TÖL304G
Forritunarmál
Vikublað 10
Weekly 10

Snorri Agnarsson

20. október 2024

Efnisyfirlit

1 Efni vikunnar — This Weeks Material

Við höldum áfram með Morpho, förum síðan í Haskell forritunarmálið og ræðum einnig *ruslasöfnun* þegar tími gefst til. Í Canvas má finna bækling, `gc.pdf`, um ruslasöfnun sem þið skuluð lesa.

We continue with Morpho and then proceed to the Haskell programming language and also discuss garbage collection when time allows. In Canvas there can be found a booklet, `gc.pdf`, about garbage collection. Read that booklet.

2 Haskell

Forritunarmálið Haskell er tagað fallsforritunarmál með latri gildun. Haskell hefur heimasíðu á vefnum[1] og sækja má Haskell þýðendur þaðan. Við munum nota Glasgow Haskell[2].

Sækja má Glasgow Haskell kerfið af vefnum[3] fyrir ýmis stýrikerfi. Þið munuð þurfa að afpakka kerfinu með skipuninni

```
tar xf <nafn sóttrar skrár>
```

þar sem þið setjið viðeigandi skráarnafn í skipunina. Þá verður til mappa sem inniheldur Glasgow Haskell kerfið.

The Haskell programming language is a typed functional programming language with lazy evaluation. Haskell has a home page on the web[4] and Haskell compilers can be downloaded from there. We will use Glasgow Haskell[5].

Glasgow Haskell can be downloaded from the web[6] for various operating systems. You will need to unpack the system using the command

```
tar xf <name of downloaded file>
```

where you put the appropriate file name in the command. Then a folder will be created that contains the Glasgow Haskell system.

3 Námsefni í Haskell — Material in Haskell

Í Haskell munum við sjá ýmislegt nýtt — In Haskell we will see various new things:

- Löt gildun — lazy evaluation, call by need.
- Nýr listaritháttur (list comprehension), sem minnir á hefðbundinn rithátt fyrir mengi í stærðfræðinni.

  A new list notation, list comprehension, that is reminiscent of traditional set notation in mathematics.
- Ýmsir mismunandi valkostir til að skilgreina jafngild föll (t.d. pattern matching).

  Various options for defining equivalent functions (for example pattern matching).

[1] http://www.haskell.org
[2] http://www.haskell.org/platform/
[3] https://www.haskell.org/ghc/download_ghc_9_0_1.html
[4] http://www.haskell.org
[5] http://www.haskell.org/platform/
[6] https://www.haskell.org/ghc/download_ghc_9_0_1.html

- Ný aðferð til að forrita með hliðarverkunum án þess að glata aðalkostum fallsforritunar (*einstæður*, monads).

  A new method for programming with side effects without losing the advantages of functional progamming (monads).

Af þessum atriðum eru það lata gildunin og listarithátturinn sem við munum leggja aðaláherslu á. Auk þess hefur Haskell tögun sem er mjög lík töguninni í CAML. Í Haskell eru reyndar nýjir möguleikar í töguninni sem CAML býður ekki upp á, en ekki verður lögð áhersla á þá þætti.

Of these new things it is the lazy evaluation and the list notation that we will most emphasize. Also, Haskell has typing that is very much like the typing in CAML. Haskell has, however, new possibilities in the typing that CAML does not offer, but we will not emphasize those aspects.

3.1 Vefgæði — Web Materials

Matthías Páll Gissurarson benti mér einu sinni á góða Haskell bók á vefnum, Learn You a Haskell for Great Good![7]. Einnig má finna vefsíðuna Try Haskell[8] þar sem prófa má Haskell án þess að setja neitt upp á tölvunni þinni.

Matthías Páll Gissurarson once pointed out to me a good Haskell book on the web, Learn You a Haskell for Great Good![9]. Also we can use the web page Try Haskell[10] where we can try out Haskell without installing anything on your computer.

3.2 Löt gildun — Lazy Evaluation

Í Haskell er það ófrávíkjanleg regla að viðföng falla eru ekki gilduð (evaluated) fyrr en nauðsyn krefur. Sama gildir um listaskilgreiningar, sem hefur þær afleiðingar að listar í Haskell eru keimlíkir straumum í Scheme.

In Haskell arguments to function are invariably not evaluated until necessary. The same hold for list definitions, which has the consequence that lists in Haskell are similar to streams in Scheme.

Eftirfarandi Haskell skilgreiningar nýta sér lötu gildunina í Haskell — The following Haskell definitions use the lazy evaluation in Haskell:

```
1    and :: [Bool] -> Bool
2    and [] = True
3    and (True : xs) = and xs
4    and (False : xs) = False
5
```

[7] http://learnyouahaskell.com

[8] http://tryhaskell.org

[9] http://learnyouahaskell.com

[10] http://tryhaskell.org

```
6     or :: [Bool] -> Bool
7     or [] = False
8     or (True : xs) = True
9     or (False : xs) = or xs
10
11    fib1 :: [Integer]
12    fib1 =
13      [1,1] ++ map (\(a,b)->a+b) (zip fib1 (tail fib1))
14
15    fib2 :: [Integer]
16    fib2 = [1,1] ++ zipWith (+) fib2 (tail fib2)
```

Lesið ykkur til um föllin map, zip og zipWith sem hér eru notuð.

Read about the functions map, zip and zipWith that are used here.

Lata gildunin í Haskell veldur stundum vandræðum. Íhugið til dæmis þetta fall — The lazy evaluation in Haskell may cause problems in some cases. Consider this function for example:

```
1     sum f n =
2       hjalp 0 0
3       where
4        hjalp s i =
5          if i>n then
6            s
7          else
8            hjalp (s+f(i+1)) (i+1)
```

Þegar við köllum á þetta fall með eftirfarandi Haskell segð — When we call this function with the following Haskell expression:

```
1     sum (\i->i^2) 10
```

þá reiknum við út $\sum_{i=1}^{10} i^2$ — Then we compute $\sum_{i=1}^{10} i^2$

En það gerist ekki með því að fyrst sé reiknuð talan 0, síðan talan $0+1^2$, síðan talan $0+1^2+2^2$, o.s.frv. Það sem gerist er að fyrst er smíðuð *segðin* 0, síðan *segðin* $0+1^2$, síðan *segðin* $0+1^2+2^2$, o.s.frv. Að lokum höfum við segðina $0+1^2+2^2+\ldots+10^2$, sem verður þá loksins gilduð til að skila gildi Haskell segðarinnar **sum** (\i->i^2) 10.

But this does not happen by first computing 0, then $0+1^2$, then $0+1^2+2^2$, et cetera. What happens is that first the *expression* 0 is constructed, then the *expression* $0+1^2$, then the *expression* $0+1^2+2^2$, et cetera. Finally we have the expression $0+1^2+2^2+\ldots+10^2$ which will then be evaluated to produce the value of the Haskell expression **sum** (\i->i^2) 10.

3.3 Listarithátturinn — List Notation

Haskell listarithátturinn hefur (auk venjulegs rithátta, svipað og í CAML og Morpho) annars vegar einfaldar runur á fernu sniði — The Haskell list notation has (in addition to the regular notation, similar to CAML and Morpho) simple sequences in four formats

- [i ..]
- [i .. j]
- [i , j ..]
- [i , j .. k]

og hins vegar flóknari og öflugri rithátt sem býður upp á skilgreiningar svo sem þessar — and also a more complicated and powerful notation that offers the possibility of definitions such as these:

```
1    fib3 :: [Integer]
2    fib3 = [1,1] ++ [a+b|(a,b)<-zip fib3 (tail fib3)]
```

Þessi listaritháttur í Haskell er kallaður *list comprehension* og er hannaður til að vera svipaður í útliti og merkingu eins og mengjaritháttur inn sem við eigum að venjast.

This list notation in Haskell is called *list comprehension* and is designed to be similar in notation and semantics to the set notation we are familiar with.

Við erum vön að sjá mengjaskilgreiningar svo sem $\{x^2 | x \in \{1..5\}\}$. Í Haskell má á svipaðan hátt skilgreina *listann*

```
1 [ x^2 | x<-[1..5] ]
```

Þetta er lögleg Haskell segð sem skilar [1,4,9,16,25].

We are used to seeing set definitions such as $\{x^2 | x \in \{1..5\}\}$. In Haskell we can similarly define the *list*

```
1 [ x^2 | x<-[1..5] ]
```

This is a valid Haskell expression that yields [1,4,9,16,25].

Athugið þó að listasegð er e.t.v. ekki reiknanleg. Til dæmis er gagnslaust að skilgreina eftirfarandi lista, þótt löglegur sé:

```
1 [(x,y,z) | x<-[1..], y<-[1..], z<-[1..], x^2+y^2==z^2]
```

Note that a list expression is not necessarily computable. For example it is useless to define the following, even though it is legal:

```
1 [(x,y,z) | x<-[1..], y<-[1..], z<-[1..], x^2+y^2==z^2]
```

En skrifa má aftur á móti — We can, however, write:

```
1 [(x,y,z)|x<-[2..],
2           y<-[1..(x-1)],
3           let z=floor $ sqrt $ fromIntegral (x^2+y^2),
4           x^2+y^2==z^2,
5           (gcd (gcd x y) z)==1
6 ]
```

Íhugið þessa segð vandlega. Hér kemur fyrir allt það sem leyft er í þessari gerð listaskilgreininga í Haskell:

Consider this expression carefully. Here we see every kind of thing that is allowed in this kind of list definition in Haskell:

- Framleiðendur svo sem — Generators such as

  x<−[2..].

- Skilgreiningar svo sem — Definitions such as

  let z=**floor** $ **sqrt** $ **fromIntegral** (x^2+y^2)

- Síur svo sem — Filters such as

  x^2+y^2==z^2.

Á Haskell síðunni[11] má finna formlega skilgreiningu á listarithættinum. Listarithátturinn bætir engu við það sem forrita má í Haskell án hans. Með hjálp fallanna concatMap, map og filter má gera allt það sem gera má með listarithættinum. En það er þá oftast flóknara og krefst stundum auk þess einfaldra hjálparfalla.

On the Haskell page[12] we can find a formal definition of the list notation. The notation does not add anything to what can be programmed in Haskell without it. With the help of the functions concatMap, map og filter we can do all that can be done using the list comprehension. But this is often more complicated and sometimes requires simple helper functions.

Til dæmis — For example:

```
1  [x^2|x<-[1..10]]
2   =
3  map (\x->x^2) [1..10]
4   =
5  concatMap (\x->[x^2]) [1..10]
```

og/and:

```
1  [(x,y)|x<-[1,2,3],y<-['a','b','c']]
2   =
3  concatMap (\x->[(x,y)|y<-['a','b','c']]) [1,2,3]
4   =
5  concatMap (\x->(concatMap (\y->[(x,y)])
6                            ['a','b','c']
7                  )
8            )
9            [1,2,3]
```

4 Haskell dæmi — Haskell Examples

Kíkið á eftirfarandi Haskell dæmi — Look at the following Haskell examples.

[11] http://haskell.org/onlinereport/exps.html

[12] http://haskell.org/onlinereport/exps.html

4.1 Prímtölur — Prime Numbers

Listi (straumur) allra prímtalna — The list (stream) of all prime numbers.

```
1  -- Höfundur: Snorri Agnarsson, snorri@hi.is
2  -- Þýðið með eftirfarandi skipun:
3  --    ghc -o primes.exe --make primes.hs
4
5  -- Author: Snorri Agnarsson, snorri@hi.is
6  -- Compile with the following command:
7  --    ghc -o primes.exe --make primes.hs
8
9  import System.CPUTime
10
11 -- Notkun: primes
12 -- Gildi:  Óendanlegur vaxandi listi allra prímtalna,
13 --          [2,3,5,7,11,13,17,...]
14
15 -- Usage:  primes
16 -- Value:  An infinite ascending list of all primes,
17 --          [2,3,5,7,11,13,17,...]
18 primes = [2]++[p|p<-[3,5..],isPrime p]
19
20 -- Notkun: factor ps n
21 -- Fyrir:  n er heiltala, stærri en 1, ps er vaxandi
22 --          óendanlegur listi ójafnra prímtalna.
23 --          Allar prímtölur sem ganga upp í n eru í ps.
24 -- Gildi:  Listi prímþátta n í vaxandi röð, eins oft
25 --          og þeir koma fyrir.  Margfeldi talnanna í
26 --          listanum er því n og allar tölurnar eru
27 --          prímtölur.
28
29 -- Usage:  factor ps n
30 -- Pre:    n is an integer, greater than 1, ps is an
31 --          ascending infinite list of unequal primes.
32 --          All primes that divide n are in ps.
33 -- Value:  The list of the prime factors of n in
34 --          ascending order, as often as they occur.
35 --          The product of the numbers in the list is
36 --          therefore n and all the numbers are primes.
37 factor (p:ps) n
38   | p*p > n       = [n]
39   | mod n p == 0 = [p] ++ factor (p:ps) (div n p)
40   | True          = factor ps n
41
42 -- Notkun: isPrime n
43 -- Fyrir:  n er heiltala >= 2.
44 -- Gildi:  True ef n er prímtala, False annars.
45
46 -- Usage:  isPrime n
47 -- Pre:    n is an integer >= 2.
```

```
48 -- Gildi:  True if n is a prime, False otherwise.
49 isPrime n = (head (factor primes n))==n
50
51 {-
52 Use:  dur <- measure s
53 Pre:  s is a finite list of values.
54       We must be inside the IO monad for
55       this call to be legal.
56 Post: The last value in the list s has
57       been written to stdout.
58       dur contains the time taken to find
59       the last value in s, in seconds,
60 -}
61 measure :: Show t => [t] -> IO Double
62 measure x = do
63     start <- getCPUTime
64     print (last x)
65     end <- getCPUTime
66     return (1.0e-12*(fromIntegral(end-start)))
67
68
69 -- Skrifum lista 100 fyrstu prímtalnanna
70 -- Write a list of the first 100 primes
71 --main :: IO ()
72 --main = do { print (take 100 primes) }
```

4.2 Pýþagórasarþrenndir — Pythagorean Triples

Listi allra Pýþagorasarþrennda (x, y, z) þar sem x, y og z eru heiltölur þ.a. $x^2+y^2 = z^2$ og engin þrennd er margfeldi af annari þrennd.

A list of all Pythagorean triples (x, y, z) where x, y and z are integers such that $x^2 + y^2 = z^2$ and no triple is a multiple of another triple.

```
1  -- Höfundur: Snorri Agnarsson, snorri@hi.is
2  -- Þýðið með eftirfarandi skipun:
3  --         ghc --make Pyth.hs
4
5  -- Author: Snorri Agnarsson, snorri@hi.is
6  -- Compile with the following command:
7  --         ghc --make Pyth.hs
8
9  pyth1 =
10   [(x,y,z) | y <- [2..],
11              x <- [1..(y-1)],
12              let z=floor (sqrt (fromIntegral (x^2+y^2))),
13              x^2+y^2==z^2,
14              (gcd (gcd x y) z)==1
15   ]
16
```

```
pyth2 =
  do { y <- [2..]
     ; x <- [1..(y-1)]
     ; let z=floor $ sqrt $ fromIntegral (x^2+y^2)
     ; zz <- if x^2+y^2==z^2 then [z] else []
     ; if (gcd (gcd x y) zz)==1 then [(x,y,zz)] else []
     }

pyth3 =
  [2..] >>= \y ->
  [1..(y-1)] >>= \x ->
  (let
     z=floor $ sqrt $ fromIntegral (x^2+y^2)
   in
     if x^2+y^2==z^2
       then
         [z]
       else
         []
  ) >>= \z ->
  if (gcd (gcd x y) z)==1
    then
      [(x,y,z)]
    else
      []

pyth4 =
  concatMap
    (\y ->
      (filter
        (\(x,y,z) -> (gcd (gcd x y) z)==1)
        (filter
          (\(x,y,z) -> x^2+y^2==z^2)
          (concatMap
            (\x ->
               let
                 z=floor (sqrt (fromIntegral (x^2+y^2)))
               in
                 [(x,y,z)]
            )
            [1..(y-1)]
          )
        )
      )
      [2..]

pyth5 =
  filter
    (\(x,y,z) -> (gcd (gcd x y) z)==1)
```

```
67      (filter
68        (\(x,y,z) -> x^2+y^2==z^2)
69        (concatMap
70          ( \y ->
71            (
72              concatMap
73                ( \x ->
74                  let
75                    z=floor (sqrt (fromIntegral (x^2+y^2)))
76                  in
77                    [(x,y,z)]
78                )
79                [1..(y-1)]
80            )
81          )
82          [2..]
83        )
84      )
85
86  -- Notkun: intSqrt n
87  -- Fyrir:  n er heiltala, n>=1
88  -- Gildi:  Listi allra heiltalna k>0 þ.a. k*k==n
89  -- Aths.:  Listinn verður annaðhvort tómur eða
90  --         inniheldur eina heiltölu.
91
92  -- Usage:  intSqrt n
93  -- Pre:    n is an integer, n>=1
94  -- Value:  A list of all integers k>0 such that k*k==n
95  -- Note:   The list will either be empty or contain one
96  --         integer.
97  intSqrt :: Integral a => a -> [a]
98  intSqrt n =
99      let
100       -- Notkun: help k
101       -- Fyrir:  k er heiltala, k>=1
102       -- Gildi:  Listi allra heiltalna k>0 þ.a. k*k==n
103
104       -- Usage:  help k
105       -- Pre:    k is an integer, k>=1
106       -- Value:  The list of all integers k>0 such that k*k==n
107       help k =
108         let k' = (div ((div n k)+k) 2)
109         in
110           if k' == k || k' == k+1
111             then
112               if k'^2 == n then [k'] else if k^2 == n then [k] else []
113             else
114               help k'
115     in
116       help$floor$sqrt$fromIntegral n
```

```
117
118 pyth6 =
119   [(x,y,z) | y <- [1..],
120              x <- [1..y],
121              z <- intSqrt (x^2+y^2),
122              (gcd (gcd x y) z)==1
123   ]
124
125 main :: IO ()
126 main =
127   print (take 10 pyth1) >>= \_ ->
128   print (take 10 pyth2) >>= \_ ->
129   print (take 10 pyth3) >>= \_ ->
130   print (take 10 pyth4) >>= \_ ->
131   print (take 10 pyth5) >>= \_ ->
132   print (take 10 pyth6)
```