onderzoeksverslag Functioneel Paradigma

Jelmer van Vugt (618770) – 22-06-2022

# Abstract

dd

Contents

[Abstract 1](#_Toc54356060)

[Lijst met afkortingen 3](#_Toc54356061)

[Begrippenlijst 4](#_Toc54356062)

[1. Inleiding 5](#_Toc54356063)

[2. Methodologie 6](#_Toc54356064)

[2.2 Literatuurstudie 6](#_Toc54356065)

[2.2 Prototyping 6](#_Toc54356066)

[3. Onderzoeksresultaten 7](#_Toc54356067)

[3.1 Literatuuronderzoek 7](#_Toc54356068)

[3.1.1 Wat is imperatief en declaratief programmeren? 7](#_Toc54356069)

[3.1.2 Welke termen komen veel voor binnen het functionele paradigma? 7](#_Toc54356070)

[3.1.3 Wat zijn states en hoe hebben deze invloed op het functionele paradigma? 9](#_Toc54356071)

[3.2 Labonderzoek 9](#_Toc54356072)

[3. Conclusie 10](#_Toc54356073)

[4. Discussie 11](#_Toc54356074)

[5. Literatuurlijst 12](#_Toc54356075)

[6. Bijlagen 13](#_Toc54356076)

# Lijst met afkortingen

ff

# Begrippenlijst

ff

# 1. Inleiding

Dit document is het onderzoeksverslag dat deel uit maakt van het beroepsproduct PO Paradigma uit het vak Algoritmes, ProgrammeerTalen en Paradigma’s in het derde jaar Software Development zoals aangeboden op de Hogeschool van Arnhem en Nijmegen.

In dit onderzoeksverslag zijn de volgende hoofd- en deelvragen beantwoordt.

## Onderzoeksvragen

Wat is functioneel programmeren?

* Wat is imperatief en declaratief programmeren?
* Welke termen komen veel voor binnen het functionele paradigma?
* Wat is Haskell?
* Hoe implementeer ik een Pythagoras fractal tree in Haskell?
* Welke aspecten uit het functionele paradigma komen terug in de uitwerking van de fractal tree?

## 1.2 Challenge

Als deel van het PO paradigma diende de student een zogeheten ‘challenge’ uit de kiezen en deze uit te werken in een functionele programmeertaal naar keuze. Ik heb ervoor gekozen om in de taal Haskell de Pythagoras fractal tree uit te werken. De uitwerking hiervan is tevens het prototype dat in deelvraag vier en vijf wordt behandeld.

Ik heb voor Haskell gekozen omdat ik hier voor die beroepsproduct al meerdere keren van heb gehoord maar nooit eerder ervaring mee heb opgedaan. Het was één van de eerste open-source ‘pure’ functioneel programmeertalen en lijkt mij daarom een mooie taal om mijn eerste ervaring met het functionele paradigma op te doen. De fractal tree lijkt mij hierbij een uitdagende maar haalbare opdracht om uit te werken.

# 2. Methodologie

In dit hoofdstuk worden de twee verschillende onderzoeksmethoden toegelicht die gebruikt zijn om de deelvragen te beantwoorden.

## 2.1 Onderzoeksmethoden

In dit kopje zijn de verschillende onderzoeksmethoden die binnen dit onderzoeksverslag zijn toegepast beknopt toegelicht.

|  |  |  |
| --- | --- | --- |
| Onderzoeksmethode | Toelichting | Toegepast binnen deelvraag |
|  |  |  |
| Literatuurstudie | Binnen een literatuurstudie wordt informatie vergaard uit leesbare artikelen. Deze informatie kan afkomstig zijn van websites, boeken, tijdschriften, kranten, et cetera. Uit deze gevonden artikelen kunnen selecte stukken worden gebruikt om op deze manier bij te dragen bij het beantwoorden van de deelvraag. | 1, 2, 3 |
| Prototyping | Binnen de prototyping onderzoeksmethode wordt een prototype ontwikkeld om een concept, design of probleem beter te begrijpen en/of de technische limieten en mogelijkheden de ontdekken. | 4 |
| Showroom | Binnen de showroom onderzoeksmethode wordt een gemaakt product – in dit geval een algoritme – aan de tand gevoelt en zijn bruikbaarheid, correctheid en functioneliteit ontleedt. | 5 |

# 3. Onderzoeksresultaten

In dit hoofdstuk zullen de resultaten van elke deelvraag worden behandeld.

## 3.1 Literatuuronderzoek

Binnen deze paragraaf zijn de deelvragen behandeld die door middel van een literatuuronderzoek zijn beantwoord.

### 3.1.1 Wat is imperatief en declaratief programmeren?

Programmeertalen binnen het functionele – en het object georienteerde paradigma hebben in feite beide hetzelfde doel, namelijk: het bouwen van software. Ondanks dat beide paradigma’s hetzelfde einddoel hebben zit er veel verschil in hoe ze dit doel bereiken.

Het grootste verschil tussen de twee paradigma’s is dat object georienteerde talen imperatief zijn en functionele talen declaratief. Imperatieve programmeren beschrijven *hoe* een programma werkt en declaratieve *wat* een programma moet doen. *(Golliwitzer, 2020)* Om hier een beter beeld bij te illustreren pakken we voor het gemak het bouwen van een huis. Imperatieve programma’s zouden de volgende structuur hebben: bouw de fundering, voeg het skelet toe, installeer de utiliteiten, bouw de muren en werk het tot slot mooi af. Een aanroep om een huis te bouwen in een declaratief programma zou er als het volgende uitzien: Ik wil een huis met een open haard en drie verdiepingen, het maakt me niet uit hoe je dit doet. Dit wordt ook wel abstracte code genoemd.   
Talen die gebruik maken van het imperatieve paradigma worden ook wel procedurele talen genoemd. Voorbeelden hiervan zijn Java en C++. Talen die gebruik maken van het declaratieve paradigma worden ook wel functionele talen genoemd. Voorbeelden hiervan zijn Haskell en Lisp.

### 3.1.2 Welke termen komen veel voor binnen het functionele paradigma?

Onder dit kopje worden een aantal veel voorkomende termen binnen het functionele paradigma onder elkaar gezet en toegelicht.

#### 3.1.2.1 Purity

In sommige functionele programmeertalen is het mogelijk voor functies om naast de returnwaarde te berekenen en terug te geven ook overige acties te ondernemen. Deze overige acties worden ook wel *side-effects* genoemd. Deze zijn side-effects genoemd omdat deze onverwachte acties mogelijk ongewenst het gedrag van andere functies kunnen beïnvloeden.

Om dit door side-effect verzoorzaakte mogelijke ongewenste gedrag te voorkomen is het concept ‘pure functions’ bedacht. Functies die worden beschouwd als ‘pure’ moeten voldoen aan de onderstaande eisen: *(Brassuer, 2014)*

* De functie moet altijd dezelfde waarde teruggeven gegeven dezelfde inputwaarden.
* De waardie die de functie mag alleen berekend worden door middal van variablen binnen zijn eigen scope. De lokale variabelen en eventuele parameters.
* De functie mag geen variabelen en/of objecten buiten zijn eigen scope aanpassen.

Omdat pure functies altijd dezelfde output moeten geven met dezelfde input zijn deze erg gemakkelijk te unit-testen. Ook hebben pure functies omdat deze geen interactie hebben met de buitenwereld amper impact op concurrency, wat erg voordelig is.

##### Lazy evaluation & Memoization

Lazy evaluation en memoization is een concept dat niet enkel bij talen binnen het functionale paradigma voorkomt. Omdat functionele talen echter grotendeels werken met pure functies simplificeert het dit wel.

Binnen het object geörienteerde talen wordt vaak gebruik gemaakt van het concept dat ‘eager evaluation’ heet. Een compiler die werkt volgens het eager evaluation principe evalueert een variabele direct nadat het een tegenkomt; ook in de gevallen waar dit nog niet nodig is of compleet onnodig is. Tevens kan het voorkomen dat dezelfde intensieve evaluaties meerdere keren herhaalt worden, wat resources op een systeem kost en ten koste gaat van de performance. *(Bencevic, 2018)*

Functionele talen die werken met een combinatie van lazy evaluation en memoization trachten zoveel mogelijk onnodige evaluaties te voorkomen. Omdat pure functies gegeven dezelfde argumenten altijd dezelfde uitkomst hebben kunnen de uitkomsten van bepaalde – er vanuit gaande dat deze dezelfde parameters ontvangen – worden hergebruikt in plaats van opnieuw worden berekent. Wat de performance enorm bevordert.   
Een voordeel dat lazy evaluation ook met zich meebrengt is de mogelijkheid om oneindig grote lijsten te kunnen gebruiken. Dit is mogelijk omdat een waarde in deze lijst pas wordt berekent wanneer deze nodig is.

##### Immutable data

Binnen veel functionele programmeertalen wordt er gebruik gemaakt van ‘immutable data’. Door veelal gebruik te maken van datastructuren die niet te wijzigen zijn is het schrijven van pure functies een stuk makkelijker. Ook brengt dit het voordeel met zich mee dat concurrency problemen veel minder voorkomend zijn omdat verschillende threads niet gelijktijdig dezelfde data kunnen gebruiken.

##### Monads

Ondanks dat pure functies de standaard zijn binnen het functionele paradigma kan het soms voorkomen dat een functie een actie moet verrichten die normaalgesproken voor een side-effect zou zorgen. Omdat side-effects zoveel mogelijk vermeden dienen te worden is hier een concept op bedacht, namelijk: monads.

Een monad fungeert als het ware als een container die door abstractie net doet alsof deze (mogelijke) side-effects niet bestaan. Denk hierbij aan bijvoorbeeld een I/O operatie. Doordat deze container een functie die (in sommige gevallen) een side-effect produceert omvat kan deze functie puur blijven. Dit stelt de ontwikkelaar in staat zich te focussen op de waardes die daadwerkelijk relevant zijn. *(Jitani, 2021).* Ook brengt de monad een set operaties met zich mee die uitgevoerd kunnen worden op deze container. Hierbij kan gedacht worden aan: map, return, join en bind.

#### 3.1.2.2 Higher order functions

Higher-order functions (HOFs) zijn functies die een functie gebruiken als een argument of een functie gebruiken als return waarden. Een HOF kan dit ook beide doen. Een voorbeeld hiervan is een functie die zowel een lijst als een functie krijgt als argumenten. Vervolgens voert de functie op elk item binnen de lijst de meegegeven functie uit en geeft de nieuwe lijst terug.

Een van de voordelen die HOFs met zich meebrengen is het verminderen van repetitie binnen de code. Neem als voorbeeld een for lus. Deze bestaan altijd, exclusief de implementatie binnen de lus, uit dezelfde structuur. Het herhalen van deze lussen kan opgevangen worden met een HOF. Dit concept staat ook wel bekend als polymorfie voor functies.

#### 3.1.2.2 Currying

Naarmate functies een groter aantal parameters ontvangen worden deze al snel onleesbaar. Om dit probleem op te lossen is het zogeheten concept ‘currying’ verzonnen.

Currying is het proces waarin één functie die meerdere argumenten ontvangt wordt opgebroken in meerdere individuele functies die elk maar één parameter ontvangen en elkaar in serie aanroepen. Dit proces wordt net zolang herhaalt tot alle paraments zijn gebruikt. Elke functie berekend een waarde gebaseerd op zijn argument en geeft deze terug aan de functie boven hem in de hiërarchie tot het uiteindelijke resultaat wordt teruggegeven door de eerste functie in de serie. *(Haskell Wiki, 2020)*

Door één grote functie op te splitsen in verschillende sub-functies wordt de code makkelijker om te refactoren. Ook maakt dit de code makkelijker te lezen én te begrijpen voor de ontwikkelaar(s). Currying gaat erg goed samen met ander concepten binnen het functionele paradigma die leunen op dit concept. Een voorbeeld hiervan is ‘partial application’. In een notendop is partial application een concept waarin een ge-curryde methode aangeroepen kan worden zonder dat deze hoeft te voldoen aan alle parameters. Vervolgens kan het programma later de overige argumenten meegeven waarna de functie wordt uitgevoerd. *(Gupta, 2018)*

### 3.1.3 Wat is Haskell?

Jj

## 3.2 Labonderzoek

Binnen deze paragraaf zullen de resultaten van alle deelvragen worden behandeld die met een labonderzoek zijn verkregen.

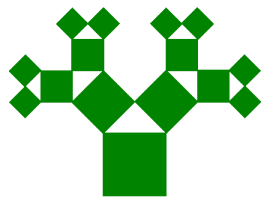
### 3.2.1 Hoe implementeer ik een Pythagoras fractal tree is Haskell?

Het implementeren van de Pythagoras’ fractal tree in Haskell bestaat uit drie fasen. In de eerste fase is de logica achter de fractal tree ontleedt, dit zodat deze kennis kon worden toegepast tijdens het implementeren van het algoritme. In de tweede fase is een keuze gemaakt voor een toolstack die het mogelijk maakt de fractal tree te implementeren. In de derde fase is het algoritme geschreven, deze is in de volgende deelvraag behandeld.

#### Fase 1

De Pythagoras boom is opgebouwd uit verschillende vierkantjes (fractals) die allemaal van één root fract vandaan stammen. Bij elke nieuwe iteratie van de boom stammen er twee nieuwe fractals uit de onderliggende die moeten voldoen aan de volgende drie regels:

* Het formaat van de fractals wordt bepaald door de volgende formule:
* De linker fractal wordt 45 graden naar links geroteerd en verplaatst zich ....
* De rechter fractal wordt 45 graden naar rechts geroteerd en verplaatst zich ...

In feite kan deze fractal tree tot in de oneindigheid door blijven itereren. Gebruikelijk is echter om deze na tien iteraties te stoppen. In de onderstaande afbeelding is een Pythagoras’ fractal tree afgebeeld na vier iteraties.

### Fase 2

Voor het implementeren van de Pythagoras fractal tree in Haskell is een Cabal project met de Gloss library gebruikt.

Het is in Haskell mogelijk om zonder externe libraries illustraties op het scherm te tekenen. Dit kost echter relatief heel veel moeite om voor elkaar te krijgen en leek mij niet waar de focus van de opdracht op moest liggen. Om deze reden is gekozen om gebruik te maken van de externe library Gloss om de illustratie uit te werken.

Om Gloss te kunnen implementeren in het Haskell project is de package manager Cabal gebruikt. Deze dient ook gebruikt te worden om het project te executeren.

### 3.2.2 Welke aspecten uit het functionele paradigma komen terug in de uitwerking van de fractal tree?

WIPWIP

# 4. Conclusie

ff

# 5. Discussie

ff

# 6. Literatuurlijst

Golliwitzer, Z. (2020) *Imperative vs. Declarative Programming (procedural, functional, and OOP).* Geraadpleegd op 8 juni 2022, via:<https://medium.com/@zach.gollwitzer/imperative-vs-declarative-programming-procedural-functional-and-oop-b03a53ba745c>.

Brassuer, A. (2014) *Functional Programming: Pure Functions.* Geraadpleegd op: 8 juni 2022, via: [https://www.sitepoint.com/functional-programming-pure-functions/#:~:text=A%20pure%20function%20is%20a,always%20return%20the%20same%20result](https://www.sitepoint.com/functional-programming-pure-functions/" \l ":~:text=A%20pure%20function%20is%20a,always%20return%20the%20same%20result).

Bencevic, M. (2018). *What is Lazy Evaluation? — Programming Word of the*. Geraadpleegd op: 8 juni 2022, via: <https://medium.com/background-thread/what-is-lazyevaluation-programming-word-of-the-day-8a6f4410053f>.

Jitani, V. (2022). *Monads explained.* Geraadpleegd op: 8 juni 2022, via: <https://towardsdatascience.com/monads-from-the-lens-of-imperative-programmer-af1ab8c8790c>.

Haskell Wiki. (2020) *Currying.* Geraadpleegd op 23 oktober 2020, via: <https://wiki.haskell.org/Currying>.

Gupta, D. (2018) *Javascript – Currying VS Partial Application.* Geraadpleegd op 23 oktober 2020, via: <https://towardsdatascience.com/javascript-currying-vs-partial-application-4db5b2442be8>.

Ram, M. (2019) *Functional Programming in simple terms.* Geraadpleegd op 18 oktober 2020, via: <https://medium.com/angular-in-depth/functional-programming-in-simple-terms-abcef30a2ad1>.

Khan, A. (2015) *What is functional programming?* Geraadpleegd op 23 oktober 2020, via: <https://wiki.haskell.org/Functional_programming#What_is_functional_programming.3F>.

# 7. Bijlagen

fff