Funkcie a funkcionály na ceste k Wholemeal (functional) programming

Peter Borovanský

I-18

http://dai.fmph.uniba.sk/courses/FPRO/

Čo je wholemeal (celozrnné)

Geraint Jones: Wholemeal programming means to **think big**:

- **work with an entire list**, rather than a sequence of elements
- develop **a solution space**, rather than an individual solution
- imagine **a graph**, rather than a single path.

first

- **solve a more general problem**,

then

- extract the interesting bits and pieces by transforming the general program into more specialized ones

Wholemeal programming je štýl rozmýšlania, programovania

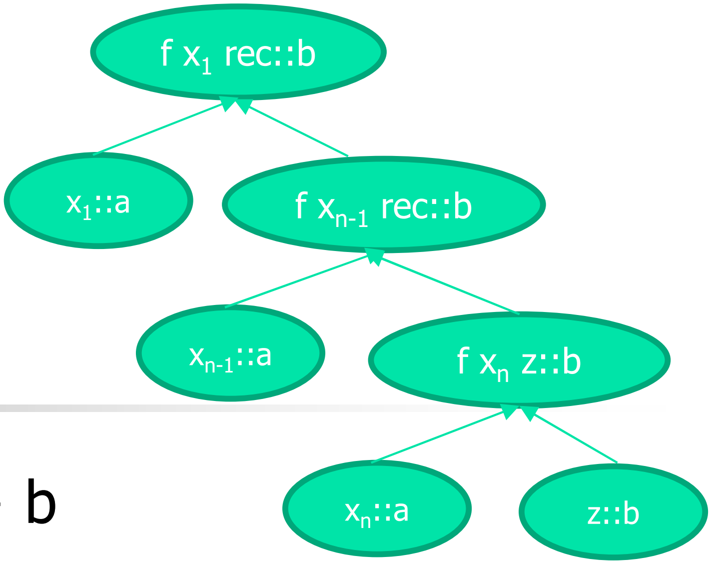
... privedie vás k *šlachtickým* manierom vo funkcionálnom svete

Celozrnný programátor musí

poznať funkcie a najzákladnejšie funkcionály

- map/filter
  - map f xs = map f $[x_1, \ldots, x_n]$ = $[f\ x_1, \ldots, f\ x_n]$ = [f x | x <- xs]
  - filter f xs = filter p $[x_1, \ldots, x_n]$ = [x | x <- xs, p x]
- foldr/foldl
  - foldr f z $[x_1, \ldots, x_n]$ = $(f\ x_1\ (f\ x_2\ \ldots\ (f\ x_n\ z)..))$
  - foldl f z $[x_1, \ldots, x_n]$ = $(..((f\ z\ x_1)\ x_2)\ \ldots\ x_n)$
- scanr/scanl
  - scanr f z $[x_1, \ldots, x_n]$ = reverse $[z, (f\ x_n\ z), \ldots, (f\ x_2 \ldots (f\ x_n\ z)..), (f\ x_1\ (f\ x_2 \ldots (f\ x_n\ z)..))]$
  - scanl f z $[x_1, \ldots, x_n]$ = $[z, (f\ z\ x_1), ((f\ z\ x_1)\ x_2), \ldots, (..((f\ z\ x_1)\ x_2)\ \ldots\ x_n)]$
  - scanr1 f $[x_1, \ldots, x_n]$ = reverse $[x_n, (f\ x_{n-1}\ x_n), \ldots, (f\ x_1\ (f\ x_2 \ldots (f\ x_{n-1}\ x_n)..))]$
  - scanl1 f $[x_1, \ldots, x_n]$ = $[x_1, (f\ x_1\ x_2), ((f\ x_1\ x_2)\ x_3), \ldots, (..((f\ x_1\ x_2)\ x_3)\ \ldots\ x_n)]$
- iterate
  - iterate f x = $[x, (f\ x), ((f\ \ x)\ x), \ldots, f^n\ x, \ldots]$
- concat, … a t.ď.

Haskell – foldr




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

foldr f z []      = z
foldr f z (x:xs) = f x (foldr f z xs)

a : b : c : [] -> f a (f b (f c z))
```

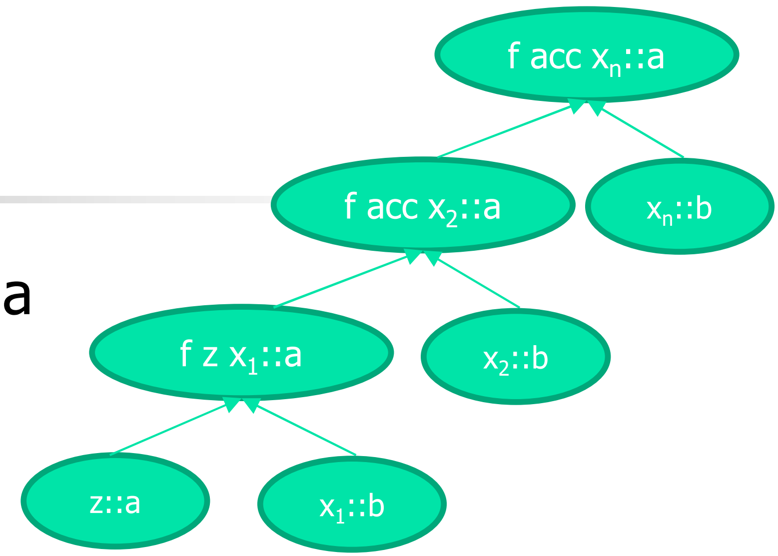
```
     :                     f
    / \                   / \
   a   :    foldr f z    a   f
      / \   --------->      / \
     b   :                 b   f
        / \                   / \
       c  []                 c   z
```

```
Main> foldr (+) 0 [1..100]
5050
```

```
Main> foldr (\x y->10*y+x) 0 [1,2,3,4]
4321
```

```
-- g je vnorená lokálna funkcia

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f z = g
  where    g []       = z
           g (x:xs) = f x (g xs)
```

Haskell – foldl




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

foldl f z []       =  z
foldl f z (x:xs) =  foldl f (f z x) xs

a : b : c : [] -> f (f (f z a) b) c
```

```
  :                    f
 / \                  / \
a   :    foldl f z   f   c
   / \   ------>    / \
  b   :            f   b
     / \          / \
    c  []        z   a
```

```
Main> foldl (+) 0 [1..100]
5050

Main> foldl (\x y->10*x+y) 0 [1,2,3,4]
1234
```

Vypočítajte

- foldr max (-999) [1,2,3,4]
  foldl max (-999) [1,2,3,4]

- foldr (_ -> \y ->(y+1)) 0 [3,2,1,2,4]
  foldl (\x -> _ ->(x+1)) 0 [3,2,1,2,4]

- foldr (-) 0 [1..100] =

(1-(2-(3-(4-...-(100-0))))) = 1-2 + 3-4 + 5-6 + ... + (99-100) = -50

- foldl (-) 0 [1..100] =

  (...(((0-1)-2)-3) ... - 100) = -5050

Kvíz

```
   :                         f
  / \                       / \
 a   :    foldl f z        f   c
    / \   ------>         / \
   b   :                 f   b
      / \               / \
     c  []             z   a
```

foldr (:) [] xs =   xs

foldr (:) ys xs =   xs++ys

foldr ? ? xs = reverse xs

foldr ((:) . h) [] = ???

```
   :                         f
  / \                       / \
 a   :    foldr f z        a   f
    / \   -------->           / \
   b   :                     b   f
      / \                       / \
     c  []                     c   z
```

Pre tých, čo zvládli kvíz, odmena !

kliknite si podľa vašej politickej orientácie

Funkcia je hodnotou

- [a->a] je zoznam funkcií typu a->a
  napríklad: [(+1),(+2),(*3)] je [\x->x+1,\x->x+2,\x->x*3]

- čo je foldr (.) id [(+1),(+2),(*3)] ?? [a->a]
  akého je typu
  foldr (.) id [(+1),(+2),(*3)] 100 303
  foldl (.) id [(+1),(+2),(*3)] 100 ???

lebo skladanie fcií je asociatívne:

- ((f . g) . h) x = (f . g) (h x) = f (g (h x)) = f ((g . h) x) = (f . (g . h)) x

- funkcie nevieme porovnávať, napr. head [(+1),(+2),(*3)] == id

- funkcie vieme permutovať, length $ permutations [(+1),(+2),(*3),(^2)]

fold.hs

Maximálna permutácia funkcií

- zoznam funkcií aplikujeme na zoznam argumentov

```
apply          :: [a -> b] -> [a] -> [b]
apply fs args = [ f a | f <- fs, a <- args]
```

```
apply [(+1),(+2),(*3)] [100, 200]
[101,201,102,202,300,600]
```

Dokážte/vyvraťte: map f . apply fs = apply (map (f.) fs)

- čo počíta tento výraz

```
maximum $
   apply
     (map (foldr (.) id) (permutations [(+1),(^2),(*3),(+2),(/3)]))
     [100]
```

31827

- ((+1).(+2).(*3).(^2).(/3)) 100

3336.333333333334

- ((/3).(^2).(*3).(+2).(+1)) 100

31827.0

fold.hs

take pomocou foldr/foldl

Výsledkom foldr ?f? ?z? xs je funkcia, do ktorej keď dosadíme n, vráti take n:

… preto aj ?z? musí byť funkcia, do ktorej keď dosadíme n, vráti take n []:

```
take'          :: Int -> [a] -> [a]
take' n xs  =  (foldr pomfcia (\_ -> []) xs) n where
                     pomfcia x h = \n -> if n == 0 then []
                                                 else x:(h (n-1))
                     alebo
                     pomfcia x h n =  if n == 0 then [] else x:(h (n-1))
                     alebo
take''' n xs = foldr (\a -> \h -> \n -> case n of
                                            0 -> []
                                            n -> a:(h (n-1)) )
                       (\_ -> [])
                     xs
                     n
```

foldoviny.hs

Extrémny príklad celozrnného

2 vzájomne rekurzívne fcie

```
rozdelParneNeparne :: [Integer] -> ([Integer],[Integer])
rozdelParneNeparne [] = ([],[])
rozdelParneNeparne (x:xs) = (xp, x:xn) where (xp, xn) = rozdelNeparneParne xs
```

```
rozdelNeparneParne :: [Integer] -> ([Integer],[Integer])
rozdelNeparneParne [] = ([],[])
rozdelNeparneParne (x:xs) = (x:xp, xn) where  (xp, xn) = rozdelParneNeparne xs
```

```
rozdielSuctu :: [Integer] -> Integer
rozdielSuctu xs = sum parneMiesta - sum neparneMiesta
    where (parneMiesta, neparneMiesta) = rozdelParneNeparne xs
```

Celozrnné riešenie:

rozdielSuctu = negate . foldr (-) 0

alebo len -foldr(-)0

RozdielSuctu.hs

Krok-po-kroku

(len pre tých, čo to nepochopili ešte)

- **Krok 1** - zbierame párne a nepárne prvky do zoznamov

```
rozdielSuctu'' xs  = (sum p) - (sum n)
        where (p,n) = foldr (\x -> \(a,b) -> (b,x:a)) ([],[]) xs
```

- **Krok 2** - prečo nepočítať súčet už hneď

```
rozdielSuctu''' xs  = p - n
        where (p,n) = foldr (\x -> \(a,b) -> (b,a+x)) (0,0) xs
```

- **Krok 3** – ušetrený where, zistíme, čo je uncurry

```
rozdielSuctu'''' xs  = uncurry (-) $ foldr (\x -> \(a,b) -> (b,a+x)) (0,0) xs
uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (a,b) = f a b
```

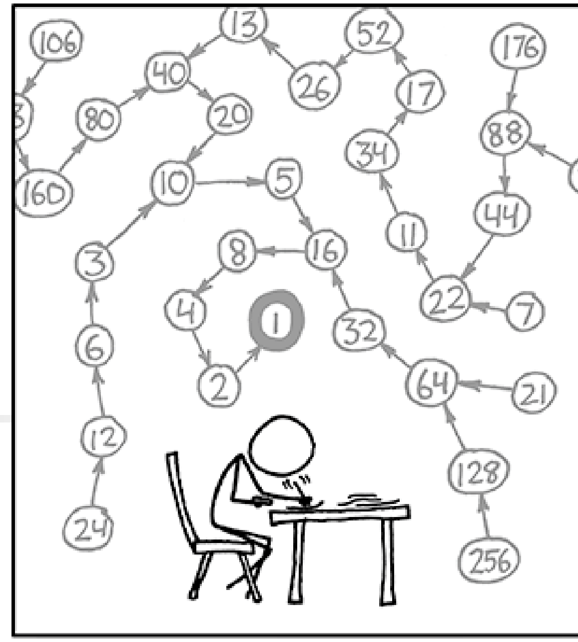
- **Krok 4** – ušetrený explicitný argument

```
rozdielSuctu'''''   = uncurry (-) . foldr (\x -> \(a,b) -> (b,a+x)) (0,0)
```

RozdielSuctu.hs

Collatz
(a na príkladoch)

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod{2} \\ 3n+1 & \text{if } n \equiv 1 \pmod{2}. \end{cases}$$



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Čo robí táto funkcia ?

```
collatz     :: Integer -> [Integer]
collatz 1  = []
collatz n | even n              = n : collatz (n `div` 2)
          | otherwise           = n : collatz (3 * n + 1)
```

To sú prvky tzv. Colatzovej postupnosti

```
collatz'= takeWhile (/=1) . iterate (\x -> if even x then x `div` 2 else 3 * x + 1)
```

iterate :: (a -> a) -> a -> [a]

iterate f x = [x, f x, f f x, f f f x, … , f^nx, …]

27, 82, **41**, 124, 62, **31**, 94, **47**, 142, **71**, 214, **107**, 322, **161**, 484, 242, **121**, 364, 182, **91**, 274, **137**, 412, 206, **103**, 310, **155**, 466, **233**, 700, 350, **175**, 526, **263**, 790, **395**, 1186, **593**, 1780, 890, **445**, 1336, 668, 334, **167**, 502, **251**, 754, **377**, 1132, 566, **283**, 850, **425**, 1276, 638, **319**, 958, **479**, 1438, **719**, 2158, **1079**, 3238, **1619**, 4858, **2429**, 7288, 3644, 1822, **911**, 2734, **1367**, 4102, **2051**, 6154, **3077**, 9232, 4616, 2308, 1154, **577**, 1732, 866, **433**, 1300, 650, **325**, 976, 488, 244, 122, **61**, 184, 92, 46, **23**, 70, **35**, 106, **53**, 160, 80, 40, 20, 10, **5**, 16, 8, 4, 2, **1**

Intro.hs

Cifry
(niektoré vaše riešenia)

```
module Cifry where
cifry 12345 == [1,2,3,4,5]
cifryR 12345 == [5,4,3,2,1]

cifry n = reverse (cifryR n)
cifryR 0 = []
cifryR n = (n `mod` 10):(cifryR (n `div` 10))
```

Intro.hs

Cifry

```
module Cifry where
cifry 12345 == [1,2,3,4,5]
cifryR 12345 == [5,4,3,2,1]

cifry      :: Integer -> [Integer]
cifry n    = map(`mod` 10) $ reverse $
                  takeWhile (> 0) $ iterate (`div`10) n
iterate (`div` 10) 12345 == [12345,1234,123,12,1,0,0,0,0,0,0,0,0,0...]
                            [1,12,123,1234,12345]
                            [1,  2,   3,    4,     5]
cifry' = map(`mod` 10) . reverse . takeWhile (> 0) . iterate (`div`10)
cifryR n = map(`mod` 10) $ takeWhile (> 0) $ iterate (`div`10) n
cifryR'   = map(`mod` 10) . takeWhile (> 0) . iterate (`div`10)
```

Intro.hs

Kritérium delietel'nosti 11

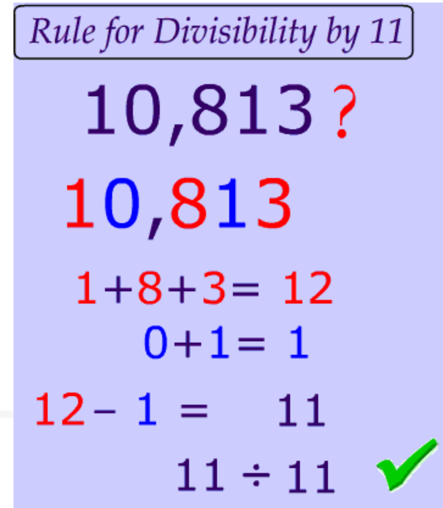
- rodné číslo 786115 3333 (ženské, *15.nov1978)
- 7861153333 `mod` 11 == 0
- 11 | 7861153333 iff 11 | 7+6+1+3+3 –(8+1+5+3+3) = 0

- naše rodné čísla sú delitelné 11, ľahká kontrola

- čísla kariet majú tiež kontrolu, Luhnnov algo, DÚ1

- čo bankové účty
- 7000155733 / 8180 – soc.poisťovňa
- cifry násobíme váhami 6,3,7,9,10,5,8,4,2,1, sčítame, výsledok delitelný 11
- 11 | 7*6+0*3+0*7+0*9+1*10+5*5+5*8+7*4+3*2+3*1
- (sum $ zipWith (*) [7,0,0,0,1,5,5,7,3,3] [6,3,7,9,10,5,8,4,2,1]) `mod` 11
- (sum $ zipWith (*) [2,7,0,1,1,3,2,4,4,3] [6,3,7,9,10,5,8,4,2,1]) `mod` 11

Delitel’nost’ 11



- SK67 8360 5207 0042 0002 6991
- 678360520700420002 6991=11*61669138245492 7275181

- Rodné číslo (.cz, .sk) je delitel’né 11

foldr (-) 0 $

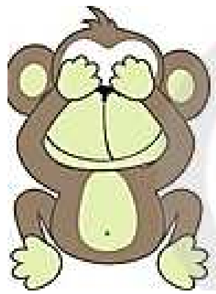
```
oneStep :: Integer -> Integer
oneStep = \n -> abs $
```
~~`uncurry (-) $ foldr (\c -> \(sp,sn) -> (c+sn, sp)) (0,0) $`~~
```
        map (`mod` 10) $ takeWhile (>0) $ iterate (`div` 10) n

allSteps :: Integer -> Bool
allSteps = \n -> 0 == (head $ dropWhile (>9) $ iterate oneStep n)

qch1   =  quickCheck(\n -> (n>0) ==> allSteps n == (n `mod` 11 == 0))
```
***Eleven>** qch1

+++ OK, passed 100 tests.

Eleven.hs

Binárne číslo $\{1\}^{+}\{0\}^{*}$

- $\underbrace{111...111}_{m}\underbrace{00.....0000}_{n} = (2^m-1) * 2^n$



```
null $                              True
  dropWhile (==1) $                 []
    dropWhile(==0) $                [1, 1]
      map (`mod` 2) $               [0, 0, 1, 1]
        takeWhile (>0) $            [12, 6, 3, 1]
          iterate (`div` 2)         12 = [12,6,3,1,0,0,0...]
```

```
suma       + i*cenaPiva = (2^m-1) * 2^n
suma `mod` cenaPiva = ((2^m-1) * 2^n) `mod` cenaPiva
suma `mod` cenaPiva = (  (2^m-1) `mod` cenaPiva
                          *
                          2^n `mod` cenaPiva
                      ) `mod` cenaPiva
```

mod 11	2^n	2^{m-1}	mod 11
1	1	0	0
2	2	1	1
4	4	3	3
8	8	7	7
5	16	15	4
10	32	31	9
9	64	63	8
7	128	127	6
3	256	255	2
6	512	511	5
1	1024	1023	0

Wholemeal in functional

- dnes na príklade Sudoku Solvera (podľa: Richard Bird)

The wholemeal approach often offers new insights or provides new perspectives on a given problem. It is nicely complemented by the idea of projective programming:

first

- solve a more general problem,

then

- extract the interesting bits and pieces by transforming the general program into more specialised ones."

https://www.cs.tufts.edu/~nr/cs257/archive/richard-bird/sudoku.pdf

rôzne sudoku solvery (v Haskelli)
http://www.haskell.org/haskellwiki/Sudoku

Sudoku

8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5

```
type Matrix a       = [Row a]
type Row a          = [a]
type Value          = Char

type Grid           = Matrix Value
               -- de facto [[Char]]
easy                :: Grid
easy                =                    -- [String] = [[Char]]
                    [   "2....1.38",
                        "........5",
                        ".7...6...",     -- String = [Char]
                        ".......13",
                        ".981..257",
                        "31....8..",
                        "9..8...2.",
                        ".5..69784",
                        "4..25...." ]

solve               :: Grid -> [Grid]   -- nájdi všetky riešenia
```

Sudoku.hs

rows . rows = id
cols . cols = id

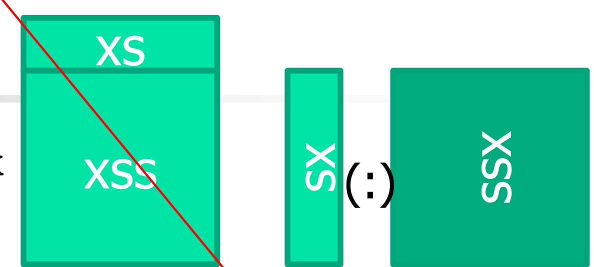
Základné definície

```
boxsize                :: Int            -- 9 štvorcov 3x3
boxsize                =  3

values                 :: [Value]        -- prípustné hodnoty
values                 =  ['1'..'9']

empty                  :: Value -> Bool        -- nevyplnené ?
empty                  =  (== '.')

blank                  :: Grid           -- vytvor prázdny štvorec
blank                  =  replicate n (replicate n '.')
                          where n = boxsize ^ 2
replicate n x          = [ x | i<-[1..n] ]

rows                   :: Matrix a -> [Row a] -- zoznam riadkov
rows                   =  id

cols                   :: Matrix a -> [Row a] -- zoznam stĺpcov
cols                   =  transpose
```

Sudoku.hs

Trasponovanie matice

(stĺpce sa stanú riadkami)




```
transpose'              :: Matrix a -> Matrix
transpose' [xs]         = [[x] | x <- xs]
transpose' (xs:xss) = zipWith (:) xs (transpose' xss)

zipWith                         :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith f (x:xs) (y:ys) = f x y : (zipWidth f xs ys)
zipWith _ _          _      = []
```

- pokúsme sa transpose' prepísať pomocou foldr:

```
transpose'' xss     = foldr (\xs -> \rek  -> zipWith (:) xs rek)
                           -- (replicate (length xss) [])
                           [ [] | _ <- [1..(length xss)]]
                         xss
```

- a funguje to ?
- vieme napísať transpose pomocou foldl

Sudoku.hs

all/any::(a->Bool)->[a]->Bool

Korektné riešenie

```
valid                  :: Grid -> Bool   -- bezosporné riešenie
valid g                =  all nodups (rows g) &&
                          all nodups (cols g) &&
                          all nodups (boxs g)

nodups                 :: Eq a => [a] -> Bool -- bez duplikátov
nodups []              =  True
nodups (x:xs)          =  not (elem x xs) && nodups xs

boxs                   :: Matrix a -> [Row a] -- zoznam 3x3 štvorcov
boxs                   =  unpack . map cols . pack
                          where
                           unpack = map concat . concat
                           pack   = group3 . map group3
                           group3  = group boxsize
                           group      :: Int -> [a] -> [[a]]
                           group n [] = []
                           group n xs = take n xs : group n (drop n xs)
```

Sudoku.hs

Turbo - SudokuStvorce

Definujte vlastnú verziu boxs, ktorá implementuje:

Nech toto je `e::Grid = [[9*i+j+1 | j <- [0..8]] | i <- [0..8]]`

```
[[1, 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],
[46,47,48,49,50,51,52,53,54],
[55,56,57,58,59,60,61,62,63],
[64,65,66,67,68,69,70,71,72],
[73,74,75,76,77,78,79,80,81]]
```

```
Main> boxs e
[[1,2,3,10,11,12,19,20,21],
[4,5,6,13,14,15,22,23,24],
[7,8,9,16,17,18,25,26,27],
[28,29,30,37,38,39,46,47,48],
[31,32,33,40,41,42,49,50,51],
[34,35,36,43,44,45,52,53,54],
[55,56,57,64,65,66,73,74,75],
[58,59,60,67,68,69,76,77,78],
[61,62,63,70,71,72,79,80,81]]
```

Riešenie s indexovaním

```
sudokuStvorce :: [[Int]] -> [[Int]]
sudokuStvorce m    =
   concat [
          [
            [ m!!(3*i+k)!!(3*j+l) | k <- [0..2], l <- [0..2] ]
              | j <- [0..2]
          ] | i <- [0..2]
   ]
```

- iné riešenie -veľmi podobné:

```
blocks :: [[Int]]
blocks = [[0..2],[3..5],[6..8]]
sudokuStvorce :: [[Int]] -> [[Int]]
sudokuStvorce xs = [
       [xs!!r!!c | r <- b1, c <- b2 ] | b1<-blocks, b2<-blocks ]
```

Riešenie s indexovaním

```
sudokuStvorce :: [[Int]] -> [[Int]]
sudokuStvorce [] = []
sudokuStvorce (x:y:z:xs) =
   [(splitto3 x)!!i ++ (splitto3 y)!!i ++ (splitto3 z)!!i |
        i<-[0..2]]
   ++ sudokuStvorce xs

splitto3 x = [take 3 x, take 3 (drop 3 x), drop 6 x]
```

```haskell
--
sudokuStvorce  :: [[Int]] -> [[Int]]
sudokuStvorce xss = [sudokuStvorce' xss i (i+3) j (j+3) |
                                    i <- [0, 3, 6], j <- [0, 3, 6]]
sudokuStvorce' :: [[Int]] -> Int -> Int -> Int -> Int -> [Int]
sudokuStvorce' xss r1 r2 s1 s2 =  concat
   [[x | (j, x) <- zip [0..] xs, j < s2 && j >= s1]
       | (i, xs) <- zip [0..] xss, i < r2 && i >= r1]
--
sudokuStvorce :: [[Int]] -> [[Int]]
sudokuStvorce m =
   foldr (++) []
     [[foldr (++) []
        [take 3 (drop (3*cc) row)   | row <- rm]
                                    | cc <- [0..2]] | rm <- rows]
   where rows = [take 3 (drop (3*rc) m) | rc <- [0..2]]
```

Boxs

(krok 1 - pack)

```
boxs   =  unpack . map cols . pack
          where
           unpack = map concat . concat
           pack   = group3 . map group3
           group3  = group boxsize
           group   :: Int -> [a] -> [[a]]
           group n [] = []
           group n xs = take n xs : group n (drop n xs)
```

```
Main > group 3 [1..9]                      -- toto robí group3
[ [1,2,3], [4,5,6], [7,8,9] ]

Main > (group3 . map group3) e             -- iný zápis pre group3 (map group3 e)
[[[[  1, 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 ], [ 49,50,51 ], [ 52,53,54 ]] ],
 [[[  55,56,57 ], [ 58,59,60 ], [ 61,62,63 ]],
   [[ 64,65,66 ], [ 67,68,69 ], [ 70,71,72 ]],
   [[ 73,74,75 ], [ 76,77,78 ], [ 79,80,81 ]] ]]
```

```
boxs   =  unpack . map cols . pack
          where
           unpack = map concat . concat
           pack   = group3 . map group3
           group3  = group boxsize
           group   :: Int -> [a] -> [[a]]
           group n [] = []
           group n xs = take n xs : group n (drop n xs)
```

Boxs

(krok 2 – map cols)

```
Main > ((map cols ) . (group3. map group3)) e

[ [  [[ 1, 2, 3 ], [ 10,11,12 ], [ 19,20,21 ]],
     [[ 4, 5, 6 ], [ 13,14,15 ], [ 22,23,24 ]],
     [[ 7, 8, 9 ], [ 16,17,18 ], [ 25,26,27 ]] ],
  [ [[ 28,29,30 ], [ 37,38,39 ], [ 46,47,48 ]],
    [[ 31,32,33 ], [ 40,41,42 ], [ 49,50,51 ]],
    [[ 34,35,36 ], [ 43,44,45 ], [ 52,53,54 ]] ],
  [ [[ 55,56,57 ], [ 64,65,66 ], [ 73,74,75 ]],
    [[ 58,59,60 ], [ 67,68,69 ], [ 76,77,78 ]],
    [[ 61,62,63 ], [ 70,71,72 ], [ 79,80,81 ]] ] ]
```

Boxs

(krok 3 - unpack)

```
boxs   =  unpack . map cols . pack
          where
           unpack = map concat . concat
           pack    = group3 . map group3
           group3   = group boxsize
           group    :: Int -> [a] -> [[a]]
           group n [] = []
           group n xs = take n xs : group n (drop n xs)
```

```
concat :: [[a]] -> [a]
concat [[1,2,3],[4,5],[6]] = [1,2,3,4,5,6]
```

```
Main > ((map concat . concat) . (map cols ) . (group3. map group3))
   e
[[ 1, 2, 3 ,  10,11,12 ,  19,20,21 ],
 [ 4, 5, 6 ,  13,14,15 ,  22,23,24 ],
 [ 7, 8, 9 ,  16,17,18 ,  25,26,27 ],
 [ 28,29,30 ,  37,38,39 ,  46,47,48 ],
 [ 31,32,33 ,  40,41,42 ,  49,50,51 ],
 [ 34,35,36 ,  43,44,45 ,  52,53,54 ],
 [ 55,56,57 ,  64,65,66 ,  73,74,75 ],
 [ 58,59,60 ,  67,68,69 ,  76,77,78 ],
 [ 61,62,63 ,  70,71,72 ,  79,80,81 ] ]
```

```
[ [ [[[ 1,, 2,, 3 ]],, [[ 10,,11,,12 ]],, [[ 19,,20,,21 ]]]],,
    [[ 4, 5, 6 ], [ 13,14,15 ], [ 22,23,24 ]],
    [[ 7, 8, 9 ], [ 16,17,18 ], [ 25,26,27 ]] ],
  [[[ 28,29,30 ], [ 37,38,39 ], [ 46,47,48 ]],
   [[ 31,32,33 ], [ 40,41,42 ], [ 49,50,51 ]],
   [[ 34,35,36 ], [ 43,44,45 ], [ 52,53,54 ]] ],
  [[[ 55,56,57 ], [ 64,65,66 ], [ 73,74,75 ]],
   [[ 58,59,60 ], [ 67,68,69 ], [ 76,77,78 ]],
   [[ 61,62,63 ], [ 70,71,72 ], [ 79,80,81 ]] ] ]
```

concat . group3= id
group3 . concat = id

Vlastnosti

Platí, že:

```
rows . rows = id
cols . cols = id
boxs . boxs = id,          kde boxs  =  unpack . map cols . pack

(unpack . map cols . pack) . (unpack . map cols . pack) =
```

dosadíme:

```
(map concat . concat) . map cols . (group3 . map group3) .    -- pokračuje nižšie
(map concat . concat) . map cols . (group3 . map group3) =
```

asociatívnosť

```
map concat . concat . map cols . group3 . map group3 .
map concat . concat . map cols . group3 . map group3 =

map concat . concat . map cols . group3 .
concat . map cols . group3 . map group3 =

map concat . concat . map cols . map cols . group3 . map group3 =
map concat . concat . group3 . map group3 =
map concat . map group3 =
id ☺
```

Dokážte, či vyvráťte, že group3 . concat = id

Na príklade

```
Riešenie Turbo – pre kontrolu
[[1,2,3,10,11,12,19,20,21],
[4,5,6,13,14,15,22,23,24],
[7,8,9,16,17,18,25,26,27],
[28,29,30,37,38,39,46,47,48],
[31,32,33,40,41,42,49,50,51],
[34,35,36,43,44,45,52,53,54],
[55,56,57,64,65,66,73,74,75],
[58,59,60,67,68,69,76,77,78],
[61,62,63,70,71,72,79,80,81]]
```

Main> e -- kde e::Grid = [[9*i+j+1 | j <- [0..8]] | i <- [0..8]]

- [[1,2,3,4,5,6,7,8,9],[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],[46,47,48,49,50,51,52,53,54],[55,56,57,58,59,60,61,62,63],[64,65,66,67,68,69,70,71,72],[73,74,75,76,77,78,79,80,81]]

Main> map group3 e

- [[[1,2,3],[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],[49,50,51],[52,53,54]],[[55,56,57],[58,59,60],[61,62,63]],[[64,65,66],[67,68,69],[70,71,72]],[[73,74,75],[76,77,78],[79,80,81]]]

Main> (group3.map group3) e

- [[[[1,2,3],[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],[49,50,51],[52,53,54]]],[[[55,56,57],[58,59,60],[61,62,63]],[[64,65,66],[67,68,69],[70,71,72]],[[73,74,75],[76,77,78],[79,80,81]]]]

Main> ((map cols).(group3.map group3)) e

- [[[[1,2,3],[10,11,12],[19,20,21]],[[4,5,6],[13,14,15],[22,23,24]],[[7,8,9],[16,17,18],[25,26,27]]],[[[28,29,30],[37,38,39],[46,47,48]],[[31,32,33],[40,41,42],[49,50,51]],[[34,35,36],[43,44,45],[52,53,54]]],[[[55,56,57],[64,65,66],[73,74,75]],[[58,59,60],[67,68,69],[76,77,78]],[[61,62,63],[70,71,72],[79,80,81]]]]

Main> (concat.(map cols).(group3.map group3)) e

- [[[1,2,3],[10,11,12],[19,20,21]],[[4,5,6],[13,14,15],[22,23,24]],[[7,8,9],[16,17,18],[25,26,27]],[[28,29,30],[37,38,39],[46,47,48]],[[31,32,33],[40,41,42],[49,50,51]],[[34,35,36],[43,44,45],[52,53,54]],[[55,56,57],[64,65,66],[73,74,75]],[[58,59,60],[67,68,69],[76,77,78]],[[61,62,63],[70,71,72],[79,80,81]]]

Main> ((map concat.concat).(map cols).(group3.map group3)) e

- [[1,2,3,10,11,12,19,20,21],[4,5,6,13,14,15,22,23,24],[7,8,9,16,17,18,25,26,27],[28,29,30,37,38,39,46,47,48],[31,32,33,40,41,42,49,50,51],[34,35,36,43,44,45,52,53,54],[55,56,57,64,65,66,73,74,75],[58,59,60,67,68,69,76,77,78],[61,62,63,70,71,72,79,80,81]]

Nájdenie všetkých riešení

```
type Choices      =   [Value]
```
-- zoznam možností jedného políčka

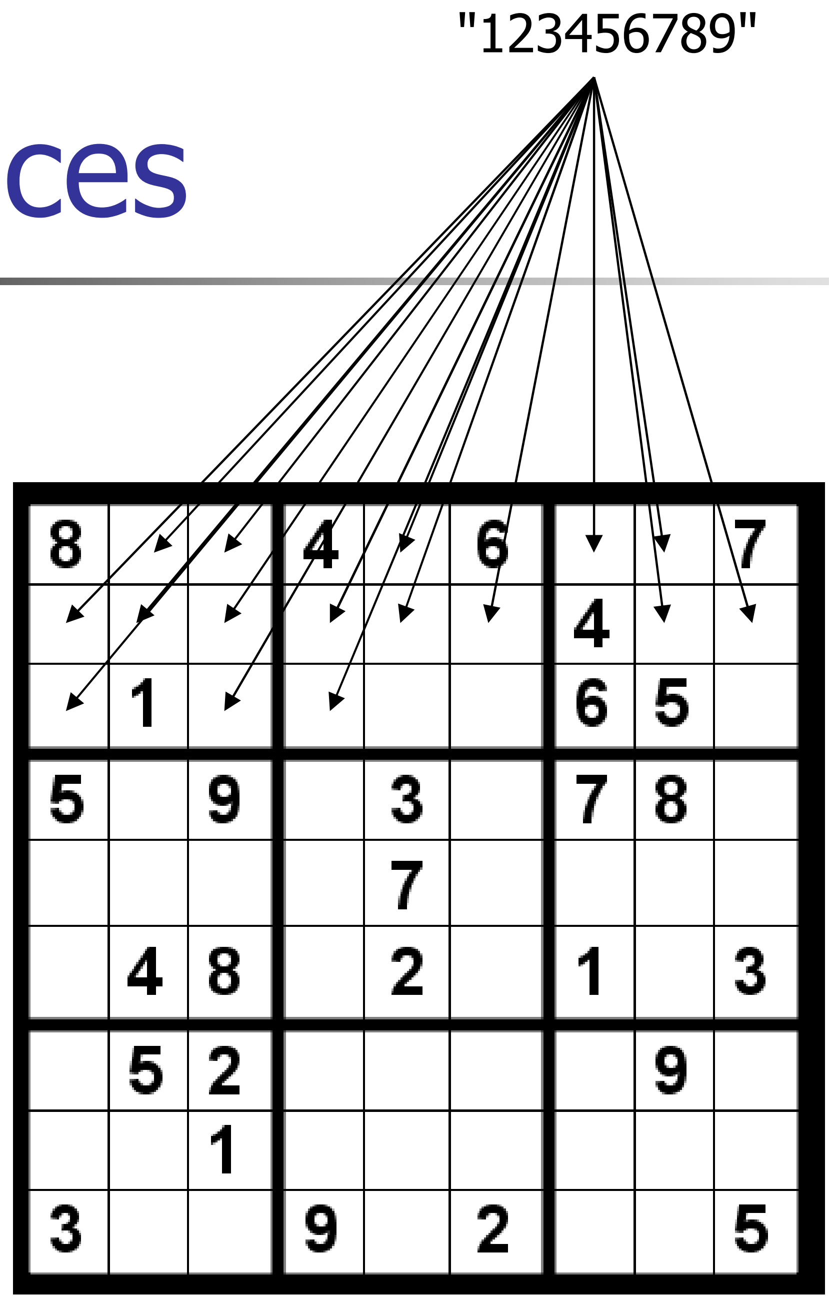
-- do každého políčka, kde je '.', vpíšeme úplne všetky možnosti

```
choices             :: Grid -> Matrix Choices
choices             =  map (map choice)
                         where
                         choice v = if empty v then values else [v]
```

```
Main> easy
["2....1.38","........5",".7...6...",".......13",".981..257","31....8..","9..8...2.",
    ".5..69784","4..25...."]
Main> choices easy
[["2","123456789","123456789","123456789","123456789","1","123456789","3","8"],["1234
    56789","123456789","123456789","123456789","123456789","123456789","123456789","12
    3456789","5"],["123456789","7","123456789","123456789","123456789","6","123456789"
    ,"123456789","123456789"],["123456789","123456789","123456789","123456789","123456
    789","123456789","123456789","1","3"],["123456789","9","8","1","123456789","123456
    789","2","5","7"],["3","1","123456789","123456789","123456789","123456789","8","12
    3456789","123456789"],["9","123456789","123456789","8","123456789","123456789","12
    3456789","2","123456789"],["123456789","5","123456789","123456789","6","9","7","8"
    ,"4"],["4","123456789","123456789","2","5","123456789","123456789","123456789","12
    3456789"]]
```

Sudoku.hs

Choices



27 singles

81-27=54 empty

9^{54} =3_381_391_913_522_726_342_930_221_472_392_241_170_198_527_451_848_561 možností

Nájdenie všetkých riešení

```
-- kartézsky súčin všetkých možností v jednom riadku
cp                        :: [[a]] -> [[a]]      --    Row[a] -> Row[a]
cp []                     =  [[]]
cp (xs:xss)               =  [y:ys | y<-xs, ys<-cp xss]

Main > cp [ [1,2,3], [4,5], [6] ]
[[1,4,6],[1,5,6],[2,4,6],[2,5,6],[3,4,6],[3,5,6]]
```

A potrebujeme cp aj na matici...

```
collapse                  :: Matrix [a] -> [Matrix a]
collapse                  =  cp . map cp
```

collapse vytvorí z matice možností, zoznam všetkych potenciálnych riešení

Sudoku.hs

Naivné riešenie

```
Main > collapse (choices easy)
```

??? Koľko ich je ???

```
Main> easy
["2....1.38","........5",".7...6...",".......13",".981..257","
   31....8..","9..8...2.",".5..69784","4..25...."]
Main> map (map (\x->if empty x then 9 else 1)) easy
[[1,9,9,9,9,1,9,1,1],[9,9,9,9,9,9,9,9,1],[9,1,9,9,9,1,9,9,9],[
   9,9,9,9,9,9,9,1,1],[9,1,1,1,9,9,1,1,1],[1,1,9,9,9,9,1,9,9],
   [1,9,9,1,9,9,9,1,9],[9,1,9,9,1,1,1,1,1],[1,9,9,1,1,9,9,9,9]
   ]
Main> (product . map product)
        (map (map (\x->if empty x then 9 else 1)) easy)
4638397686588101979328150167890591454318967698009 ☹

solve                   :: Grid -> [Grid]
solve                   =  filter valid . collapse . choices
```

rows . rows = id
cols . cols = id
boxs . boxs = id

Orezávanie možností

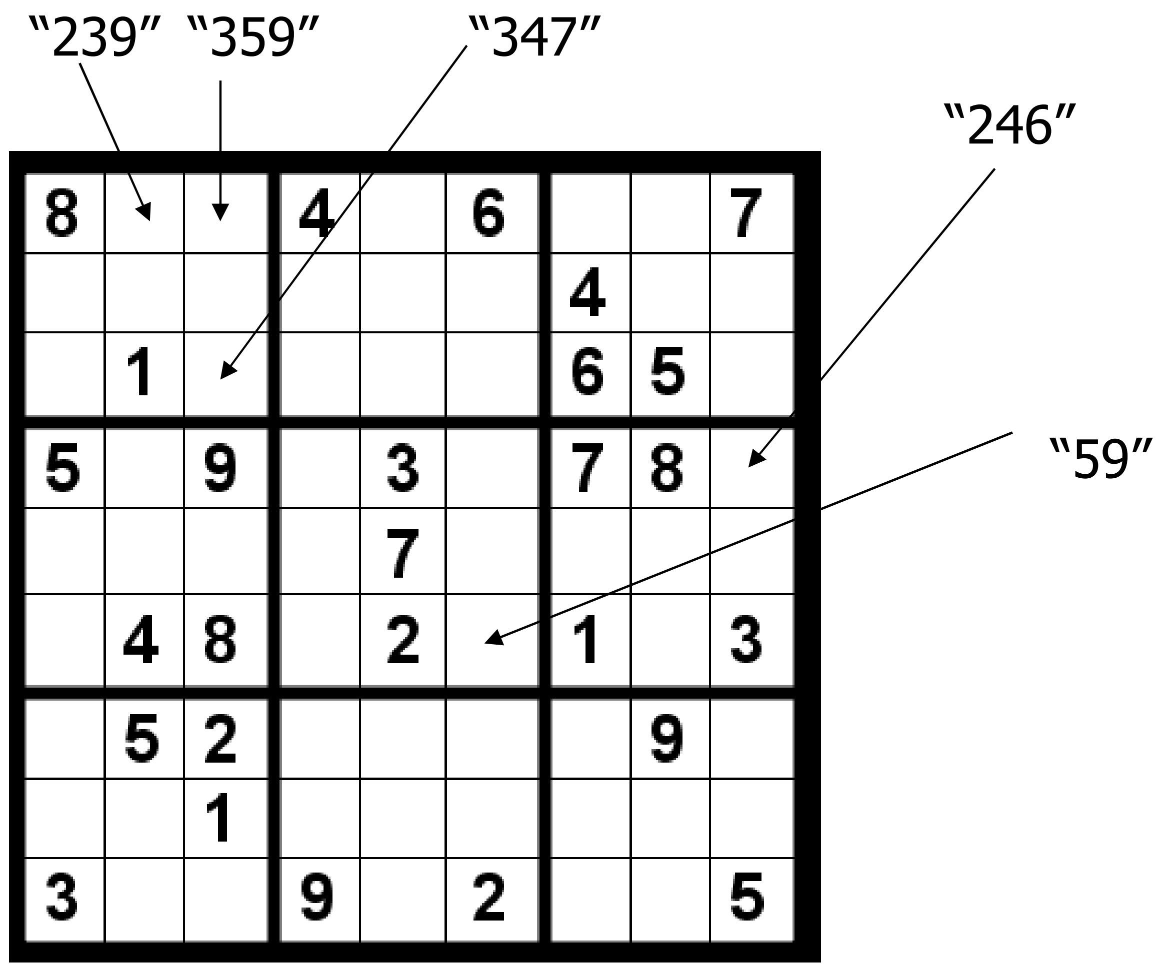
Zredukujme tie možnosti, ktoré sa vylučujú so single-možnosťami

```
prune          :: Matrix Choices -> Matrix Choices
prune          =  pruneBy boxs . pruneBy cols . pruneBy rows
                  where pruneBy f = f . map reduce . f

reduce         :: Row Choices -> Row Choices
reduce xss     =  [xs `minus` singles | xs <- xss]
                  where singles = concat (filter single xss)
                     -- singles zoznam použitých single-možností v riadku

Main> reduce [ "123","2","567","7" ]
["13","2","56","7"]

minus          :: Choices -> Choices -> Choices
xs `minus` ys  =  if single xs then xs else xs \\ ys

solve2         :: Grid -> [Grid]
solve2         =  filter valid . collapse . prune . choices
```

Koľko možností má (prune . choices) grid (napr.easy)?
Definujte funkciu v Haskelli, ktorá to spočíta...

Sudoku.hs

prune.choices



rows . rows = id
cols . cols = id
boxs . boxs = id

Opakované orezávanie

prune . prune ... prune, až kým je čo orezať ...

```
filter valid . collapse . prune . prune ... prune . prune . choices
```

```
solve3          :: Grid -> [Grid]
solve3          =  filter valid . collapse . fix prune . choices

fix             :: Eq a => (a -> a) -> a -> a
fix f x         =  if x == x' then x else fix f x'
                   where x' = f x
```

Koľko možností má (fix prune. choices) pre easy, resp. gentle, ...

Sudoku.hs

Vlastnosti matíc

```
complete   :: Matrix Choices -> Bool          -- matica možností predstavuje
complete   =  all (all single)                -- jediné riešenie

Main> (all (all single)) (choices easy)
False

void       :: Matrix Choices -> Bool          -- neexistuje riešenie, lebo
void       =  any (any null)                  -- niektorá z možností je null

safe       :: Matrix Choices -> Bool          -- konzistencia na singletony
safe m     =  all consistent (rows m) &&      --  na riadkoch
              all consistent (cols m) &&      -- na stĺpcoch
              all consistent (boxs m)         -- v štvorcoch

consistent :: Row Choices -> Bool
consistent =  nodups . concat . filter single
```

Main> consistent ["12", "2", "34", "3", "2"]
False

```
blocked    :: Matrix Choices -> Bool          -- zlá možnosť
blocked m  =  void m || not (safe m)
```

Sudoku.hs

Constraint propagation

```
solve4                    :: Grid -> [Grid]
solve4                    =  search . prune . choices

search                    :: Matrix Choices -> [Grid]
search m
  | blocked m             =  []
  | complete m            =  collapse m
  | otherwise             =  [g | m' <- expand m
                                , g  <- search (prune m')]
```

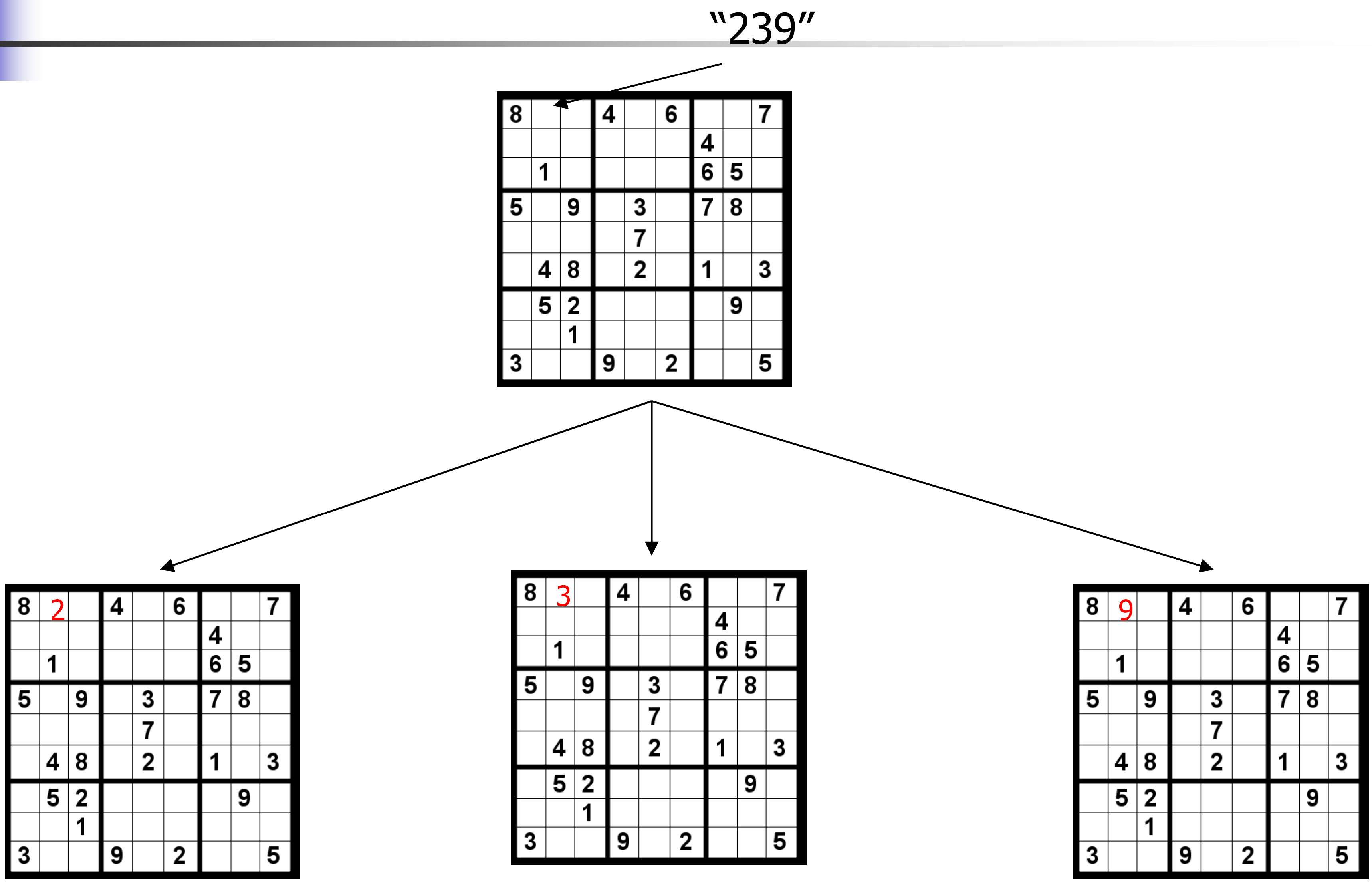
-- zober niektorú/prvú možnosť, ktorá nie je singleton, a rozpíš ju

```
expand                    :: Matrix Choices -> [Matrix Choices]
expand m                  =
   [rows1 ++ [row1 ++ [c] : row2] ++ rows2 | c <- cs]
   where
      (rows1,row:rows2) = break (any (not . single)) m
      (row1,cs:row2)    = break (not . single) row
```

zistite, čo robí break a definujte vlastnú implementáciu

Sudoku.hs

search



Minimum možností

Domáca úloha: upravte expand na

```
expandMin :: Matrix Choices -> [Matrix Choices]
```

ktorá expanduje maticu podľa políčka s minimálnym počtom možností