|  |
| --- |
| **Paradigma Onderzoek**  Osama Halabi  Sn: 628160  APP22 2022-2023 |
| **HBO ICT Software Development 2022/2023.**  **Docent**: Michel Koolwaaij  Arnhem 04-02-2023  v. 2.8 |

Inhoudsopgave

[**Samenvatting** 3](#_Toc127795704)

[**1** **INLEIDING** 3](#_Toc127795705)

[**2** **METHODENKAART** 4](#_Toc127795706)

[**3** **RESULTATEN** 5](#_Toc127795707)

[3.1 Wat is Haskell 5](#_Toc127795708)

[3.2 Wat zijn Haskell concepten? 6](#_Toc127795709)

[3.2.1 Higher-Order Functions (HOF) 6](#_Toc127795710)

[3.2.2 Algebraic Data Types (ADT) 6](#_Toc127795711)

[3.2.3 Pattern Matching 6](#_Toc127795712)

[3.2.4 Purity 6](#_Toc127795713)

[3.2.5 Immutability 6](#_Toc127795714)

[3.2.6 Totality 6](#_Toc127795715)

[3.2.7 Laziness 6](#_Toc127795716)

[3.2.8 Type inference 7](#_Toc127795717)

[3.2.9 Polymorphisms 7](#_Toc127795718)

[3.2.10 layout sensitive 7](#_Toc127795719)

[3.2.11 Garbage collector 7](#_Toc127795720)

[3.2.12 Green threads 8](#_Toc127795721)

[3.2.13 Recursion 8](#_Toc127795722)

[3.3 Verschil tussen FP en OOP 8](#_Toc127795723)

[**4** **LITERATUURLIJST** 9](#_Toc127795724)

# **Samenvatting**

# **INLEIDING**

In dit onderzoek wil ik me richten op het leren van de programmeertaal Haskell en het implementeren van een connect spel met behulp van Haskell. Vervolgens relateer ik het connect spel aan Haskell concepten. Het doel is om te bestuderen hoe Haskell-concepten kunnen worden toegepast bij de ontwikkeling van het connect spel.

**Hoofdvraag:**

Hoe leer ik zoveel mogelijk over Haskell door het Connect spel te maken?

**Deelvragen:**

1. Wat is Haskell?
2. Wat zijn Haskell concepten?
3. Hoe kunnen deze Haskell concepten worden geïmplementeerd in de connect spel?
4. Wat is het verschil tussen FP-taal en OOP?

# **METHODENKAART**

In dit hoofdstuk beschrijf ik welke onderzoekmethoden ga ik gebruiken om de hoofvraag en deelvragen te kunnen beantwoorden.

Om deze vragen te beantwoorden, gaan ik gebruik maken van literatuuronderzoek en codeanalyse. Literatuuronderzoek zal me helpen om de relevante Haskell concepten te identificeren en te begrijpen hoe deze kunnen worden toegepast in de ontwikkeling van het connect spel.

Codeanalyse zal me helpen om de geïdentificeerde Haskell concepten te implementeren in dit spel. Hiermee wil ik nagaan hoe deze concepten in de praktijk toegepast kunnen worden en of ze bijdragen aan de ontwikkeling van het spel.

**Literatuur waar ik mee ga beginnen:**

1. Introductory Haskell course of the University of Pennsylvania (CIS194)
2. haskell-beginners-2022

Beide cursussen bieden me de gelegenheid om de basisprincipes van Haskell te leren en vertrouwd te raken met de syntax en de functies van deze taal. Deze cursussen bieden ook de mogelijkheid om deze kennis toe te passen door een aantal oefeningen en projecten te maken.

Bronnen:

* *[Introductory Haskell course of the University of Pennsylvania](https://www.seas.upenn.edu/~cis1940/spring13/lectures.html)*
* [*haskell-beginners-2022*](https://github.com/haskell-beginners-2022/course-plan)

# **RESULTATEN**

In dit hoofdstuk beantwoord ik alle deelvragen, dat zou me kunnen helpen om mijn hoofdvraag te kunnen beantwoorden.

## Wat is Haskell

Aan het einde van de jaren '80 brak er een nieuw tijdperk aan in de wereld van programmeertalen en paradigma's. Een groep onderzoekers werkte hard aan de ontwikkeling van Haskell, een functionele programmeertaal. Na veel hard werk werd Haskell uiteindelijk in 1990 gepresenteerd en vernoemd naar de beroemde Amerikaanse wiskundige Haskell Brooks Curry.

Haskell is een puur functionele programmeertaal die logica en extreem gecompliceerde berekeningen verwerkt door functies te definiëren en te combineren. De voordelen zijn onder meer dat het een ideale keuze is voor het maken van complexe programma's en logische processen.

Haskell is een moderne en veelgebruikte functionele programmeertaal die volledig losstaat van andere talen. Het is ontworpen om toepassingen te ondersteunen die variëren van cijfers tot symbolen. De expressieve syntaxis en het uitgebreide type systeem zorgen ervoor dat ontwikkelaars meer flexibiliteit en controle hebben bij het bouwen van software, waardoor het een ideale keuze is voor hen die op zoek zijn naar meer mogelijkheden.

Haskell is ook zeer geschikt voor parallel en gelijktijdig programmeren, wat betekent dat het gemakkelijk is om programma's te schrijven die tegelijkertijd op meerdere processors of systemen kunnen draaien. Dit maakt het een populaire keuze voor bijvoorbeeld grote datacenters en Cloud omgevingen.

Bronnen:

* *[What is Haskell Programming Language?](https://www.geeksforgeeks.org/what-is-haskell-programming-language/)*
* [*Haskell Language*](https://www.haskell.org/)
* [*Why Haskell?*](https://medium.com/geekculture/why-haskell-a9117c42da12)

## Wat zijn Haskell concepten?

Haskell heeft veel concepten die belangrij zijn om met deze programmeertaal aan de slag te kunnen gaan. Hieronder staan de belangrijkste concepten van Haskell:

### Higher-Order Functions (HOF)

Higher order functions zijn functies die als argumenten andere functies kunnen aannemen en/of functies kunnen retourneren als resultaat. Haskell is een functionele programmeertaal die veel gebruik maakt van higher order functions.

### Algebraic Data Types (ADT)

ADT's zijn datatypen die worden gedefinieerd door algebraïsche operaties zoals sommen (sum types) en producten (product types) Deze typen worden gedefinieerd door gegevensconstructeurs die gegevens vertegenwoordigen en/of nieuwe waarden creëren.

### Pattern Matching

Dit is een manier om waarden te matchen met een patroon (pattern), en vervolgens een bepaalde actie uit te voeren op basis van de overeenkomst.

In Haskell wordt pattern matching vaak gebruikt met functiedefinities en ADT's.

### Purity

Dit betekent dat functies geen neveneffecten hebben, alleen afhankelijk zijn van invoerparameters en mogelijk andere constanten en globale variabelen, en niet afhankelijk zijn van externe factoren zoals programmastatus of het bestandssysteem.

### Immutability

Dit betekent dat eenmaal aangemaakte gegevens niet meer kunnen worden gewijzigd. In plaats daarvan worden nieuwe gegevens gemaakt op basis van bestaande gegevens.

In Haskell worden alle waarden standaard niet beïnvloed. Met andere woorden, het kan niet worden gewijzigd nadat het is gemaakt. In plaats daarvan worden nieuwe waarden gecreëerd op basis van bestaande waarden.

### Totality

Dit betekent dat de functie werkt op alle inputs, dus ze ze gooien geen hand en geen uitzonderingen، ze heben geen gevallen die niet worden gedekt.

Totality functies maken het gemakkelijker om correcte en veilige code te schrijven omdat er minder ruimte is voor fouten en crashes. Deze functies zijn eenvoudig te testen omdat de invoer- en uitvoertypen volledig zijn gespecificeerd en de functies correct zijn gedefinieerd voor alle mogelijke gevallen. Bovendien zijn de totality functies vaak eenvoudiger te optimaliseren en te compileren omdat er minder speciale gevallen zijn waarmee rekening moet worden gehouden.

### Laziness

Dit betekent dat de evaluatie van de uitdrukking wordt uitgesteld tot het moment dat het resultaat nodig is. Het houdt in dat waarden pas worden geëvalueerd als ze nodig zijn, en de uitvoering van het programma kan worden geoptimaliseerd door alleen die waarden te evalueren die echt nodig zijn.

### Type inference

Dit betekent dat de compiler automatisch de types van expressies en functies kan afleiden zonder dat de programmeur ze apart hoeft te definiëren.

In Haskell gebruikt de compiler een type-inferentiesysteem om de types van expressies te bepalen op basis van de context waarin ze worden gebruikt.

### Polymorphisms

Dit betekent dat dezelfde functie kan worden gebruikt met verschillende soorten argumenten. Er zijn twee soorten polymorfisme in Haskell: parametrisch polymorfisme en ad-hoc polymorfisme.

Parameterpolymorfisme: Dit type polymorfisme betekent dat een functie voor elk type kan worden geschreven zonder speciale aannames te doen over de structuur of eigenschappen van dat type. Dit wordt bereikt door typevariabelen te gebruiken in plaats van concrete typen.

Ad-hoc polymorfisme: Dit type polymorfisme betekent dat dezelfde functie verschillende implementaties heeft voor verschillende typen. Dit wordt bereikt door typeklassen die een interface definiëren waaraan een type moet voldoen om een ​​functie te kunnen gebruiken.

### layout sensitive

Haskell is een lay-outgevoelige taal, wat betekent dat de lay-out van de code belangrijk is voor de interpretatie van het programma. In Haskell wordt codeplaatsing bepaald door witruimte, inclusief spaties en tabs, per positie.

In tegenstelling tot veel andere programmeertalen gebruikt Haskell geen accolades of sleutelwoorden om het begin en einde van een codeblok aan te geven. In plaats daarvan wordt de code niet door een vast patroon bepaald, maar door de mate waarin de verschillende delen van het programma inspringen.

De basisregel van de Haskell-lay-out is dat alle onderdelen van een structuur even ver uitgelijnd moeten staan. Dit impliceert dat alle regels die tot hetzelfde codeblok behoren in dezelfde kolom moeten worden gestart.

### Garbage collector

Haskell gebruikt garbage collection voor het eenvoudiger maken van het geheugenbeheer en het verzekeren van de veiligheid en efficiëntie van programma's. Haskell heeft een eigen runtime systeem en dit systeem heeft een garbage collector. Deze collector controleert automatisch de geheugenruimte van het programma tijdens het garbage collection proces en verwijdert objecten die niet meer in gebruik zijn uit het geheugen.

Haskell maakt objecten door functieaanroepen en plaatst deze objecten in de "heap". Wanneer er geen referenties meer zijn die naar een object verwijzen, is het niet langer bruikbaar. De garbage collector zoekt naar objecten die niet meer gebruikt worden en maakt zo weer ruimte in het geheugen vrij.

De garbage collector van Haskell werkt volgens het "mark-and-sweep" algoritme. Dit algoritme werkt door eerst alle objecten in het geheugen te markeren die nog door het programma in gebruik zijn. Dan worden alle objecten die niet zijn gemarkeerd verwijderd en wordt het geheugen dat ze innemen vrijgemaakt.

### Green threads

In Haskell kun je "threads" maken, en husker heeft green threads - ook bekend als "lightweight threads" of "Haskell threads".

Deze threads worden uitgevoerd door het Haskell runtime-systeem en worden beheerd door de Haskell runtime-scheduler, in plaats van door het besturingssysteem. Dit houdt in dat er geen directe overgang nodig is tussen de gebruikersmodus en de kernelmodus, wat de systeemprestaties kan verbeteren.

Haskell threads zijn aangemaakt met de functie forkIO, die een nieuw Haskell thread creëert dat een bepaalde actie uitvoert.

### Recursion

Het recursieve concept is van groot belang in Haskell en wordt gebruikt om problemen op te lossen door ze op te delen in kleinere en eenvoudigere subproblemen. Recursieve functies in Haskell worden dgedefinieerd door zichzelf aan te roepen totdat een basisgeval wordt bereikt.

Recursie is een belangrijk concept in Haskell, vooral als het gaat om het verdelen van problemen in kleinere en eenvoudigere deeltjes. Recursie is een belangrijk concept in Haskell dat ik moet begrijpen als ik complexe problemen wil oplossen.

Bronnen:

* *[Learn You a Haskell for Great Good!](http://learnyouahaskell.com/chapters)*
* [*Haskell Beginners 2022*](https://github.com/haskell-beginners-2022/course-plan)
* *[Haskell Tutorial: get started with functional programming](https://www.educative.io/blog/haskell-tutorial" \l "advanced)*

## Verschil tussen FP en OOP

Het belangrijkste verschil tussen object georiënteerd programmeren (OOP) en functioneel programmeren (FP) is hoe met data en functies omgaan.

OOP is een programmeerparadigma waarbij data en functies zijn georganiseerd in objecten. Deze objecten beschrijven hun gedrag door middel van methoden. Het doel van OOP is het maken van een computermodel van de werkelijkheid, waarbij objecten gebruikt worden om de abstracties van de echte wereld te weerspiegelen. Een OOP-programma is een verzameling objecten die met elkaar communiceren.

FP stelt daarentegen dat functies de meest waardevolle elementen van een programma zijn. Het is niet nodig de werkelijkheid te modelleren, maar wel de berekening uit te leggen. FP ziet gegevens als een passieve entiteit die je gebruikt als input voor functies. De nadruk ligt op het transformeren van data door middel van functies en het combineren van bestaande functies om nieuwe functies te creëren.

Een ander verschil is dat OOP meestal veranderlijke status toestaat, waarbij je objecten kunt wijzigen terwijl het programma draait. FP produceert altijd dezelfde output voor dezelfde input en werkt met data die niet verandert en met functies zonder neveneffecten.

OOP is gebaseerd op het modelleren van de realiteit met behulp van objecten, terwijl FP berekeningen definieert als functies die geen neveneffecten hebben en de toestand niet veranderen. Beide paradigma's hebben hun sterke en zwakke punten.

Bronnen:

* *[Functional programming vs OOP: Which paradigm to use?](https://www.educative.io/blog/functional-programming-vs-oop" \l "oop)*
* [*Difference between Functional Programming and Object Oriented Programming*](https://www.geeksforgeeks.org/difference-between-functional-programming-and-object-oriented-programming/)

# **LITERATUURLIJST**

1. *Lecture notes and assignments*. (n.d.). Penn Engineers University. Retrieved February 2, 2023, from <https://www.seas.upenn.edu/%7Ecis1940/spring13/lectures.html>
2. Kovanikov, D. (2022). *GitHub - haskell-beginners-2022/course-plan: Haskell course info, plan, video lectures, slides*. GitHub. Retrieved February 2, 2023, from <https://github.com/haskell-beginners-2022/course-plan>
3. GeeksforGeeks. (2022, March 2). *What is Haskell Programming Language*. <https://www.geeksforgeeks.org/what-is-haskell-programming-language/>
4. Lipovača, M. (n.d.). *Chapters - Learn You a Haskell for Great Good!*

<http://learnyouahaskell.com/chapters>

1. Khandaker, S. (2022, January 7). *Why Haskell? - Geek Culture*. Medium. <https://medium.com/geekculture/why-haskell-a9117c42da12>
2. Thelin, R. (2021, February 25). *Haskell Tutorial: get started with functional programming*. Educative: Interactive Courses for Software Developers.

<https://www.educative.io/blog/haskell-tutorial>

1. Dorman, T. (n.d.). *Functional programming vs OOP: Which paradigm to use*. Educative: Interactive Courses for Software Developers.

<https://www.educative.io/blog/functional-programming-vs-oop>

1. GeeksforGeeks. (2022, May 12). *Difference between Functional Programming and Object Oriented Programming*.

<https://www.geeksforgeeks.org/difference-between-functional-programming-and-object-oriented-programming/>