Onderzoeksplan: Haskell JSON Parser

Praktische Opdracht Paradigma

Datum	• Vrijdag, 6 augustus 2021 *Goedgekeurd* • Vrijdag, 16 september 2022 *(opnieuw ingediend)* • Vrijdag 30 september 2022 - **INGELEVERD**
Naam	F.K.A. Soffers
Student nummer	567780
Klas	n.v.t.
Opleiding	*HBO ICT*
Profiel	*Software Development*
Semester	*Advanced Software Development*
Academie	*Academie IT & Mediadesign*
Onderwijsinstelling	*Hogeschool van Arnhem & Nijmegen*
Docent(en)	- Dhr. Niek van Diepen

HAN_UNIVERSITY OF APPLIED SCIENCES

Inhoudsopgave

Inhoudsopgave **2**

Inleiding **3**

- Opdracht 3
- Programmeertaal en challenge 4
- Vraagstelling 4
  - Hoofdvraag 4
  - Deelvraag 4
- Toets-eisen 4

Resultaten **5**

- Wat is functioneel programmeren? 5
  - Concepten 5
    - Immutable Data 6
    - Referential transparency 6
    - Modularity 6
    - Maintainability 6
    - Closure 6
    - High-order functions 6
      - Pure functions 6
- Wat zijn de bijzonderheden van Haskell? 7
  - Haskell specifiek 7
- Wat is een JSON Parser en wat is het nut? 8
- Welke functionele aspecten kan ik gebruiken? 8
- Hoe maak je een JSON Parser met Haskell? 8

Inleiding

Één van de beroepsproducten die tijdens het ASD semester voor het vak APP gemaakt moet worden is de PO Paradigma, bij deze individuele toets moet er een rapport geschreven worden over een klein onderzoek naar een functionele programmeertaal die voorheen nog onbekend was. Het gaat er bij dit PO om dat er duidelijk wordt wat wat het functionele paradigma inhoudt en dat je jezelf een nieuwe programmeertaal vaardig kunt maken.

Volgens ASD APP-8:
“De student onderzoekt zelfstandig een overwegend functionele programmeertaal en programmeert daarin een algoritme uit.”

Opdracht

“Kies een voor jou onbekende functionele programmeertaal. Leer jezelf die taal, zodanig dat je enige concepten van functioneel programmeren leert. Kies ook een Challenge, een programmeeropdracht die de functionele concepten van je taal doet uitkomen, en programmeer die Challenge uit. Schrijf een rapport over de functionele concepten van je gekozen taal, de uitwerking van de Challenge, en hoe daarin een aantal van die functionele concepten terugkomen.”

Programmeertaal en challenge

Functionele Programmeertaal	Haskell
Challenge	*“JSON Parser”*
Onderzoeksmethode (*http://ictresearchmethods.nl/Methods*)	- Literature study - Prototyping

Vraagstelling

Hoofdvraag

“Hoe realiseer ik in Haskell een JSON Parser?”

Deelvraag

- Wat is functioneel programmeren?
- Wat zijn de bijzonderheden van Haskell?
- Wat is een JSON Parser?
- Wat is het nut van een JSON Parser?
- Welke functionele aspecten kan ik gebruiken bij het maken van de JSON Parser?
- Hoe maak je een JSON Parser met Haskell?

Toets-eisen

- [PO_Paradigma 1] Onderzoek voldoet aan SD-criteria voor onderzoeken (methodenkaart).
- [PO_Paradigma 2] Onderzoekt de concepten van een functionele programmeertaal.
- [PO_Paradigma 3] Programmeert een algoritme of programma uit in een overwegend functionele programmeertaal.
- [PO_Paradigma 4] Relateert functionele concepten aan specifieke onderdelen van zijn/haar uitgeprogrammeerde algoritme.re
- Voldoet aan de AIM controlekaart.

Als al deze eisen met Voldaan of beter worden beoordeeld, is de eindbeoordeling Voldaan. Je voldoet daarmee mede aan de volgende SD-Eindkwalificaties: SD-4, SD-5, SD-7, SD-8.

Resultaten

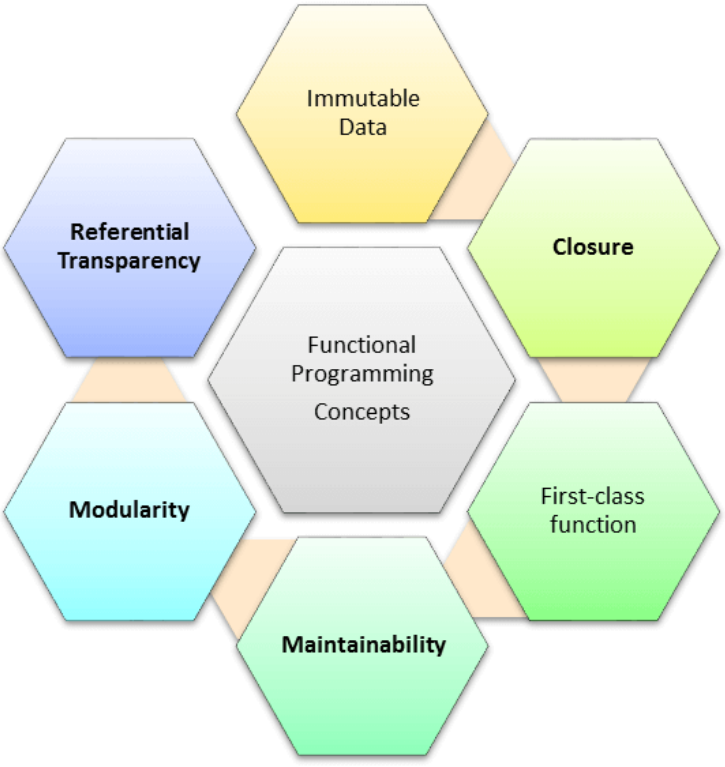
Wat is functioneel programmeren?

Een hele bondige samenvatting van functioneel programmeren is dat er bij functioneel programmeren alleen gefocust wordt op de resultaten en niet op het proces. Zo ligt de nadruk er voornamelijk op wat er berekend moet worden.

Een ander belangrijk aspect is dat de data waarmee gewerkt wordt niet te veranderen is. Zo is functioneel programmeren niet geschikt voor datasets die dynamisch zijn en tijdens het draaien van de programma's veranderen. Functioneel programmeren is dan ook gebouwd rond het concept van wiskundige functies die gebruik maken van conditionele expressies and recursie om zo berekeningen uit te voeren.

Een groot verschil met de programmeertalen die ik gewend ben, zoals Java en Python, is dat er geen ondersteuning is voor iteraties en dat functies als IF-ELSE statements en loops dus niet ondersteund worden.

Concepten



(https://www.guru99.com/functional-programming-tutorial.html)

Immutable Data

Immutable data houdt in dat het gemakkelijker moet zijn om data structures aan te maken in plaats van een structure die al bestaat aan te passen.

Referential transparency

Functionele programma's zouden operations moeten uitvoeren alsof het de eerste keer is dat ze deze uitvoeren. Zodat je weet wat er wel of niet heeft plaatsgevonden tijdens de uitvoer van het programma.

Modularity

Modulair design verhoogd de productiviteit. Kleine modules kunnen sneller opgezet worden en hebben een grotere kans om hergebruikt te worden wat ertoe leidt dat het sneller en gemakkelijker is om grotere programma's te ontwikkelen. Hiernaast draagt modulariteit er ook aan bij dat deze modules afzonderlijk getest kunnen worden op hun functioneren wat eraan bijdraagt dat er minder tijd aan debuggen en unit testen besteed hoeft te worden.

Maintainability

Functionele programma's zijn makkelijker te onderhouden omdat men zich geen zorgen hoeft te maken over veranderingen die per ongeluk plaatsvinden buiten de functie.

Closure

De closure is een inner functie die toegang heeft tot de variabelen van de parent functies, zelfs nadat de de parent functie is uitgevoerd.

High-order functions

High-order functies nemen ofwel een andere functie als argument of geven een andere functie terug als argument.

Pure functions

Een pure functie is een functie van welke de inputs zijn gedeclareerd als inputs en geen van deze zouden verborgen mogen zijn. De outputs zijn gedeclareerd als outputs.

Pure functies reageren op de parameters die ze meekrijgen. Het is niet efficiënt wanneer het niets return en bovendien bieden ze dezelfde output voor de gegeven parameters.

Een voorbeeld:

```
Function Pure(a,b)
{
        return a+b;
}
```

Het tegenovergestelde hiervan zijn impure functies, dit zijn functies die verborgen in- en outputs hebben. Impure functies kunnen niet geïsoleerd gebruikt en getest worden.

Wat zijn de bijzonderheden van Haskell?

Enkele voordelen van functioneel programmeren in het algemeen zijn als volgt:

- **Bugs-Free Code:** functioneel programmeren ondersteund het gebruik van *states* niet, dus er zijn geen bijverschijnselen in de resultaten en is het mogelijk om error vrije code te schrijven

- **Efficient Parallel Programming:** Omdat functionele programmeertalen geen *mutable* (veranderlijke) state hebben zijn er ook geen problemen met *state-changes*. Functies kunnen geprogrammeerd worden om parallel te werken als ‘instructies’. Zulke code ondersteund het gemakkelijke hergebruik en testability.

- **Efficiency:** Functionele programma's bestaan uit onafhankelijk units die gelijktijdig kunnen opereren. Dit heeft als resultaat dat zulke programma's erg efficiënt zijn.

- **Support voor Nested Functions**

- **Lazy Evaluation:** Als het programma iets niet nodig heeft wordt het ook niet gebruikt. Functioneel programmeren ondersteund onder andere Lazy Functional Constructs zoals Lazy Lists, Lazy Maps, etc.

Een groot nadeel is wel dat Functionele programma's erg ‘programmeer intensief’ zijn, en er continue refactoring nodig is en functies tijdens het uitwerken vaak veranderen.
Daarnaast doordat het geen gebruik maakt van states, moet er elke keer wanneer er een actie gedaan moet worden een nieuw object aangemaakt worden.
Dit zorgt er onder andere voor dat functionele programma's erg veel geheugen in beslag nemen ten opzichte van niet functionele programma's.

Haskell specifiek

Haskell verschilt met andere functionele programmeertalen dat het support heeft voor onder andere:

- Pure functions by default
- Monadic side effects
- Type classes
- **Syntax based on layout**

- **Haskell is een puur functionele taal**
- **geen functies met zijeffecten**
- **Lazy Evaluation (o.a. oneindige structuren)**

Wat is een JSON Parser en wat is het nut?

Parsing is een proces waarbij textuele input data genomen wordt welke geconverteerd wordt naar een *data structure*. Hierbij wordt er gecontroleerd of the input van een correcte syntax is. Parsing wordt onder andere gebruikt in compilers en interpreters voor programmeertalen om source text files te controleren en te converteren naar interne representaties welke later gebruikt worden in een proces.

Parsing wordt ook gebruikt voor het converteren van data van het ene format naar een ander format.
Bij een JSON Parser wordt dus een JSON input genomen welke geconverteerd wordt naar een ander format (of andersom natuurlijk).

Het nut van een JSON parser is dat wanneer ik bijvoorbeeld een JSON file wil versturen naar een website, dit niet als een JSON bestand verstuurd kan worden maar dat dit als een tekstbestand (een string) verpakt moet worden. Dit is waar de parser bij komt helpen, deze neemt de JSON data en maakt hier een verzendbare string van.

In dit onderzoek parse ik JSON naar een AST (Abstract Syntax Tree). Een AST wordt in *static code analysis* vaak gebruikt als intermediaire representatie waarop berekeningen kunnen worden gedaan zonder het daadwerkelijk uitvoeren van het programma. Het medium dat hiervoor vaak gekozen wordt is een AST.

Welke functionele aspecten kan ik gebruiken?

De functionele aspecten die ik kan gebruiken bij het maken van de JSON Parser zijn:

- Pure functies
- Recursie
- First-Class functies en High-Order

Hoe maak je een JSON Parser met Haskell?