Functioneel Programmeren

Pieter De Clercq

Inhoudsopgave

Hoofdstuk 1

Inleiding

test

Hoofdstuk 2

Syntaxis

Hoofdstuk 3

Semantische constructies

Hoofdstuk 4

Programma's

Fibonacci

Dit programma print de Fibonacci getallen tot het afgesloten wordt. Aangezien het zo snel ging en na één enkele seconde reeds Infinity bereikte, heb ik het vertraagd met een cook-statement. COURSES/PROGRAMS/FIBONACCI.COURSE

MBot: Politiewagen

Laat de LEDs afwisselend rood en blauw knipperen, zoals een zwaailicht van een politiewagen of een brandweerauto. Als dit te snel zou gaan zou dit niet aangenaam zijn en tevens aanleiding kunnen geven tot epilepsie, waardoor ik hier ook een vertraging heb ingebouwd. COURSES/ALGORITHMS/POLICECAR.COURSE

MBot: Lijnvolger

De lijnvolger werkt volgens een zeer eenvoudig principe. Zolang beide sensoren zwart zien, rijdt de robot rechtdoor. Wanneer de linkse of rechtse sensor wit ziet, wordt respectievelijk naar rechts of links gereden. Wanneer beide sensoren wit zien en de robot dus de weg kwijt is, wordt achteruit gereden en gedraaid naar de kant waar het het laatst wit was. COURSES/ALGORITHMS/LINEFOLLOW.COURSE

MBot: Obstakels ontwijken

De obstakel ontwijker is zeer intuïtief geprogrammeerd. Er wordt namelijk altijd rechtdoor gereden, tot dit niet meer mogelijk is (de ultrasone sensor detecteert een obstakel), waarna telkens naar dezelfde kant wordt gedraaid, namelijk rechts. Om zeker te zijn dat er altijd kan gedraaid worden, zal de robot eerst een stuk achteruit rijden als deze te dicht bij het obstakel zou staan. COURSES/ALGORITHMS/OBSTACLE_AVOID.COURSE

Hoofdstuk 5

Implementatie

-Alle getallen zijn Doubles

Alle getallen worden voorgesteld als Doubles. Dit zorgt ervoor dat ik nergens rekening moet houden met gehele getallen of kommagetallen, aangezien elk geheel getal kan worden voorgesteld als een kommagetal maar niet omgekeerd. LEXER.HS: 162-166.

Whitespace

Voor aanvang van het parsen wordt alle whitespace, dit zijn zowel spaties, tabs alsook newlines, verwijderd uit de code die geparset moet worden. Dit maakt het niet alleen sneller, het maakt ook de code een stuk aangenamer en duidelijker om te lezen. Bovendien vormde dit toch geen probleem aangezien strings niet ondersteund worden. LEXER.HS: 100-102.

```
-- |Preprocesses a file for parsing, removing all whitespace.
preprocess :: String -> String
preprocess = filter (`notElem` " \t\n\r")
```

Modulariteit

Om het project wat overzichtelijk te houden heb ik besloten niet alles in één groot bestand te plaatsen, maar het op te splitsen naar verschillende bestanden. Vooraleer ik de State monad geïmplementeerd had, had ik een simpele environment gemaakt bestaande uit een Map. Hiervoor had ik nog een bestand Environment.hs, maar dat is nu volledig weg aangezien het maar een lijn of 10 meer bevatte.

Booleans zijn Aritmetische expressies

Booleans worden bij het parsen onmiddellijk omgezet van tasty en disguisting naar respectievelijk de getallen 1 en 0. Dit naar analogie met de implementatie van booleans in C. De omgekeerde logica geldt ook, namelijk elk getal dat niet 0 is heeft booleaanse waarde tasty (true) en 0 wordt disguisting (false). Dit is geïnspireerd op de programmeertaal Python. LEXER.HS: 107-110.

```
-- |Tokenizes a boolean expression.
bool :: Parser Exp
bool = true <|> false where
  true = ident "tasty" >> return (Constant 1)
```

```
    false = ident "disguisting" >> return (Constant 0)
```

Maybe.fromJust

Op een aantal plaatsen, bijvoorbeeld bij het ophalen van variabelen uit de environment, wordt gebruik gemaakt van een Maybe constructie. Als deze Nothing zou geven, zou het programma verder niet kunnen werken, omdat bijvoorbeeld een benodigde variabele niet bestaat. Hierdoor heb ik besloten geen Maybe's te returnen, maar met de fromJust functie te werken. Deze zal het programma doen stoppen als er een Nothing teruggegeven wordt, waardoor ik zelf geen error afhandeling moet doen. Dit maakt de code ook een stuk leesbaarder. VOORBEELD – TYPES.HS: 116-118.

Tests

Voor elk stuk functionaliteit werden tests geschreven, deze zijn te vinden als .course bestanden in de map courses/tests/.

Hoofdstuk 6

Conclusie

De eerlijkheid gebied mij te vermelden dat ik nog niet alle concepten van Haskell volledig onder de knie heb. Ik ben er dan ook van overtuigd dat bepaalde zaken veel korter en efficiënter geïmplementeerd zouden kunnen worden, maar wegens projecten van andere vakken en de examens had ik hier helaas geen voldoende tijd meer voor.

Graag had ik nog ondersteuning voor strings toegevoegd. In het begin had ik hier rekening mee gehouden, maar uiteindelijk kwam ik hierdoor in de problemen en om het mezelf niet onnodig moeilijk te maken heb ik besloten dit te verwijderen.

Alle methoden zijn gedocumenteerd volgens de Haddock-stijl, documentatie gegenereerd door Haddock werd ook bijgevoegd in de docs/ map.

Gebruikte bronnen

- De slides over Monads: voor het implementeren van de Parser monad.
- De Haskell-library Text.Parsec: inspiratie voor de hulpfuncties in de Lexer.
- `https://wiki.haskell.org/Parsing_a_simple_imperative_language`: om een basisidee te krijgen over de structuur van Parsers en Evaluators.
- `http://stackoverflow.com/questions/16970431/implementing-a-language-interpreter-` verdere uitwerking van evalueren.
- `https://www.schoolofhaskell.com/school/starting-with-haskell/basics-of-haskell/12-State-Monad`: zeer goede uitleg over het gebruik van de State monad.
- `http://pointfree.io/`: het inkorten van code achteraf.

Hoofdstuk 7

Appendix Broncode