` wil zeggen.

## Twee manieren om lambda abstractie te representeren

Vergelijk nog eens:

```
kwadraat :: Int -> Int
kwadraat x = x * x
```

```
kwadr :: Int -> Int
kwadr = \ x -> x * x
```

## Eigenschappen van dingen, karakteristieke functies

De eigenschap 'deelbaar door drie' kan worden gerepresenteerd als een functie van getallen naar waarheidswaarden. De getallen $0, 3, 6, 9, \ldots$ worden door die functie afgebeeld op `True`, alle andere getallen op `False`.

Programmeurs noemen een waarheidswaarde een `Boolean`, naar de Britse logicus George Boole. Het type van `drievoud` is dus `Int -> Bool`. Hier zien we hoe de eigenschap `drievoud` wordt gedefinieerd met lambda abstractie:

```
drievoud :: Int -> Bool
drievoud = \ x -> (rem x 3 == 0)
```

```
Main> drievoud 5
False
Main> drievoud 12
True
```

## Werken met tekenrijtjes

Het type van tekens is `Char`. Rijtjes van tekens hebben type `[Char]`. Net zo hebben rijtjes van gehele getallen het type `[Int]`. Het lege rijtje wordt in Haskell aangeduid met `[]`.

Eigenschappen van rijtjes hebben dus type `[Char] -> Bool`. Hier is een eenvoudige eigenschap:

```
awoord :: [Char] -> Bool
awoord [] = False
awoord (x:xs) = (x == 'a') || (awoord xs)
```

```
Main> awoord "Jan"
True
Main> awoord "Heleen"
False
```

## Filtreren met behulp van eigenschappen

```
Main> filter drievoud [23,4,5,7,18,123]
[18,123]
Main> filter (\ x -> not (drievoud x)) [23,4,5,7,18,123]
[23,4,5,7]
Main> filter (not . drievoud) [23,4,5,7,18,123]
[23,4,5,7]
Main> filter awoord ["Jan", "kuste", "Heleen"]
["Jan"]
Main> filter (not . awoord) ["Jan", "kuste", "Heleen"]
["kuste","Heleen"]
```

## Het type van de 'filter' functie

De filter functie heeft het volgende type:

```
filter              :: (a -> Bool) -> [a] -> [a]
```

Hierbij staat a voor een willekeurig type. Je kunt immers zowel tekenrijtjes als rijtjes getallen als rijtjes van willekeurig wat filtreren, als je maar een eigenschap hebt van het goede type: `[Char] -> Bool` voor tekenrijtjes, `Int -> Bool` voor getallen, enzovoorts.

De combinatie van `filter` met een argument heeft zelf ook weer een type:

```
Main> :t filter drievoud
filter drievoud :: [Int] -> [Int]
Main> :t filter awoord
filter awoord :: [[Char]] -> [[Char]]
```

## Wat doen de volgende functies? Wat zijn hun types?

```
Main> map kwadr [1..10]
[1,4,9,16,25,36,49,64,81,100]
Main> map drievoud [1..10]
[False,False,True,False,False,True,False,False,True,False]
Main> map awoord ["Jan", "kuste", "Heleen"]
[True,False,False]
Main> all awoord ["Jan", "kuste", "Heleen"]
False
Main> any awoord ["Jan", "kuste", "Heleen"]
True
Main> any drievoud [1..10]
True
Main> or [False, True, False]
True
```

## Lijst-comprehensie

```
Main> [ 2*n | n <- [1..10] ]
[2,4,6,8,10,12,14,16,18,20]
Main> [ 2^n | n <- [1..10] ]
[2,4,8,16,32,64,128,256,512,1024]
Main>  [ x | x <- ['a' .. 'z'] ]
"abcdefghijklmnopqrstuvwxyz"
Main>  [ [x] | x <- ['a' .. 'h'] ]
["a","b","c","d","e","f","g","h"]
Main>  [ [[x]] | x <- ['a' .. 'e'] ]
[["a"],["b"],["c"],["d"],["e"]]
Main> [ [x,'y'] | x <-  ['a' .. 'h' ] ]
["ay","by","cy","dy","ey","fy","gy","hy"]
Main>  [ [x,y] | x <-  ['a' .. 'c' ], y <- ['d' .. 'f'] ]
["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

## Oneindige lijsten

Wat doet dit?

```
nullen = 0 : nullen
```

En dit?

```
nats = 0 : map (+1) nats
```

Dit heet: een luie lijst (lazy list).

## Priemgetallen herkennen

Een natuurlijk getal groter dan $1$ heet een priemgetal als het alleen deelbaar is door zichzelf en door 1.

Hier is een simpele test:

```
prime :: Integer -> Bool
prime n =
  n > 1 && all (\ x -> rem n x /= 0) [2..n-1]
```

## Priemgetallen genereren

Als je ze kunt herkennen kun je ze ook genereren:

```
primes :: [Integer]
primes = filter prime [0..]
```

## lets efficienter ...

```
prime' :: Integer -> Bool
prime' n =
  n > 1 && all (\ x -> rem n x /= 0) xs
  where xs = takeWhile (\ y -> y^2 <= n) [2..]

primes' :: [Integer]
primes' = filter prime' [0..]
```

## Wat is er mis met het volgende?

```
prime' :: Integer -> Bool
prime' n =
  n > 1 && all (\ x -> rem n x /= 0) xs
  where xs = takeWhile (\ y -> y^2 < n) [2..]

primes' :: [Integer]
primes' = filter prime' [0..]
```

## Droogt de stroom van priemgetallen ooit op . . . ?

Hoe weten we dat de stroom nooit opdroogt?

De kleinste deler van $Q = N! + 1$ moet een priemgetal zijn dat groter is dan $N$. (Waarom?)

| $N$ | $Q = N! + 1$ | kleinste deler van $Q$ |
|---|---|---|
| 2 | 3 | 3 |
| 3 | 7 | 7 |
| 4 | 25 | 5 |
| 5 | 121 | 11 |
| 6 | 721 | 7 |
| 7 | 5041 | 71 |
| 8 | 40321 | 61 |
| 9 | 362881 | 19 |
| 10 | 3628801 | 11 |
| 11 | 39916801 | 39916801 |
| 12 | 479001601 | 13 |
| 13 | 6227020801 | 83 |
| 14 | 87178291201 | 23 |
| 15 | 1307674368001 | 59 |

| $N$ | $Q = N! + 1$ | kleinste deler van $Q$ |
|---|---|---|
| 16 | 20922789888001 | 17 |
| 17 | 355687428096001 | 661 |
| 18 | 6402373705728001 | 19 |
| 19 | 121645100408832001 | 71 |
| 20 | 2432902008176640001 | 20639383 |
| 21 | 51090942171709440001 | 43 |
| 22 | 1124000727777607680001 | 23 |
| 23 | 25852016738884976640001 | 47 |
| 24 | 620448401733239439360001 | 811 |
| 25 | 15511210043330985984000001 | 401 |
| 26 | 403291461126605635584000001 | 1697 |
| 27 | 10888869450418352160768000001 | ? |

## Implementatie

Als we $N!$ en de functie voor het vinden van de kleinste deler van $Q$ die groter is dan $N$ kunnen uitwerken hebben we (in principe) een implementatie. Het eerste is gemakkelijk. Voor het tweede:

```
ldf :: Integer -> Integer -> Integer
ldf k n | rem n k == 0 = k
        | k^2 > n      = n
        | otherwise    = ldf (k+1) n

largerPrime :: Integer -> Integer
largerPrime n = ldf (n+1) (product [1..n] + 1)
```

## Iets efficienter . . .

Neem niet $N! + 1$, maar het product van alle priemgetallen kleiner dan of gelijk aan $N$, plus $1$.

De kleinste deler van dit getal moet een priemgetal groter dan $N$ zijn. (Waarom?)

```
largerPr :: Integer -> Integer
largerPr n = ldf (n+1) xs where
  xs = (product (takeWhile (<= n) primes') + 1)
```

Vergelijk:

```
Main> largerPrime 11
39916801
Main> largerPr 11
2311
```

## Een klassiek recept voor priemgetallen: de zeef

$$2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,$$
$$21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,$$
$$36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, \ldots$$

$$\boxed{2}, 3, \overline{4}, 5, \overline{6}, 7, \overline{8}, 9, \overline{10}, 11, \overline{12}, 13, \overline{14}, 15, \overline{16}, 17, \overline{18}, 19, \overline{20},$$
$$21, \overline{22}, 23, \overline{24}, 25, \overline{26}, 27, \overline{28}, 29, \overline{30}, 31, \overline{32}, 33, \overline{34}, 35,$$
$$\overline{36}, 37, \overline{38}, 39, \overline{40}, 41, \overline{42}, 43, \overline{44}, 45, \overline{46}, 47, \overline{48}, \ldots$$

$$\boxed{2}, \boxed{3}, \overline{4}, 5, \overline{6}, 7, \overline{8}, \overline{9}, \overline{10}, 11, \overline{12}, 13, \overline{14}, \overline{15}, \overline{16}, 17, \overline{18}, 19, \overline{20},$$
$$\overline{21}, \overline{22}, 23, \overline{24}, 25, \overline{26}, \overline{27}, \overline{28}, 29, \overline{30}, 31, \overline{32}, \overline{33}, \overline{34}, 35, \ldots$$
$$\overline{36}, 37, \overline{38}, \overline{39}, \overline{40}, 41, \overline{42}, 43, \overline{44}, \overline{45}, \overline{46}, 47, \overline{48}, \ldots$$

$$\boxed{2}, \boxed{3}, \overline{4}, \boxed{5}, \overline{6}, 7, \overline{8}, \overline{9}, \overline{10}, 11, \overline{12}, 13, \overline{14}, \overline{15}, \overline{16}, 17, \overline{18}, 19, \overline{20},$$
$$\overline{21}, \overline{22}, 23, \overline{24}, \overline{25}, \overline{26}, \overline{27}, \overline{28}, 29, \overline{30}, 31, \overline{32}, \overline{33}, \overline{34}, \overline{35},$$
$$\overline{36}, 37, \overline{38}, \overline{39}, \overline{40}, 41, \overline{42}, 43, \overline{44}, \overline{45}, \overline{46}, 47, \overline{48}, \ldots$$

## Implementatie als luie lijst

```
sieve :: [Integer] -> [Integer]
sieve (n:ns) =
  n : sieve (filter (\ k -> rem k n /= 0) ns)

sievePrimes :: [Integer]
sievePrimes = sieve [2..]
```

## Kunnen de paren van natuurlijke getallen worden opgesomd (afgeteld) ?

In 'woordenboek volgorde' kan niet: kijk maar:

$$(0,0),(0,1),(0,2),\ldots$$

Zo kom je niet eens aan $(1,0)$ toe ...

Maar het kan wel:

$$[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0),(0,4),$$
$$(1,3),(2,2),(3,1),(4,0),(0,5),(1,4),(2,3),(3,2),(4,1),(5,0),(0,6),$$
$$(1,5),(2,4),(3,3),(4,2),(5,1),(6,0),(0,7),(1,6),(2,5),(3,4),(4,3),$$
$$(5,2),(6,1),(7,0),(0,8),(1,7),(2,6),(3,5),(4,4),(5,3),(6,2),\ldots$$

Implementatie:

```
natpairs = [ (x, z-x) | z <- [0..], x <- [0..z] ]
```

Met een variatie hierop kun je ook alle drietallen van natuurlijke getallen opsommen:

```
nattriples = [ (x, y-x, z-y) | z <- [0..],
                               y <- [0..z],
                               x <- [0..y] ]
```

## Opdracht: Breuken opsommen

Kun je met behulp van `natpairs` laten zien dat ook de verzameling van alle positieve breuken aftelbaar is? Hoe?

Kun je ook een implementatie geven? Een breuk in eenvoudigste vorm is een paar $(n, m)$ (of $\frac{n}{m}$) met de eigenschap dat $m \neq 0$ en dat $m$ en $n$ geen gemeenschappelijke delers hebben (behalve $1$). Dit laatste kun je implementeren met `gcd n m == 1`. Hierbij staat `gcd` voor 'greatest common divisor' ('grootste gemene deler').

## Opdracht: Mersenne priemgetallen vinden

Een Mersenne-getal is een natuurlijk getal van de vorm $2^p - 1$, waarbij $p$ een priemgetal is. $2^2 - 1 = 3$, $2^3 - 1 = 7$, $2^5 - 1 = 31$ zijn voorbeelden van Mersenne getallen die priem zijn. De volgende functie genereert Mersenne priemgetallen:

```
mersenne :: [(Integer,Integer)]
mersenne = [ (p,2^p -1) | p <- primes',
                          prime' (2^p - 1) ]
```

Wat is het grootste Mersenne priemgetal dat je met deze functie kunt vinden?

## GIMPS

De grootste nu bekende priemgetallen zijn Mersenne getallen.

Er zijn maar 47 Mersenne priemgetallen bekend.

Het grootste daarvan, tevens het grootste nu bekende priemgetal, is $2^{43112609} - 1$. Dit priemgetal is gevonden in 2008. Dit is een getal van $12978189$ (decimale) cijfers.

Zie `http://www.mersenne.org/` voor meer info over The Great Internet Mersenne Prime Search (GIMPS).

Zie `http://primes.utm.edu/largest.html` voor informatie over de grootste nu bekende priemgetallen.

## Opdracht: priemparen vinden

Een priempaar is een paar $(p, p+2)$ van natuurlijke getallen met de eigenschap dat $p$ en $p+2$ allebei priemgetallen zijn. Voorbeelden zijn:

$$(5,7), (11,13), (17,19), (29,31), (41,43).$$

Kun je een functie schrijven die priemparen genereert?

## Opdracht: priem-drietallen vinden

Een priem-drietal is een drietal $(p, p+2, p+4)$, met $p, p+2, p+4$ alledrie priem. Het eerste priem drietal is $(3, 5, 7)$.

Bestaan er nog meer? Waarom wel/niet? Kun je ze genereren met de computer?

## Opdracht: een vermoeden weerleggen

Schrijf een Haskell programma dat kan worden gebruikt om de volgende bewering over priemgetallen te weerleggen:

Als $p_1, \ldots, p_k$ alle priemgetallen zijn die kleiner zijn dan $n$, dan is

$$(p_1 \times \cdots \times p_k) + 1$$

een priemgetal.

Je weerlegt dit vermoeden door een tegenvoorbeeld te geven. Schrijf een Haskell programma dat tegenvoorbeelden genereert.

## Opdracht: Pythagorische drietallen genereren

Als een metselaar of timmerman een rechte hoek moet uitzetten, bij voorbeeld voor het leggen van een fundering, maakt hij (of zij) een zogenaamde 'drie, vier, vijf steek': een driehoek met zijden van 3, 4 en 5 meter. De stelling van Pythagoras garandeert dan dat het een rechthoekige driehoek is. (Waarom?)

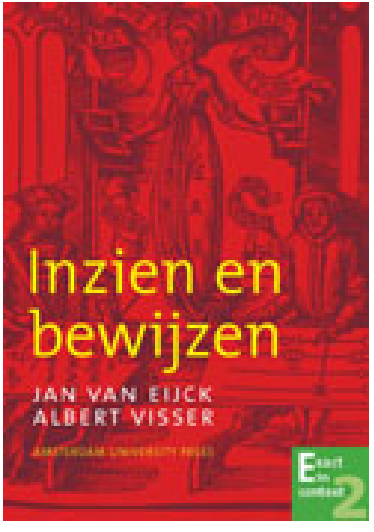
Een pythagorisch drietal is een drietal positieve natuurlijke getallen $(x, y, z)$ met de eigenschap dat $x^2 + y^2 = z^2$.

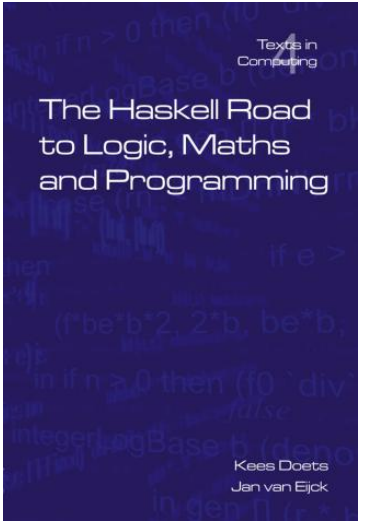
Implementeer een functie die Pythagorische drietallen genereert. De uitvoer moet zijn:

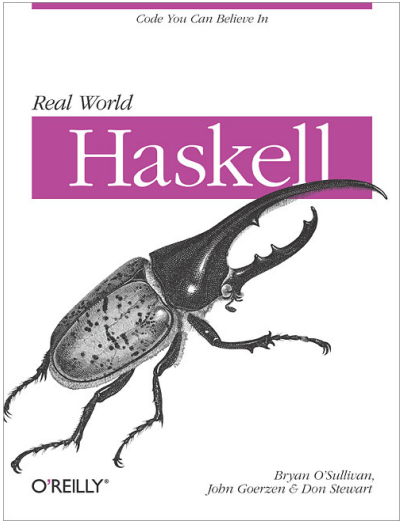
```
Main> pythTriples
[(3,4,5),(6,8,10),(5,12,13),(9,12,15),(8,15,17),...
```

Zijn er ook pythagorische drietallen $(x, y, z)$ met $x = y$? Waarom niet?

## Literatuur




http://www.cwi.nl/~jve/HR/ http://book.realworldhaskell.org/