Universiteit van Amsterdam

Programmeertalen

Haskell

Erik Kooistra, Ana Oprescu en Damian Frölich

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: donderdag 24 februari 2022, 23:59

1 Puzzels

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`.

Je kan je implementatie testen door het op Codegrade in te leveren, zorg er dan voor dat je bestand Puzzels.hs heet, begint met de volgende regel: *module Puzzels where* en dat de namen van de functies overeenkomen met wat er in dit document staat.

[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 Sudoku

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. De solver zal een sudoku file inlezen, deze omzetten naar een `Sudoku`, de sudoku oplossen en de oplossing printen.

De eerste stap is het bepalen welke mogelijkheden er zijn gegeven een rij (row), kolom (column), blok (subgrid) en positie (pos). Hiervoor zijn de volgende functies nodig:

1. `freeInRow :: Sudoku -> Row -> [Value]`: Voor een gegeven sudoku en een gegeven rij, geeft deze functie de sudoku waarden die nog niet in de rij zitten.
2. `freeInColumn :: Sudoku -> Column -> [Value]`: Voor een gegeven sudoku en een gegeven kolom, geeft deze functie de sudoku waarden die nog niet in de kolom zitten.
3. `freeInSubgrid :: Sudoku -> (Row, Column) -> [Value]`: Voor een gegeven sudoku en een gegeven blok — bepaald door de rij en kolom paar — geeft deze functie de sudoku waarden die nog niet in het blok zitten.
4. `freeAtPos :: Sudoku -> (Row,Column) -> [Value]`: Voor een gegeven sudoku en een gegeven positie — bepaald door de rij en kolom paar — geeft deze functie de sudoku waarden die nog ingevuld kunnen worden op de gegeven positie.

Daarnaast hebben we nog een lijst nodig van alle lege posities in de sudoku. Hiervoor is de volgende functie nodig:

- `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 een oplossing geldig is.

1. `rowValid :: Sudoku -> Row -> Bool`: Kijkt of de gegeven rij voldoet aan de regels voor een rij.
2. `colValid :: Sudoku -> Column -> Bool`: Kijkt of de gegeven kolom voldoet aan de regels voor een kolom.
3. `subgridValid :: Sudoku -> (Row,Column) -> Bool`: Kijkt of het gegeven blok voldoet aan de regels voor een blok.
4. `consistent :: Sudoku -> Bool`: Kijkt of een sudoku voldoet aan alle regels.

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.

```
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 -> 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.

2.2 Ander type sudoku

Er zijn ook variaties op de orginele 9x9 sudoku, zoals de sudoku van de NRC[2]. Breid je sudoku solver uit om ook NRC sudoku's op te lossen. Zorg ervoor dat je eerder gebruikte functies hergebruikt, maak dus niet 2 aparte solvers. Om dit netjes te doen is het handig om een Solver data type te maken die de functies voor een Solver bevat. Deze Solver zal je dan moeten meegeven als argument aan bepaalde functies. Mits je dit met de Solver data type gaat doen, moet je ook een Solver voor de normale sudoku definieren. Om aan te geven dat de NRC solver gebruikt moet worden, zal het tweede argument de string *nrc* zijn. De volgende code snippet kan je gebruiken om de argumenten te parsen en de juiste Solver te kiezen:

```
getSolver :: [String] -> Solver
getSolver (_:"nrc":_) = nrcSolver
getSolver _ = normalSolver
```

De `normalSolver` en `nrcSolver` zijn functies die de bijbehorende Solver returnen.

3 Tests & Tools

De 'functionaliteit' categorie wordt automatisch nagekeken en daarvoor wordt pytest[3] gebruikt. **Let op:** er zijn ook tests die pas zichtbaar worden na de deadline. Je moet er dus voor zorgen dat je solver werkt voor elke sudoku en niet alleen voor de tests die zichtbaar zijn.

Onoplosbare sudoku moeten een exit code die niet gelijk is aan 0 teruggeven en dit kan je doen met behulp van **error**.

Ook wordt idiomatiek en stijl deels automatisch nagekeken, zie de **hlint** categorie in de rubric, met behulp van hlint en een checker die kijkt naar trailing whitespace. Voor hlint geldt: 'suggestions' worden door de TAs handmatig bekeken en voor de andere fouten gaan er automatisch punten af in de **hlint** categorie.

[2] `https://www.nrc.nl/rubriek/sudoku/`
[3] `https://docs.pytest.org/en/stable/getting-started.html`