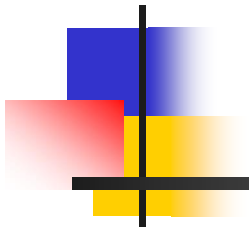
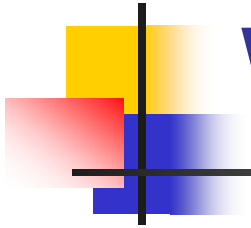


Wholemeal (functional) programming

Peter Borovanský

I-18

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



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,

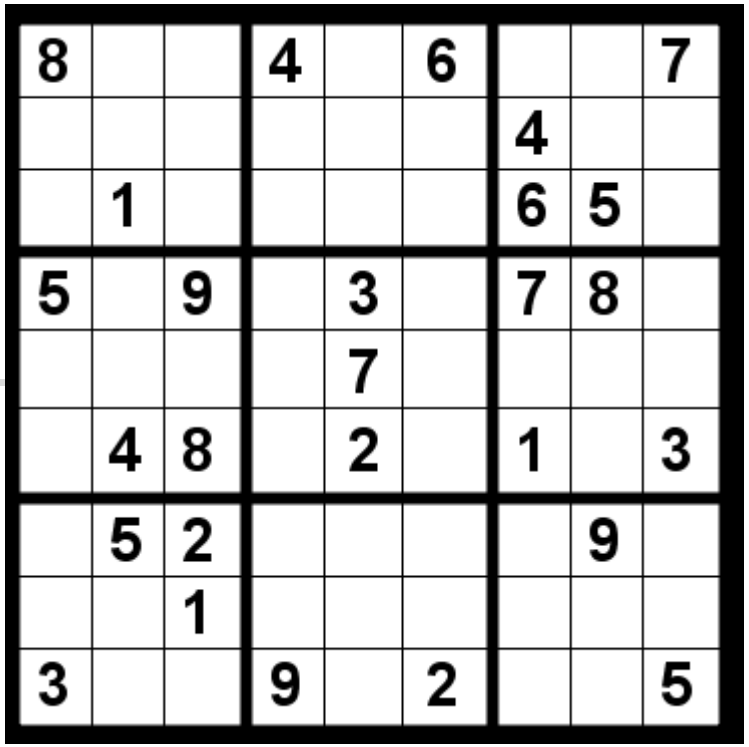
then

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

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

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

Sudoku




```
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

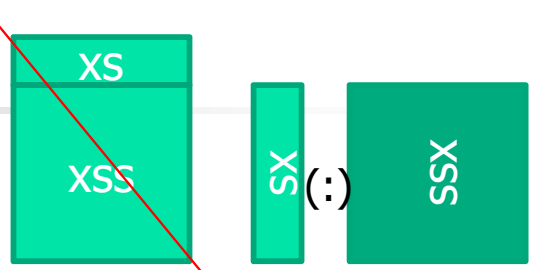
Základné definície

```
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
```

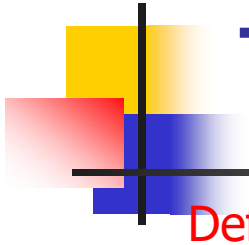
- a funguje to ?
- vieme napísať transpose pomocou foldl

Sudoku.hs

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



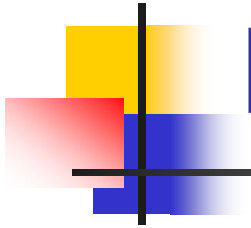
Turbo - SudokuStvorce

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]]
```

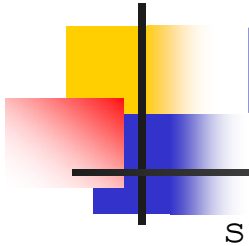


Riešenie s indexovaním

```
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:

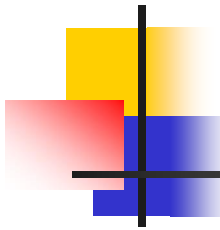
```
blocks :: [[Int]]
blocks = [[0..2],[3..5],[6..8]]
sudokuStvorce :: [[Int]] -> [[Int]]
sudokuStvorce xs = [
       [xs!!r!!c | r <- b1, c <- b2 ] | b1<-blocks, b2<-blocks ]
```



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]
```

```
--
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

(krok 2 – map cols)

```
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 > ((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)

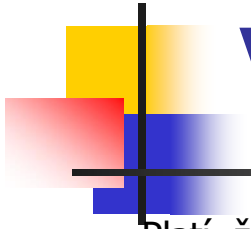
```
boxs  =  unpack . map cols . pack
         where
          unpack = map concat . concat
          pack   = group3 . map group3
          group3  = group boxsize
          group   :: Int -> [a] -> [[a]]
          group n [] = []
          group n xs = take n xs : group n (drop n xs)
```

```
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 ] ]
```

```
[[ [[[ 11,, 22,, 33 ]],, [[ 1100,,1111,,1122 ]],, [[ 1199,,2200,,2211 ]]]],,
    [[ 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:

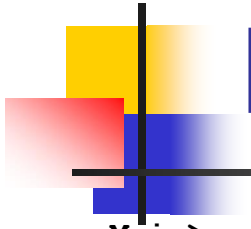
```
(map concat . concat) . map cols . (group3 . map group3) .
(map concat . concat) . map cols . (group3 . map group3) =
```

-- pokračuje nižšie

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]]`

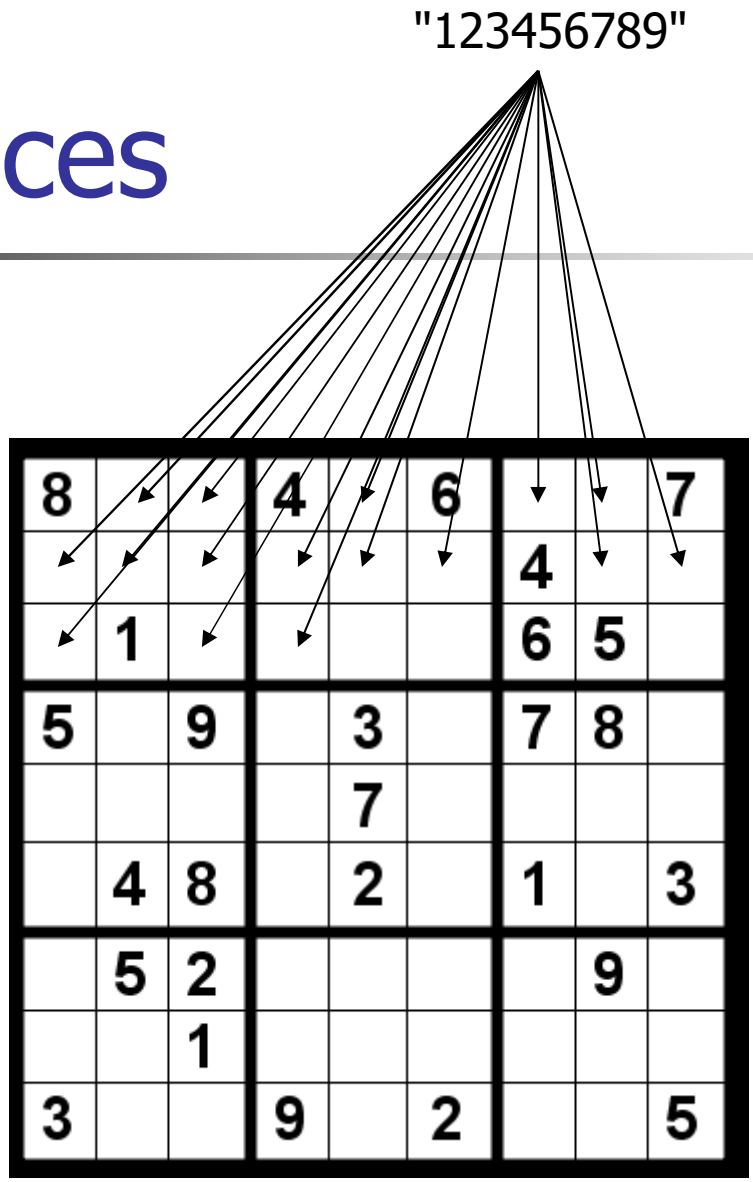
Nájdenie všetkých riešení

```
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

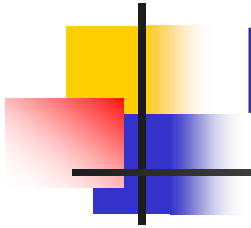
Choices



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

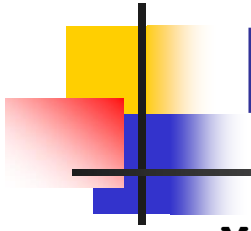
```
-- kartézsky súčin všetkých možností v jednom riadku
cp                      :: [[a]] -> [[a]]      --   Row[a] -> Row[a]
cp []                   =  [[]]
cp (xs:xss)             =  [y:ys | y<-xs, ys<-cp xss]

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)

```
46383976865881019793281501678905914543189676980O9 ☹
```

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

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

Orezávanie možností

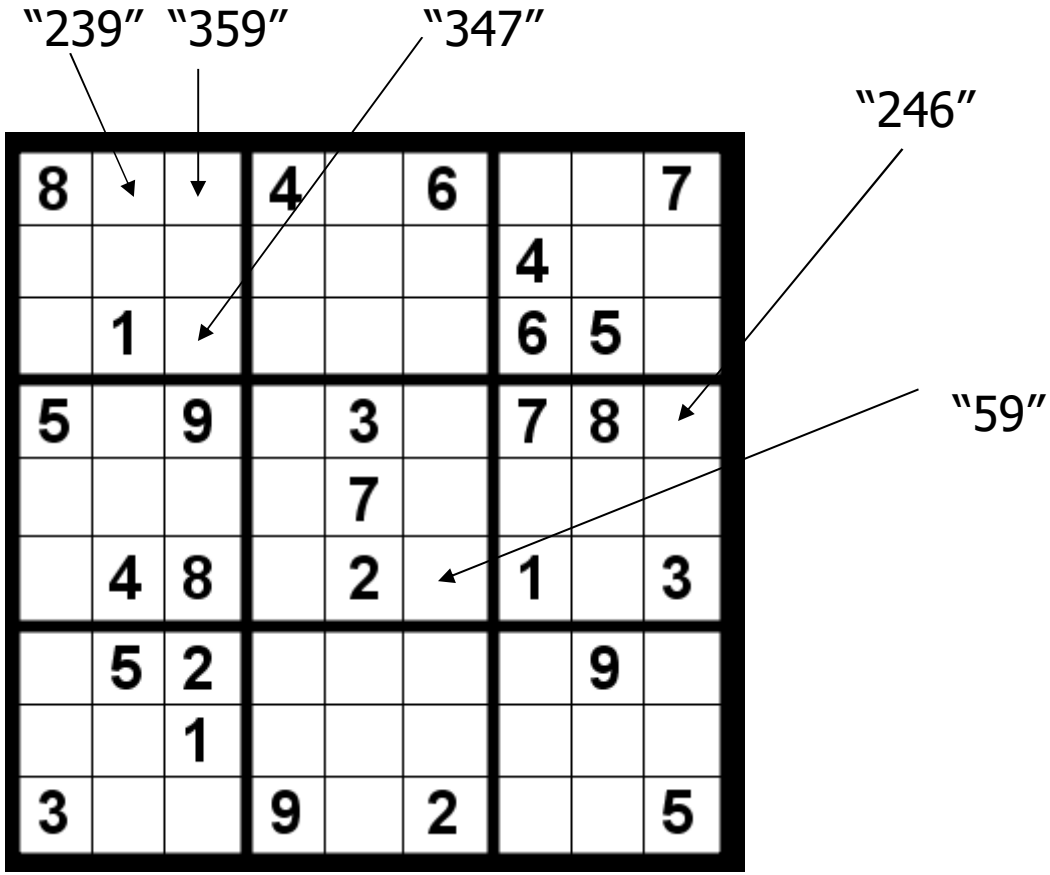
Zredukujme tie možnosti, ktoré sa vylučujú so single-možnosťami

```
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')]
```

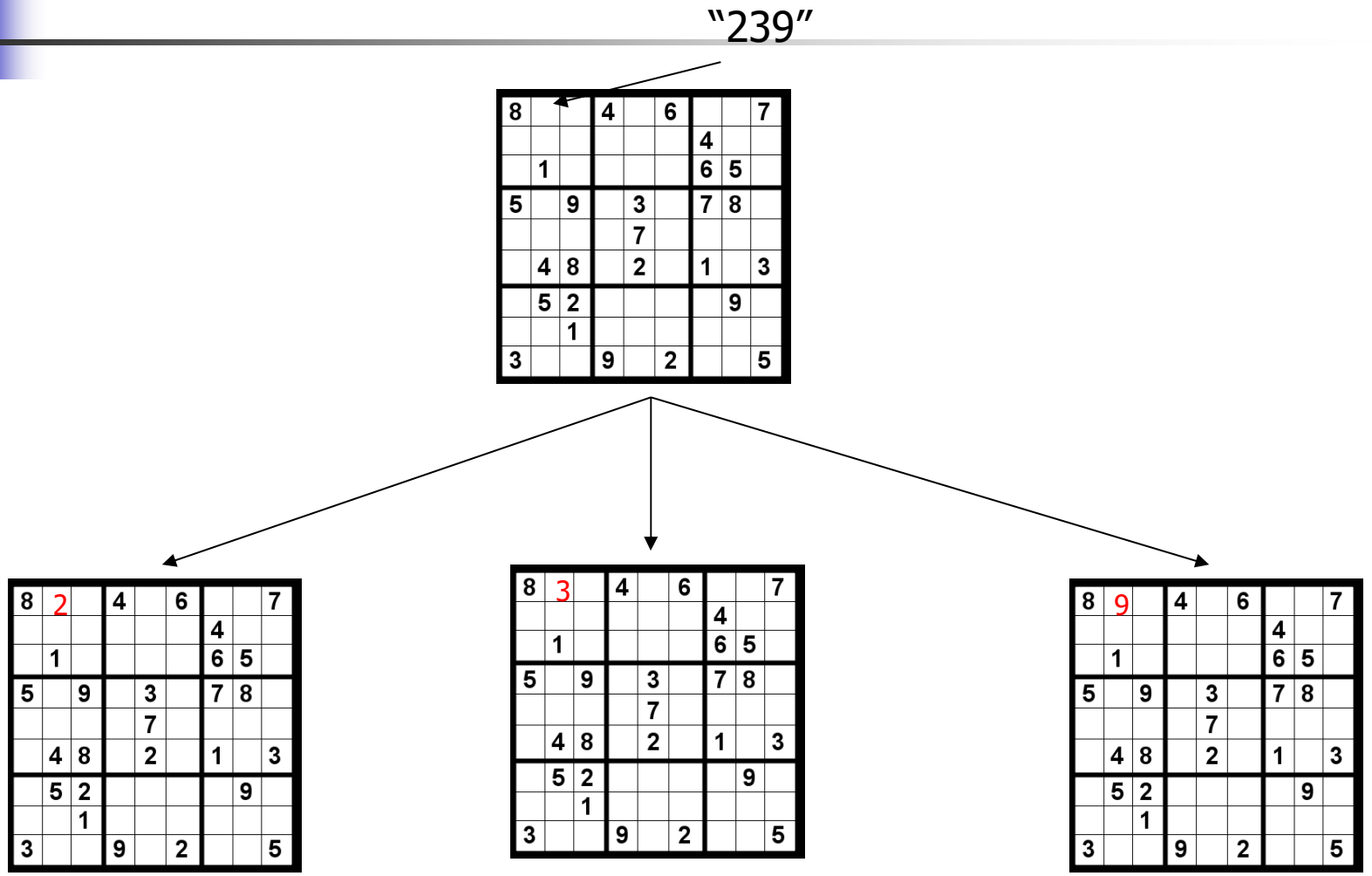
-- zober niektorú/prvú možnosť, ktorá nie je singleton, a rozpíš ju

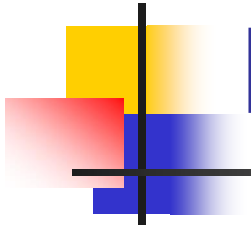
```
expand                  :: Matrix Choices -> [Matrix Choices]
expand m                =
   [rows1 ++ [row1 ++ [c] : row2] ++ rows2 | c <- cs]
   where
      (rows1,row:rows2) = break (any (not . single)) m
      (row1,cs:row2)    = break (not . single) row
```

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