Introductie tot de cursus

In deze introductie willen wij u informeren over de bedoeling van de cursus, de opzet van het cursusmateriaal en over de manier waarop u de cursus kunt bestuderen. U vindt in deze introductie dus nog geen echte studiestof, maar slechts praktische en studietechnische informatie die u inzicht geeft in de aard en opzet van de cursus en die u kan helpen bij het studeren.

1 Plaats en functie van de cursus

De cursus Functioneel programmeren is een cursus van het tweede niveau met een studielast van 100 uur. De cursus geeft een vervolg aan andere programmeercursussen in de informaticaopleiding van de Open Universiteit Nederland, zoals Objectgeoriënteerd programmeren met Java en de cursussen over het programmeren van een webapplicatie.
In een functionele programmeertaal staan functies centraal: functies kunnen als argument worden meegegeven aan een andere functie, worden opgeleverd als resultaat van een functie of worden opgeslagen in een datastructuur. Dit maakt het mogelijk om programma's op een hoog abstractieniveau op te schrijven en om te abstraheren van veelgebruikte programmeerpatronen, wat leidt tot bondige programma's.
In de cursus wordt gebruikgemaakt van Haskell, een moderne, sterk getypeerde, lazy programmeertaal. De programmeertaal is vernoemd naar de logicus Haskell B. Curry.
Het doel van de cursus is om op een andere manier naar het schrijven van programma's te kijken. Allerlei belangrijke concepten en programmeerconstructies worden in de cursus behandeld. Deze concepten zijn ook steeds vaker terug te vinden in mainstream programmeertalen.
Functionele programmeertalen behoren tot de klasse van *declaratieve talen*. In een declaratieve taal ligt de nadruk op het specificeren *wat* je wilt uitrekenen, en minder op *hoe* dit uitrekenen nu precies gebeurt. Een andere klasse van programmeertalen zijn de imperatieve talen, waaronder bijvoorbeeld Java en C# vallen. Een imperatief programma kan worden gezien als een reeks instructies die achtereenvolgens moeten worden uitgevoerd door een computer. Deze instructies hebben veelal een neveneffect, zoals het veranderen van de waarde in een geheugenlocatie door een toekenning, om het doel van het programma te bereiken. In een declaratieve taal zijn neveneffecten afwezig, of spelen slechts een beperkte rol. De meest bekende declaratieve talen zijn toegespitst op één enkel toepassingsgebied: HTML is een declaratieve taal voor het beschrijven van een webpagina, SQL voor het bevragen van een database en reguliere expressies voor het beschrijven van een taal.
Functionele programmeertalen hebben een rijke geschiedenis die teruggaat tot de ontwikkeling van de *lambda-calculus* door Alonzo Church in de jaren dertig. De lambda calculus (λ-calculus) is een wiskundige theorie over het rekenen met en redeneren over functies. Deze theorie staat aan de basis van de functionele programmeertalen, en verklaart ook de Griekse letter die te zien is op de voorkant van het werkboek en het tekstboek.
Lisp (John McCarthy, 1958) wordt algemeen gezien als de eerste functionele programmeertaal. Inmiddels zijn er vele functionele talen ontwikkeld die op een aantal punten nogal verschillen. Eén belangrijk verschil is of er wel of niet neveneffecten plaatsvinden tijdens het evalueren van een functie. Een ander verschil is of programma's statisch worden getypeerd (bij het compileren) of dynamisch worden getypeerd (bij het uitvoeren). Hieronder volgt een opsomming van de belangrijkste functionele programmeertalen.

- Lisp is de oudste functionele programmeertaal en wordt nog steeds veel gebruikt. Scheme en Clojure zijn bekende dialecten van de taal Lisp.
- ML is een strikte functionele programmeertaal waarin neveneffecten kunnen plaats-

vinden in functies. Functies staan centraal in ML, maar omdat neveneffecten zijn toegestaan kan er ook imperatief met de taal worden geprogrammeerd. De taal OCaml is een variant waarin functionele aspecten worden gecombineerd met concepten uit objectgeoriënteerde programmeertalen. F# (Don Syme, 2005) is een recent ontwikkelde programmeertaal gebaseerd op ML en OCaml, en is bedoeld voor het .NET platform van Microsoft.
- Miranda is de eerste functionele programmeertaal waarin functies en neveneffecten van elkaar zijn gescheiden. Opvolgers van deze taal zijn Haskell en Clean, een programmeertaal afkomstig van de Radboud Universiteit Nijmegen. Clean lijkt sterk op Haskell.
- Erlang is een algemene programmeertaal ontwikkeld door Ericsson met bijzonder goede ondersteuning voor concurrency en het schrijven van gedistribueerde systemen. De taal is niet statisch getypeerd.

Doelgroepen

De cursus Functioneel programmeren is een verplicht onderdeel van de bacheloropleiding informatica. De cursus is ook geschikt als losse cursus voor personen die behoefte hebben aan een op zichzelf staande cursus over de basisprincipes achter het functionele programmeerparadigma. We verwachten dat de cursus ook interessant is voor softwareontwikkelaars die niet in de praktijk werken met een functionele programmeertaal, vanwege de programmeerconcepten die in de cursus naar voren komen. De cursus biedt slechts een introductie tot het functioneel programmeren: tal van geavanceerde onderwerpen komen niet aan bod, waaronder bijvoorbeeld structuren zoals functoren en arrows en de vele uitbreidingen op het typesysteem.

2 Inhoud van de cursus

2.1 VOORKENNIS

Om de cursus met succes te kunnen volgen is enige ervaring met een andere programmeertaal gewenst, bijvoorbeeld op het niveau van de cursus Objectgeoriënteerd programmeren in Java 1. De cursus veronderstelt vrijwel geen voorkennis uit andere cursussen. Elementaire kennis van functies en recursief kunnen denken, zoals dat wordt aangeboden in de cursus Discrete wiskunde A, komt van pas. Het opstellen van een recursieve functie komt in het tekstboek uitgebreid aan bod (zie paragraaf 2.5). Aan dit onderwerp wordt een volledige leereenheid besteed in dit werkboek.
Omdat het tekstboek in het Engels is geschreven, is een redelijke leesvaardigheid van de Engelse taal een vereiste.

2.2 LEERDOELEN

Na het bestuderen van de cursus wordt verwacht dat u
- de ideeën achter functioneel programmeren begrijpt en de verschillen ten opzichte van een imperatieve programmeertaal kunt uitleggen
- een functie kunt opstellen, in het bijzonder door gebruik te maken van patroonherkenning, gevalsonderscheid en recursie
- in staat bent om hogere-orde functies te gebruiken en deze zelf te definiëren
- overweg kunt met het typesysteem, met name met polymorfe en overloaded types
- kunt uitleggen waarom monadische effecten nodig zijn in Haskell om input/output te realiseren
- de basis begrijpt van lazy evaluatie, zoals het werken met oneindige lijsten, en 'equational reasoning'
- de opgedane kennis kunt toepassen door zelf een correct en bondig programma te schrijven in de programmeertaal Haskell.

2.3 OPBOUW VAN DE CURSUS

De cursus is gebaseerd op het Engelstalig tekstboek *Programming in Haskell*, Graham Hutton, Cambridge University Press, 2007. Het tekstboek bestaat uit 13 hoofdstukken en moet in zijn geheel worden bestudeerd. Bij ieder hoofdstuk is een leereenheid

geschreven met een opsomming van de leerdoelen, extra uitleg van de stof, extra opgaven om mee te oefenen en een zelftoets. We geven een overzicht van de opbouw van het tekstboek.

Hoofdstukken 1–4
studielast: 19 uur

De eerste vier hoofdstukken staan in het teken van het schrijven van eenvoudige Haskell-functies. Er wordt achtergrondinformatie gegeven over de programmeertaal Haskell en de belangrijkste kenmerken van deze functionele programmeertaal, en er wordt kennisgemaakt met de WinHugs-interpretator en de Prelude met standaardfuncties. In hoofdstuk 3 komen types en typeklassen aan de orde, en wordt uitgelegd welke rol zij spelen bij het programmeren. In hoofdstuk 4 komen diverse concepten langs voor het opstellen van een functie, zoals guards, patroonherkenning, anonieme lambda-functies en secties.

Hoofdstukken 5–7
studielast: 17 uur

De hoofdstukken 5 tot en met 7 behandelen concepten waarmee meer complexe functies geschreven kunnen worden. Lijstcomprehensies helpen bij het schrijven van functies over lijsten – een datastructuur die in veel gevallen kan worden gebruikt bij het opstellen van een programma. Hoofdstuk 6 beschrijft hoe recursieve functies kunnen worden gedefinieerd en geeft een stappenplan voor het opstellen van zo'n recursieve functie. Vervolgens wordt gekeken naar hogere-orde-functies: dit zijn functies die een functie als argument nemen en daardoor uiterst flexibel zijn. In het bijzonder wordt een aantal hogere-orde-functies geïntroduceerd die veel voorkomende patronen van recursie over lijsten vangen: *map*, *filter* en *foldr*. Na deze hoofdstukken bent u in staat om te beginnen aan de eerste practicumopdracht. Meer informatie over deze opdracht wordt gegeven in paragraaf 5.

Practicumopdracht 1
studielast: 12 uur

Hoofdstukken 8–9
studielast: 12 uur

Hoofdstukken 8 en 9 beschrijven hoe ontleders en interactieve programma's kunnen worden geprogrammeerd in Haskell. Hoewel ontleders en interactieve programma's op het eerste gezicht niets gemeenschappelijk lijken te hebben, zal blijken dat ze beide gebruikmaken van dezelfde operator (namelijk de monadische sequentie-operator) om onderdelen samen te stellen tot een complexer geheel. Deze overeenkomst maakt dat in beide gevallen de do-notatie kan worden gebruikt: dit is een notatie die maakt dat programma's, op die plaatsen waar de volgorde van evaluatie er toe doet, meer imperatief ogen. Bij het bestuderen van interactieve programma's zal het ook duidelijk worden hoe input/output kan worden gerealiseerd in een taal waarin functies geen neveneffecten kunnen hebben.

Hoofdstukken
10–11
studielast: 11 uur

In hoofdstuk 10 wordt uitgelegd hoe nieuwe types en typeklassen worden gedefinieerd. Hoofdstuk 11 beschrijft een groter probleem waarvoor een oplossing in Haskell wordt ontwikkeld: eerst wordt een naïeve uitwerking beschreven die later wordt omgevormd tot een efficiëntere uitwerking. Na het bestuderen van hoofdstuk 11 kunt u beginnen aan de tweede practicumopdracht. Ook deze opdracht bespreken we verder in paragraaf 5.

Practicumopdracht 2
studielast: 12 uur

Hoofdstukken
12–13
studielast: 11 uur

In de laatste hoofdstukken komen twee meer geavanceerde onderwerpen aan de orde: de lazy evaluatiestrategie die het onder andere mogelijk maakt om te werken met oneindige datastructuren, en bewijzen van programma-eigenschappen volgens het principe van inductie.

3 Aanwijzingen voor het bestuderen van de cursus

3.1 OVERZICHT VAN HET CURSUSMATERIAAL

Het cursusmateriaal bestaat uit het tekstboek, dit werkboek en de cursussite. Op de cursussite staat alle informatie die aan verandering onderhevig is, zoals bijvoorbeeld de planning van het maken en inleveren van de opdrachten, en actuele informatie over de studiebegeleiding.

Tekstboek

Het tekstboek *Programming in Haskell* is geschreven door Graham Hutton, een gerenommeerd docent en onderzoeker op het gebied van functionele programmeertalen. Het tekstboek introduceert op een compacte wijze de belangrijkste principes van het functioneel programmeren. Op de website van het tekstboek, `www.cs.nott.ac.uk/~gmh/book.html`, is aanvullend materiaal te vinden dat kan helpen bij het bestuderen van de hoofdstukken. Voor ieder hoofdstuk zijn PowerPoint slides ont-

wikkeld door de auteur, die zelfstandig bekeken kunnen worden. Videocolleges, gegeven door Erik Meijer, waarin diezelfde slides worden gebruikt, zijn beschikbaar voor elk hoofdstuk via Microsoft's Channel 9.

Andere tekstboeken

Op de Haskell Wiki (`www.haskell.org`) staan andere boeken en tutorials vermeld om kennis te maken met de programmeertaal Haskell. Twee andere tekstboeken zijn het noemen waard omdat ze online te lezen zijn:

- *Learn You a Haskell* (`www.learnyouahaskell.com`) is een rijk geïllustreerd tekstboek waarin de basis van Haskell wordt uitgelegd.
- *Real World Haskell* (`book.realworldhaskell.org`) behandelt een aantal geavanceerde onderwerpen die nodig zijn om de programmeertaal in 'de echte wereld' te gebruiken.

Werkboek

Het werkboek is de leidraad voor het bestuderen van de cursus. In het werkboek staan aanwijzingen wanneer passages uit het tekstboek moeten worden gelezen. De nummering van de hoofdstukken in het tekstboek loopt gelijk op met de leereenheden in het werkboek. Iedere leereenheid bevat een beschrijving van de leerdoelen, een schatting van de studielast en opgaven met terugkoppelingen. Een leereenheid wordt afgesloten met een zelftoets waarmee kan worden nagegaan of de stof voldoende is begrepen.

Codefragmenten

In het werkboek zijn vele codefragmenten opgenomen. Deze fragmenten kunnen letterlijk worden overgenomen in een Haskell-module. Een voorbeeld van een codefragment is de onderstaande definitie van de faculteitsfunctie.

$$\begin{aligned} &\mathit{faculteit} :: \mathit{Int} \to \mathit{Int} \\ &\mathit{faculteit}\ 0 = 1 \\ &\mathit{faculteit}\ n = n * \mathit{faculteit}\ (n-1) \end{aligned}$$

Sessie met interpretator

Naast codefragmenten worden in het werkboek ook interactieve sessies met de WinHugs-interpretator getoond. In een sessie kunnen expressies (zoals *product* $[1\,..\,5]$) worden uitgerekend tot een waarde (zoals 120). Interactieve sessies zijn te herkennen aan de command prompt, hieronder te zien als `Hugs>`, en aan het gebruikte lettertype.

```
Hugs> product [1..5]
120
```

Cursussite

Aanvullend materiaal bij de cursus is beschikbaar via Studienet, de elektronische leeromgeving van de Open Universiteit. De cursussite van Functioneel programmeren is te bereiken via `studienet.ou.nl`. Hier vindt u onder andere:

- de installatiehandleiding bij de software
- verwijzingen naar aanvullend materiaal bij het tekstboek
- mogelijke aanvullingen op de stof en errata
- beide practicumopdrachten
- informatie over het tentamen
- oefententamens met uitwerking
- de discussiegroep voor het stellen van vragen
- contactgegevens van de studiebegeleiders.

Zelf oefenen

Om de programmeertaal te beheersen is het absoluut noodzakelijk om zelf te oefenen met het schrijven van code. Gebruik de Haskell-interpretator WinHugs tijdens het bestuderen van een leereenheid om opdrachten uit te werken, en om te experimenteren met alternatieve oplossingen. Om het oefenen makkelijker te maken, is voor elke leereenheid een bouwsteen beschikbaar waarin de definities zijn opgenomen die aan de orde komen. Deze bouwstenen zijn te downloaden vanaf de cursussite.

3.2 PROGRAMMATUUR

WinHugs

Bij het bestuderen van de cursus heeft u een Haskell- compiler of -interpretator nodig, en een tekstverwerker. We raden u aan om gebruik te maken van de Hugs-interpretator: dit is een lichtgewicht applicatie met een gebruiksvriendelijke user interface. De Hugs-interpretator is te downloaden via `www.haskell.org/hugs`. Er zijn distributies beschikbaar voor diverse soorten systemen. In het werkboek gaan we uit van de versie voor het Windows-besturingssysteem, met de naam WinHugs.

Een handleiding voor het installeren van WinHugs is te vinden op de cursussite.

GHC

Hoewel WinHugs goed genoeg is voor de cursus, wordt in de praktijk meestal de Glasgow Haskell Compiler (GHC) gebruikt voor serieuzere projecten. GHC is een 'state of the art' compiler, met vele extra mogelijkheden en bibliotheken. Bij de GHC-compiler hoort een interpretator (GHCi) waarmee interactieve sessies kunnen worden uitgevoerd. Wie GHC wil gebruiken voor de cursus wordt geadviseerd om de laatste versie van het Haskell Platform te installeren via `www.haskell.org/platform`. Het Haskell Platform is beschikbaar voor Windows, Mac en diverse Linux- en Unix-varianten.

Notepad++

Om Haskell-modules te kunnen schrijven heeft u een teksteditor nodig. In principe volstaat elke teksteditor, al is het raadzaam een editor te kiezen die 'syntax highlighting' ondersteunt voor de taal Haskell. Op de cursussite staan instructies hoe de gratis editor Notepad++ kan worden opgehaald van de website die te bereiken is via `www.notepad-plus-plus.org`.

4 Studiebegeleiding

Informatie over de groepsbegeleiding van de cursus kunt u vinden op de cursussite op Studienet. Buiten de periode waarin groepsbegeleiding plaatsvindt, is er alleen studiebegeleiding via Studienet.

Discussiegroep

Via de cursussite van Functioneel programmeren kunt u ook de discussiegroep van de cursus bereiken, waar u uw vragen over de cursus en uw eventuele problemen met de software kwijt kunt. Antwoorden kunnen van medecursisten komen of van een van de studiebegeleiders of auteurs van de cursus.

5 Wijze van toetsing

De cursus Functioneel programmeren wordt getoetst met twee practicumopdrachten en een schriftelijk tentamen.

Twee practicumopdrachten studielast: 24 uur

Met de twee practicumopdrachten moet u laten zien dat u zelfstandig een Haskell-programma kunt ontwikkelen, en dat u de onderwerpen die in de cursus behandeld worden voldoende beheerst. De practicumopdrachten hebben beide een studielast van 12 uur, en zijn te vinden op de cursussite. Aan de eerste opdracht kan worden begonnen nadat leereenheid 7 is bestudeerd. Na leereenheid 11 begint u aan de tweede opdracht. Uitwerkingen kunnen worden opgestuurd naar de examinator, en worden beoordeeld als voldoende of onvoldoende. Naast de beoordeling ontvangt u beknopte feedback op uw uitwerking. Alleen correcte Haskell-programma's (dat wil zeggen, programma's die worden geaccepteerd door een Haskell-compiler of interpretator) zullen in behandeling worden genomen. De opdrachten tellen niet mee in het eindcijfer: voor beide opdrachten moet wel een voldoende zijn gehaald om de cursus af te ronden.

Openboek tentamen

De cursus wordt afgesloten met een schriftelijk openboektentamen. Het is toegestaan om tijdens het tentamen gebruik te maken van het cursusmateriaal. Het is echter niet toegestaan om tijdens het tentamen gebruik te maken van elektronische hulpmiddelen. In het bijzonder mogen apparaten waarop een Haskell-compiler of -interpretator aanwezig is niet worden gebruikt. Meer informatie over het tentamen en de tentamendata kunt u vinden in de Studiegids Informatica en op de cursussite op Studienet.

Eindtoets

Aan het eind van het werkboek is een eindtoets opgenomen die representatief is voor het tentamen. De vragen van de eindtoets zijn qua aantal en moeilijkheidsgraad vergelijkbaar met de opgaven van het schriftelijk tentamen. Wij raden u sterk aan deze eindtoets pas te maken als u klaar bent met de tentamenvoorbereiding.

6 Cursusteam

Het werkboek is geschreven door Bastiaan Heeren, Harrie Passier en Manuela Witsiers. De auteurs zijn allen werkzaam bij de Open Universiteit Nederland.

Adviezen over de opzet en de inhoud van de cursus zijn gegeven door prof. dr. S. Doaitse Swierstra van de Universiteit Utrecht.
De uitwerkingen van de geselecteerde vragen in het tekstboek zijn gebaseerd op de Engelstalige antwoorden van de auteur, die zijn te vinden op de webpagina van het tekstboek *Programming in Haskell*. Het cursusteam wil Graham Hutton vriendelijk bedanken voor het beschikbaar stellen van de broncode van de antwoorden.