Wholemeal (functional) programming

Peter Borovanský

I-18

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

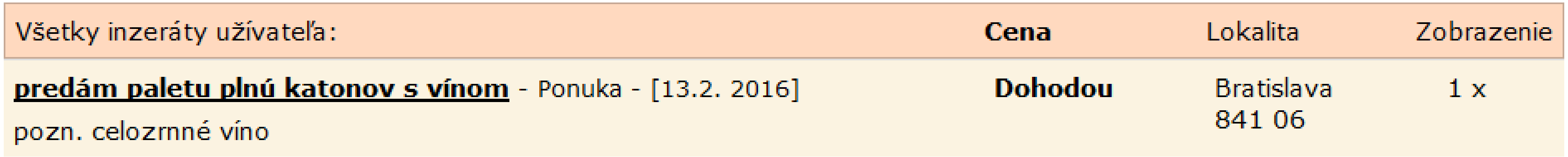
Čo je wholemeal (celozrnné)

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.

Wholemeal programming je štýl rozmýšlania, programovania

Zpráva z tisku: neznámy pachatel odcizil paletu plnou kartónŢ s vínem



Všetky inzeráty užívateľa:	Cena	Lokalita	Zobrazenie
predám paletu plnú katonov s vínom - Ponuka - [13.2. 2016] pozn. celozrnné víno	**Dohodou**	Bratislava 841 06	1 x

... privedie vás k šlachtickým manierom vo funkcionálnom svete

Celozrnný programátor musí

poznať funkcie a najzákladnejšie funkcionály

- map/filter
- foldr/foldl
- scanr/scanl
- iterate
- ...
- ...
- ...
- ...
- ...

Extrémny príklad celozrnného

```
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
```

Krok-po-kroku

- Krok 1 - zbierame párne a nepárne prvky do zoznamov

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

- Krok 2 - prečo nepočítať súčet už hneď

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

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

```
rozdielSuctu"" xs  = uncurry (-) $ foldr (\x -> \(p,n) -> (n,p+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 -> \(p,n) -> (n,p+x)) (0,0)
```

Celozrnné krok-po-kroku

(a na príkladoch)

Čo robí táto funkcia ?

```
foo                          :: [Integer] -> Integer
foo []                        = 0
foo (x:xs)     | odd x        = (3*x + 1) + foo xs
               | otherwise = foo xs
```

Sčíta 3x+1 pre každý prvok x vstupného zoznamu, ale len tie nepárne...

```
foo'  xs  = [ 3*x+1 | x <- xs, odd x] -- už toto je výrazný progres v čitateľnosti

foo''  xs  = sum (map (\x -> 3*x+1) ( filter (odd) xs)) -- to isté len s filter/map
foo''' xs  = sum $ map (\x -> 3*x+1) $ filter (odd) xs -- poznajúc operátor $
foo''''     = sum . map (\x -> 3*x+1) . filter (odd)       -- poznajúc kompozíciu .
foo'''''    = sum . map ((+1).(*3)) . filter (odd)          -- 2xpoznajúc kompozíciu
foo''''''   = foldr (+) 0 . map ((+1).(*3)) . filter (odd)  -- extrémna verzia
```

Celozrnné krok-po-kroku

(a na príkladoch)

Čo robí táto funkcia ?

```
goo                         :: [Integer] -> Integer
goo   []                    = 1
goo   (x:xs)  | even x    = (x-2) * goo xs
              | otherwise = goo xs
```

Vynásobí všetky párne prvky vstupného zoznamu zmenšené o 2

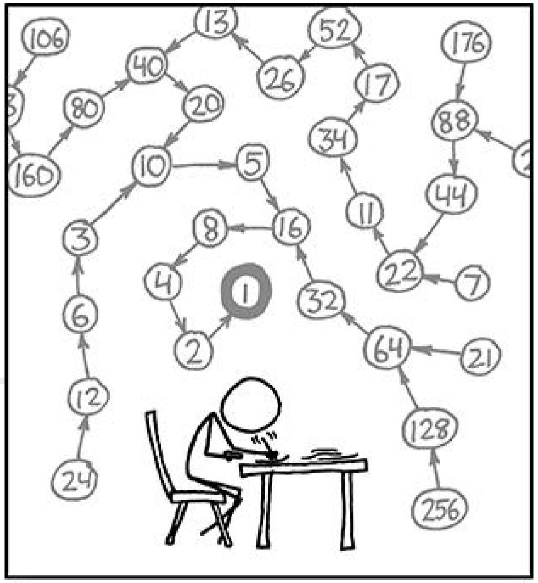
```
goo'  xs  = [ x-2 | x <- xs, even x] -- už toto je výrazný progres v čitateľnosti
```

```
goo ' '    = product . map (subtract 2) . filter (even)
```

Colatz

(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 ?

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

To sú prvky tzn. Colatzovej postupnosti

```
hoo'  = 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**

Celozrnné krok-po-kroku

(a na príkladoch)

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

Čo robí táto funkcia ?

```
moo       :: Integer -> Integer
moo 1     = 0
moo n     | even n              = n + moo (n `div` 2)
          | otherwise           = moo (3 * n + 1)
```

Zrejme je to súčet párnych prvkov tzn. Colatzovej postupnosti,

```
2+(                                            teda   sum . filter (even) . hoo
    snd $
      last $
        takeWhile ((/=1).fst) $
          iterate (\(x,s) -> if even x then (x `div` 2,x+s) else (3 * x + 1,s) )
          (n,0)
    )
moo'' n = snd $ last $ takeWhile ((/=1).fst) $
        iterate (\(x,s) -> if even x then (x `div` 2,x+s) else (3 * x + 1,s) ) (n,2)
```

Cifry

(riešenie Viktor S.)

```
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)
```

0	1	2	3	4	5	6	7
-1	2	1	-3	2	3	-3	1

Maximálny súčet

to

from

	0	1	2	3	4	5	6	7
0	-1	1	2	-1	-1	2	-1	0
1	×	2	3	0	2	5	2	3
2	×	×	1	-2	0	3	0	1
3	×	×	×	-3	-1	2	-1	0
4	×	×	×	×	2	5	2	3
5	×	×	×	×	×	3	0	1
6	×	×	×	×	×	×	-3	-2
7	×	×	×	×	×	×	×	1

$x_{from} + \ldots + x_{to}$

[[-1], [1,2], [2,3,1], [-1,0,-2,-3], [-1,2,0,-1,2], [2,5,3,2,5,2],[-1,2,0,-1,2,0,-3]]

Maximálny súčet

```
maxSucet' :: [Int] -> Int
maxSucet' [] = 0
maxSucet' xs =
      maximum (map (maximum)
             (init (
                 foldl (\t -> \p -> (p:(map (+p) (head t))):t)   [[]]   xs)))

maxSucet'' xs = maximum $ map (maximum) $
             init $ foldl (\t -> \p -> (p:(map (+p) (head t))):t) [[]] xs

maxSucet''' = maximum . map (maximum) .
             init . foldl (\t -> \p -> (p:(map (+p) (head t))):t) [[]]
```

```
maxSucet' [(-1), 2, 1, (-3), 2, 3, 1] == 6
maxSucet'' [(-1), 2, 1, (-3), 2, 3, 1] == 6
```

Kadane Algo

```
kadane :: [Int] -> Int -> Int -> Int -- list -> tempMax -> globalMax -> max
kadane []     _       globalMax = globalMax
kadane (x:xs) tempMax globalMax = kadane xs newTempMax newGlobalMax
    where
          newTempMax = max (tempMax + x) 0
          newGlobalMax = max globalMax newTempMax

kadane' :: [Int] -> Int
kadane' (x:xs) = snd $ foldr f (0,0) xs
   where f x (tempMax, globalMax) = let newTempMax = max (tempMax + x) (
         in (newTempMax, max globalMax newTempMax)

kadane [(-1), 2, 1, (-3), 2, 3, 1] 0 0 == 6
kadane' [(-1), 2, 1, (-3), 2, 3, 1] == 6
```

Kadane.hs

Najčastejšie vyskytujúce slovo

Nájdi najčastejšie vyskytujúcie sa slovo v reťazci

```
-- rozdeľ na slová podľa oddelovača, viac pozri Data.List.Split
splitOneOf :: String -> String -> [String]
splitWords  = filter(/= "") . splitOneOf " .,;!@#$%^&*()'"

chunks          :: [String] -> [[String]]
chunks [] = []
chunks xs@(w:_) = takeWhile (==w) xs: chunks (dropWhile (==w) xs)

type FreqTable = [(Int,String)]
chunkLengths    :: [[String]] -> FreqTable
chunkLengths xs = map (\chunk -> (length chunk, head chunk)) xs
```

Najčastejšie vyskytujúce slovo

```
mostFrequent :: String -> String
mostFrequent ws =
    snd $ last $ sort $ chunkLengths $ chunks $ sort $ splitWords $ map toLower ws

-- funkcionálna verzia
mostFrequent' =
    snd .last . sort . chunkLengths . chunks . sort . splitWords . map toLower

-- zátvorková verzia pre rodených Lispistov
mostFrequent'' ws =
                snd (
                   last (
                     sort (
                       chunkLengths (
                         chunks (
                           sort (
                             splitWords (
                               map toLower ws
                   )))))))
```

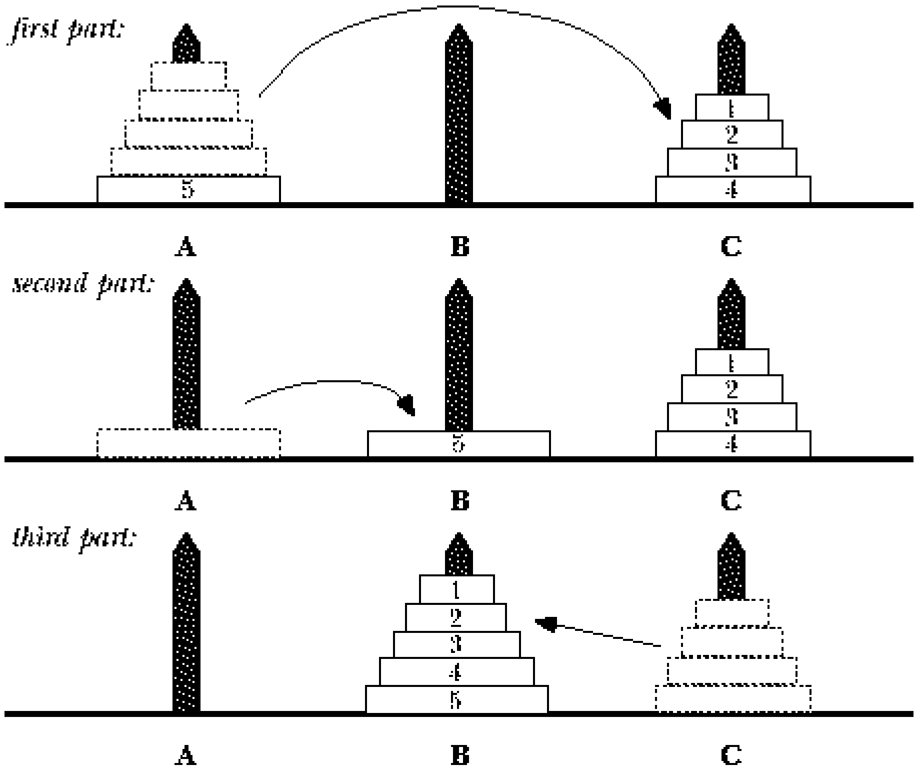
Vstupný text:

```
hamlet = "There was this king sitting in his garden all alane " ++
  "When his brother in his ear poured a wee bit of henbane. " ++
  "He stole his brother's crown and his money and his widow. " ++
  "But the dead king walked and got his son and said Hey listen, kiddo! " ...
```

Kartézsky súčin

Inšpirácie a zdroje

- Ralf Hinze: Functional Pearl: La Tour d'Hanoi, http://www.cs.ox.ac.uk/ralf.hinze/publications/ICFP09.pdf



Wholemeal in functional

- (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."

http://www.cs.tufts.edu/~nr/comp150fp/archive/richard-bird/sudoku.pdf

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
                          -- [[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
```

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
```

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
                            pack   = split . map split
                            split  = chop boxsize
                            unpack = map concat . concat

chop                  :: Int -> [a] -> [[a]]
chop n []             =  []
chop n xs             =  take n xs : chop n (drop n xs)
```

Domáca úloha

Domáca úloha: 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]]
```

```
boxs          :: Matrix a -> [Row a]
boxs          =  unpack . map cols . pack
                 where
                   pack   = split . map split
                   split  = chop boxsize
                   unpack = map concat . concat
```

Boxs – krok 1

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

Main > (split . map split) e                              -- iný zápis pre split (map split 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          :: Matrix a -> [Row a]
boxs          =  unpack . map cols . pack
                 where
                   pack   = split . map split
                   split  = chop boxsize
                   unpack = map concat . concat
```

Boxs – krok 2

```
Main > ((map cols ) . (split. map split)) 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

```
boxs          :: Matrix a -> [Row a]
boxs          =  unpack . map cols . pack
                 where
                   pack   = split . map split
                   split  = chop boxsize
                   unpack = map concat . concat
```

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

```
Main > ((map concat . concat) . (map cols ) . (split. map split)) 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 ] ]
```

concat . split = id
split . 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 . (split . map split) .   -- pokračuje nižšie
(map concat . concat) . map cols . (split . map split) =
```

asociatívnosť

```
map concat . concat . map cols . split . map split .
map concat . concat . map cols . split . map split =

map concat . concat . map cols . split .
concat . map cols . split . map split =

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

Domáca úloha: dokážte, vyvráťte: split . concat = id

Na príklade

```
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 split 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> (split.map split) 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).(split.map split)) 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).(split.map split)) 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).(split.map split)) 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]]
```

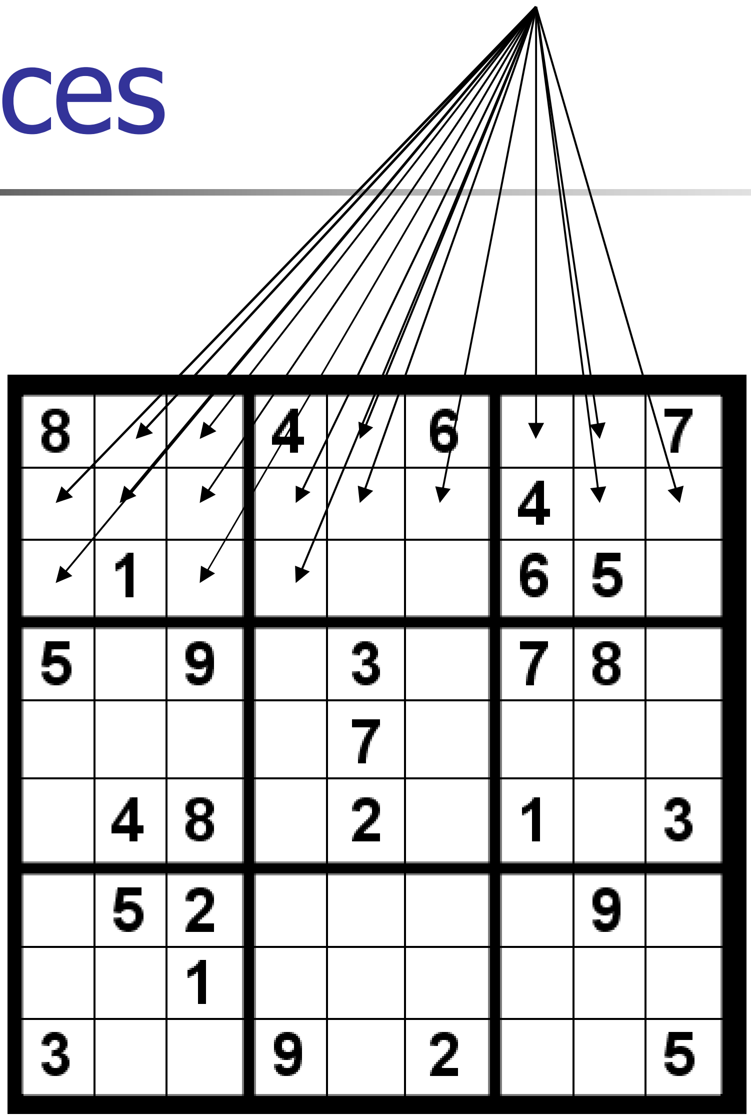
Nájdenie všetkých riešení

```
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"]]
```

Choices

"123456789"



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

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

Najivné 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

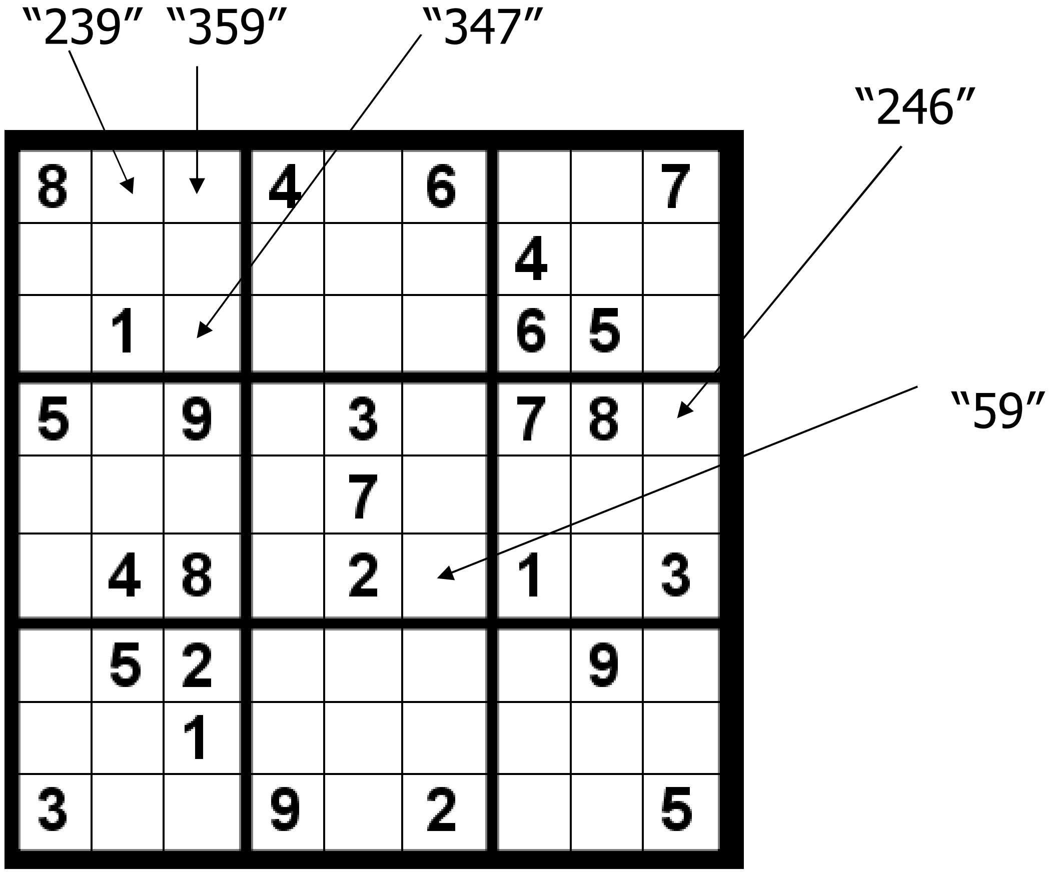
Orezávanie možností

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
```

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

prune.choices



"239" "359" "347"

"246"

"59"

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

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

Opakované orezávanie

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

```
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
```

Domáca úloha:

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

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)
```

Constraint propagation

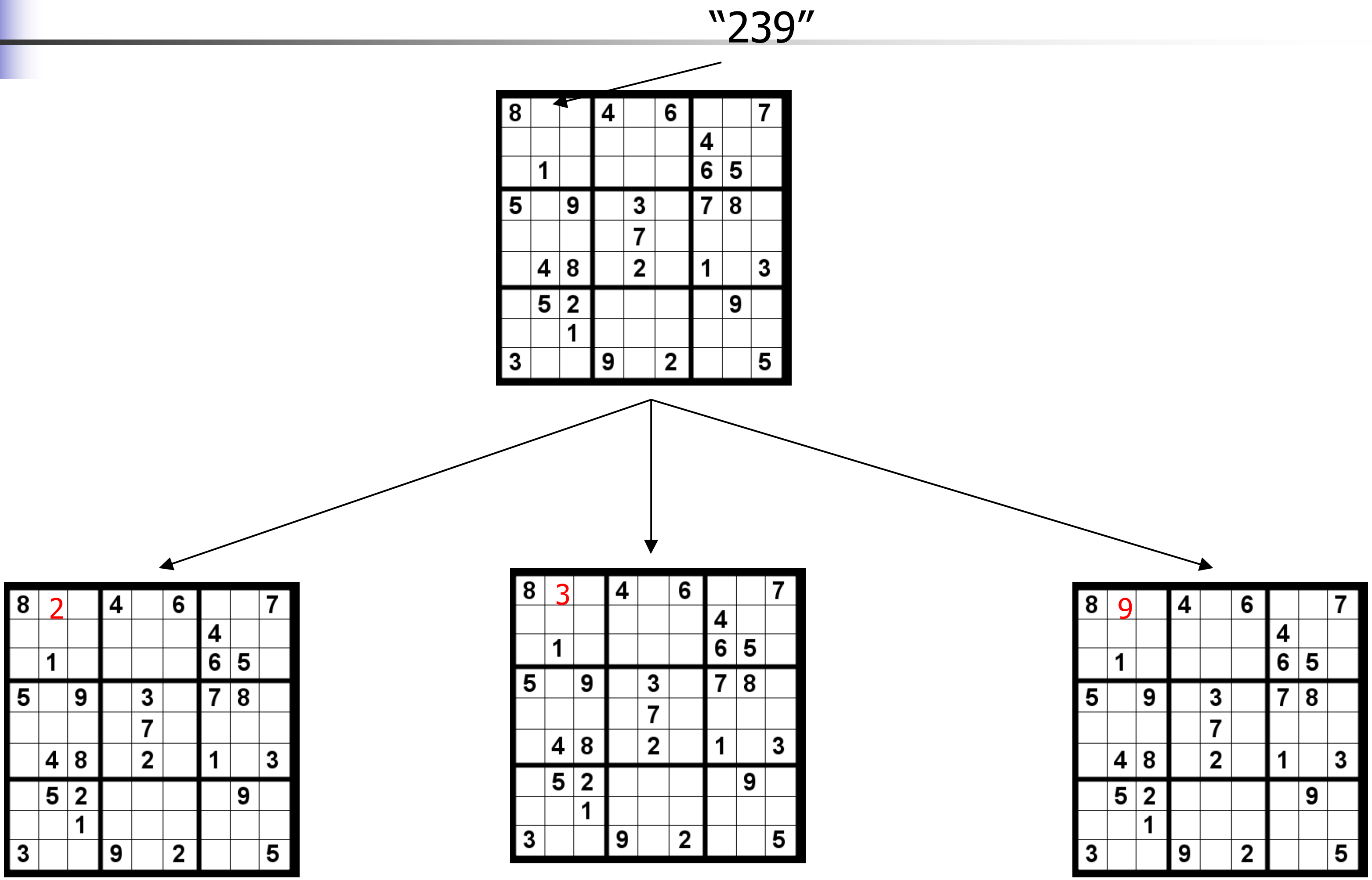
```
solve4                      :: Grid -> [Grid]
solve4                      =  search . prune . choices

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
```

Domáca úloha: zistite, čo robí break a definujte vlastnú implementáciu

search



"239"

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

8	2		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

8	3		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

8	9		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

Minimum možností

Domáca úloha: upravte expand na

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

ktorá expanduje maticu podľa políčka s minimálnym počtom možností