Universiteit van Amsterdam

Opgave 2

Erik Kooistra en Ana Oprescu

Sudoku

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

Deadline: Individueel: woensdag 12 februari 2020, 16:00
Team: zondag 16 februari 2020, 18:00

1 Introductie en puzzeltjes (Individueel)

Voordat je begint met het grotere Haskell-programma dat je deze week gaat schrijven, zijn er wat kleinere puzzeltjes die je moet oplossen. Dit vragen we niet om je te pesten en ook niet om te kijken hoe goed je stukken code op het internet kunt zoeken, maar omdat je hierdoor wat handigheid met en begrip over Haskell zult ontwikkelen. Op het hoorcollege is behandeld dat er in Haskell enkele standaardfuncties zitten die de taal krachtiger maken en nu ga je deze functies ook echt toepassen. Hieronder staan elf functies die met behulp van deze standaardfuncties gedefinieerd kunnen worden, doe dit en gebruik je opgedane kennis bij de rest van de opdracht.

1. Definieer `length` in termen van `foldr` en noem dit `length'`.
2. Kom erachter wat `or` doet. Definieer je eigen versie in termen van `foldr` en noem dit `or'`.
3. Definieer `elem x` in termen van `foldr`[1] en noem dit `elem'`.
4. Definieer `map f` in termen van `foldr` en noem dit `map'`.
5. Definieer `(++)` in termen van `foldr` en noem dit `plusplus`.
6. Definieer `reverse` in termen van `foldr` en noem dit `reverseR`.
7. Definieer `reverse` in termen van `foldl` en noem dit `reverseL`.
8. (expert) Definieer `(!!)` in termen van `foldl`.
9. Maak een functie `isPalindrome` die bepaalt of een string (of lijst) een palindroom is.
10. Maak een functie, genaamd `fibonacci`, die een oneindige lijst met de Fibonacci reeks teruggeeft in termen van `scanl`.
11. (expert)[2] Gegeven de volgende type definitie `type List = (Int) -> Int`, schrijf een functie die het mogelijk maakt op elementen toe te voegen aan dit type

Lever deze functies in als de module `Puzzles`, door een bestand "`Puzzles.hs`" aan te maken met bovenaan:

```
module Puzzles

where
```

gevolgd door jouw uitwerkingen, **met de juiste benaming!** Anders falen de tests en haal je onvoldoende. Op canvas, onder modules, staat een `tar` bestand, genaamd: `HaskellAssignment.tar.gz`, daarin kan je tests vinden om lokaal te gebruiken. Lees ook even Sectie 3 en Sectie 5.

[1] Om dit te laten werken moet je Haskell hinten dat de parameter in de Eq-typeclass zit, bijvoorbeeld: `elem' :: Eq a => a -> [a] -> Bool`

[2] Hele goede oefening voor de team opdracht.

2 Sudoku Team

De Sudoku is een algemeen bekende puzzel; achtergrondinformatie en regels erover zijn makkelijk online te vinden. De bedoeling is dat je een algemene Sudoku solver implementeert in het bestand `SudokuSolver.hs`. Hiervoor heb je het bestand `Sudoku.hs` nodig, deze kan je vinden in het `tar` bestand, genaamd: `HaskellAssignment.tar.gz`, te vinden op Canvas onder modules.
Belangrijk: je mag niks verwijderen uit of toevoegen aan het `Sudoku.hs` bestand, aangezien we onze eigen versie zullen gebruiken bij de tests. Ook is het niet mogelijk om de types in `Sudoku.hs` in je eigen bestand te definiëren, want dan ontstaat er een 'naming collision' en falen je tests. Als alles lokaal goed gaat, zal het ook op Codegrade goed gaan.

```haskell
module SudokuSolver

where

import Sudoku
import Data.List
import Data.List.Split
import Data.Maybe


positions, values :: [Int]
positions = [1..9]
values    = [1..9]

blocks :: [[Int]]
blocks = [[1..3], [4..6], [7..9]]

showDgt :: Value -> String
showDgt 0 = " "
showDgt d = show d

showSubgridRow:: [Value] -> String
showSubgridRow = unwords . map showDgt

showRow :: [Value] -> String
showRow sr =
  "| "
  ++
  intercalate " | " (map showSubgridRow $ chunksOf 3 sr)
  ++
  " |"


showGrid :: Grid -> String
showGrid grid =
  "+-------+-------+-------+\n"
  ++
  intercalate "\n+-------+-------+-------+\n" rows
  ++
  "\n+-------+-------+-------+"
  where rows = map (intercalate "\n") $ chunksOf 3 $ map showRow grid


sud2grid :: Sudoku -> Grid
```

```haskell
sud2grid s =
  [ [ s (r,c) | c <- [1..9] ] | r <- [1..9] ]


grid2sud :: Grid -> Sudoku
grid2sud gr = \(r,c) -> pos gr (r,c)
  where pos :: [[a]] -> (Row,Column) -> a
        pos gr (r,c) = (gr !! (r - 1)) !! (c - 1)


printSudoku :: Sudoku -> IO()
printSudoku = putStrLn . showGrid . sud2grid
```

De gedefinieerde types kan je vinden in het bestand: `Sudoku.hs`.

Het doel van de solver is een sudoku als een Grid accepteren en een opgeloste sudoku, als een Grid, teruggeven. Een voorbeeld van zo'n Grid is als volgt.

```haskell
example1 :: Grid
example1 =
    [ [5, 3, 0, 0, 7, 0, 0, 0, 0]
    , [6, 0, 0, 1, 9, 5, 0, 0, 0]
    , [0, 9, 8, 0, 0, 0, 0, 6, 0]
    , [8, 0, 0, 0, 6, 0, 0, 0, 3]
    , [4, 0, 0, 8, 0, 3, 0, 0, 1]
    , [7, 0, 0, 0, 2, 0, 0, 0, 6]
    , [0, 6, 0, 0, 0, 0, 2, 8, 0]
    , [0, 0, 0, 4, 1, 9, 0, 0, 5]
    , [0, 0, 0, 0, 8, 0, 0, 7, 9]
    ]
```

De eerste stap is het bepalen welke mogelijkheden er zijn gegeven een rij (row), kolom (column) en een blok (subgrid). Daarnaast hebben we nog een lijst nodig van alle lege posities in de sudoku. Hiervoor zijn de volgende functies nodig.

1. `extend :: Sudoku -> (Row,Column,Value) -> Sudoku`
2. `freeInRow :: Sudoku -> Row -> [Value]`
3. `freeInColumn :: Sudoku -> Column -> [Value]`
4. `freeInSubgrid :: Sudoku -> (Row,Column) -> [Value]`
5. `freeAtPos :: Sudoku -> (Row,Column) -> [Value]`
6. `openPositions :: Sudoku -> [(Row,Column)]`

Nu we kunnen uitrekenen wat de mogelijkheden zijn voor elke vrije plek in de sudoku moeten we ook kunnen controleren of de sudoku geldig is voor de gegeven oplossing.

1. `rowValid :: Sudoku -> Row -> Bool`
2. `colValid :: Sudoku -> Column -> Bool`
3. `subgridValid :: Sudoku -> (Row,Column) -> Bool`
4. `consistent :: Sudoku -> Bool`

2.1 Sudoku oplossen

Met al deze verschillende functies hebben we nu voldoende informatie om te gaan beginnen aan het oplossen van een sudoku, dit gaan we doen door middel van een zoekboom. Maar eerst moeten we daarvoor nog wat types definiëren. Een Constraint heeft als waarde een x, y coördinaat in de sudoku en een lijst met mogelijke opties die daar kunnen. De benodigde types zijn al gedefinieerd in het bestand `Sudoku.hs`, maar zien er als volgt uit:

```
type Constraint = (Row,Column,[Value])
type Node = (Sudoku,[Constraint])
```

De volgende functie kan handig zijn tijdens het debuggen: `printNode :: Node -> IO() printNode = printSudoku . fst`

De eerste stap in het oplossen, is een lijst genereren van alle mogelijke Constraints gesorteerd van minste opties naar meeste opties. De functie definitie daarvoor is als volgt
`constraints :: Sudoku -> [Constraint]`

Met al deze informatie is het nu mogelijk om de solve functie te schrijven. Deze heeft als definitie `solveSudoku :: Sudoku -> Maybe Sudoku`

Hoe je dit het beste kan implementeren is als eerste een lijst van initial constraints te maken, en dan per mogelijkheid verder zoeken naar nieuwe mogelijkheden tot je niet meer verder kan.

Het is handig om een helper te schrijven die er als volgt uit ziet:
`solveAndShow :: Grid -> IO()`

3 Belangrijk

Houd je aan de naamgeving die in de opdracht staat, anders falen je tests en haal je geen voldoende. Ook mag er geen gebruik gemaakt worden van Haskell compiler 'directives' (dit zijn comments tussen `{-# #-}`). Als laatste is het belangrijk dat je geen `main` functie in je `SudokuSolver.hs` of `Puzzles.hs` bestand plaatst. Testen van je programma kan je doen via `ghci` of met de aangeleverde tests.

4 Tools

Jullie uitwerkingen worden qua werking automatisch nagekeken. Ook, wordt een deel van jullie stijl en idiomatische code automatisch nagekeken. Voor stijl en idiomatische code geldt: 'suggestions' worden door de TAs handmatig nagekeken en voor de andere fouten gaan er automatisch punten af. Op een enkele keer na is het ook gewoon noodzakelijk om de 'suggestions' te verbeteren. Oftewel, verbeter ook de 'suggestions'.

Dit doen we met behulp van de volgende programma's:

- Tests
  - tasty
  - tasty-hunit
  - tasty-quickcheck
  - tasty-smallcheck
- Stijl & idiomatiek
  - hlint

Om `Tasty` en friends te installeren heb je `stack` nodig, zie: `https://docs.haskellstack.org/en/stable/README/#how-to-install`. Als je stack geïnstalleerd hebt kan je Tasty en friends installeren met de volgende shell commando's:

```
stack install tasty-1.2.3
stack install tasty-hunit-0.10.0.2
stack install tasty-smallcheck-0.8.1
stack install tasty-quickcheck
```

Hlint kan je installeren via Ubuntu: `sudo apt install hlint`. Gebruik je geen ubuntu 18.04? Dan kan je hlint via stack installeren, de versie die gebruikt wordt is: **2.0.11**.

5 Inleveren

Voor de individuele opdracht **moet** je alleen het bestand `Puzzles.hs` inleveren en voor de team opdracht **moet** je alleen het bestand `SudokuSolver.hs` inleveren! Nogmaals, houd je aan de bestandsnamen en de naamgeving in de opdracht.