KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Een gecombineerde calculus voor algebraïsche, scoped en parallelle effecten

Lander Debreyne

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotor:
Prof. dr. ir. T. Schrijvers

Begeleider:
Ir. B. van den Berg

Academiejaar 2022 – 2023

Voorwoord

...

Lander Debreyne

Inhoudsopgave

Voorwoord i

Samenvatting iv

Lijst van figuren en tabellen v

1 Inleiding 1
- 1.1 Effecten . . . 1
- 1.2 Effect handlers . . . 2
- 1.3 Doel . . . 3
- 1.4 Overzicht . . . 3

2 Achtergrond 5
- 2.1 Effect handlers . . . 5
- 2.2 Algebraïsche effecten . . . 5
- 2.3 Calculus . . . 6
- 2.4 Scoped Effecten . . . 6
- 2.5 Parallelle Effecten . . . 6

3 Startpunt 7
- 3.1 λ_{sc}: A Calculus for Scoped Effects & Handlers . . . 7
- 3.2 λ_{p}: Parallel Algebraic Effect Handlers . . . 7

4 Motivatie en Uitdagingen 9
- 4.1 Beperkingen op parallelle uitvoering effecten . . . 9
- 4.2 Parallelle semantiek inbouwen in de calculus . . . 9

5 Syntaxis 11

6 Operationele Semantiek 13
- 6.1 Letrec: Recursieve functies . . . 13
- 6.2 Parallelle semantiek . . . 13

7 Type- en Effect-Systeem 15
- 7.1 Lijsten . . . 15

8 Uitgewerkte Voorbeelden 17
- 8.1 Voorbeelden met Scoped Effecten . . . 17
- 8.2 Voorbeelden met Scoped Effecten . . . 17
- 8.3 Voorbeelden met Beide Effecten . . . 17

9 Metatheorie **19**
9.1 Lemma's . . . 19
9.2 Behoud . . . 19
9.3 Vooruitgang . . . 19

10 Evaluatie **21**
10.1 Bijdrage . . . 21
10.2 Bruikbaarheid . . . 21
10.3 Correctheid . . . 21
10.4 Implementatie . . . 21
10.5 Backwards compatibility . . . 21

11 Gerelateerd Werk **23**

12 Besluit **25**
12.1 Resultaten . . . 25
12.2 Beperkingen . . . 25
12.3 Toekomstig werk . . . 25

Bibliografie **29**

Samenvatting

In dit `abstract` wordt een al dan niet uitgebreide samenvatting van het werk gegeven. De bedoeling is wel dat dit tot 1 bladzijde beperkt blijft.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

Programmeren is het proces van het schrijven van instructies die een computer kan uitvoeren om een specifieke taak te volbrengen. Dit gebeurt meestal in een programmeertaal, een speciale taal die bestaat uit een set woorden, symbolen en regels waarmee instructies aan een computer gegeven worden. Om de eigenschappen en het gedrag van een programmeertaal formeel te bestuderen, worden calculi gebruikt. Programmeertaal-calculi bieden een wiskundig kader waarin formele analyses kunnen worden uitgevoerd om de fundamentele eigenschappen van de taal te begrijpen. Met behulp van abstracte algebra en logische notatie worden de syntaxis, de operationele semantiek en het type-en-effectsysteem van een vereenvoudigde versie van de programmeertaal voorgesteld.

1.1 Effecten

Effecten zijn het resultaat van een actie of bewerking die door een computerprogramma wordt uitgevoerd. Het kan gaan om het weergeven van informatie op het scherm, het opslaan van gegevens in een database of het uitvoeren van een berekening. In programma's zijn effecten noodzakelijk om het gewenste gedrag te bereiken. Anderzijds kunnen effecten onbedoelde en ongewenste gevolgen hebben. Daarom is het elegant introduceren en correct afhandelen van effecten een belangrijke open vraag in het onderzoek naar programmeertaal. Met een goede programmeertaal kunnen effecten worden geïsoleerd, gecontroleerd en beheerd op een voorspelbare manier. Het maakt het redeneren, uitbreiden, testen en onderhouden van programma's met effecten duidelijk en efficiënt voor de programmeur. Expliciete constructies voor het redeneren over effecten zijn essentieel.

De meest verspreide aanpak is de monad [4], met als bekendste voorbeelden Optional in Java, Result in Rust, en de IO monad en de Maybe monad in Haskell. In Haskell zijn monad transformers [3] populair om modulaire compositie van effecten te realiseren. Dit gebeurt door verschillende types monads te combineren tot een enkele, gecombineerde monad.

1.2 Effect handlers

Een tweede dergelijke constructie is het gebruik van algebraïsche effecten [6] en effect handlers. Het concept is om de aanroep van het effect, de syntaxis, te scheiden van de afhandeling van het effect, de semantiek. Een effect handler kan worden beschouwd als een functie die verantwoordelijk is voor de afhandeling van een effect in een andere functie, zodat dit op een voorspelbare en modulaire manier kan gebeuren. Voorbeelden van effect handlers zijn de try/catch constructie in Java en het async/await patroon in JavaScript.

De belangrijkste eigenschap van programmeren met effect handlers is de hoge mate van modulariteit en diversiteit van semantiek die de programmeur kan bereiken met dezelfde bouwstenen. Dit vloeit voort uit overloading van effecten via verschillende handlers, modulaire compositie van effecten door verschillende monads te combineren tot een enkele, samengevoegde monad, en scheiding van syntax en semantiek bij de afhandeling van effecten. Deze eigenschappen maken dit tot een veelbelovende aanpak voor het programmeren met effecten.

Algebraïsche effect handlers [1] handelen de effecten af die het resultaat zijn van de output van algebraïsche bewerkingen. Niet alle computationele effecten kunnen echter worden gemodelleerd als algebraïsche effecten. Het toevoegen van andere families van effecten is een optie om deze beperking te overwinnen. Andere effecten stellen andere eisen aan de handlers die deze effecten afhandelen.

Scoped effecten [2], [7], [9], [5] zijn effecten waarbij het gedrag in scope beperkt is tot een deel van de totale berekening. Dit type effect verdeelt het programma in een computatie die binnen het bereik van het effect valt en een deel dat buiten het bereik valt. De λ_{sc} calculus [2] beschrijft een calculus voor algebraïsche en scoped effecten.

Parallelle algebraïsche effecten [8] zijn effecten die parallel behandeld kunnen worden. In de effect handler aanpak worden effecten standaard sequentieel behandeld door de handler. De λ_p calculus maakt parallelle afhandeling van effecten mogelijk en lever performantieverbeteringen op voor parallelliseerbare algebraïsche effecten.

Calculi van effect handlers zouden ideaal gezien drie eigenschappen moeten bezitten.

- Overloading van de effecten: De effecten krijgen een verschillende semantiek door het schrijven en gebruiken van verschillende handlers voor hetzelfde programma.
- Modulaire compositie: Het modulair combineren van bewerkingen van verschillende effecten is mogelijk zonder strikte beperkingen.
- Effectinteracties: Effecten interacteren met elkaar, waarbij de volgorde van de verschillende handlers een rol speelt. De interactie tussen effecten biedt nieuwe

semantische mogelijkheden.

Het scheiden van operaties en hun behandeling, respectievelijk in de effecten en de handlers, zorgt voor overloading en de modulaire compositie.

1.3 Doel

Er is een gebrek aan literatuur voor een calculus die algebraïsche, scoped en parallelle algebraïsche effecten combineert. De onderzoeksvraag voor deze masterproef luidt:

Hoe kan een minimale calculus worden gedefinieerd die sequentiële en parallelle afhandeling van algebraïsche en scoped effecten modelleert met behoud van handler overloading, modulaire compositie en effect interacties?

1.4 Overzicht

Hoofdstuk 2 geeft meer achtergrondinformatie over de onderwerpen die in deze masterproef worden behandeld. Vervolgens wordt in 3 de literatuur besproken waarop deze thesis voortbouwt. De motivatie voor dit onderwerp en de uitdagingen zijn het onderwerp van 4. De volgende hoofdstukken vormen de bijdrage van deze masterproef met de syntaxis in 5, de operationele semantiek in 6, het type-en effect-systeem in 7, voorbeelden in 8 en de metatheorie in 9 voor de uitgebreide calculus. Vervolgens geeft hoofdstuk 10 een evaluatie van de masterproef, hoofdstuk 11 geeft een overzicht van gerelateerd werk, en tot slot bevat hoofdstuk 12 de conclusie.

Hoofdstuk 2

Achtergrond

2.1 Effect handlers

Effect handlers zijn een mechanisme om effecten gestructureerd te behandelen in een programmeertaal. Ze stellen de programmeur in staat om de pure, functionele code in het programma te scheiden van de impure, effectvolle code. Dit gebeurt door specifieke functies, effect handlers te definiëren die de effecten afhandelen. Effect handlers maken het makkelijker over de logica van de code te redeneren.

Effect handlers worden gebruikt om voor elk effect dat kan voorkomen in een programma de instructies te beschrijven die moeten uitgevoerd wanneer dat effect optreedt. Dit maakt de scheiding mogelijk tussen het effect en de pure berekening, waardoor het gemakkelijker wordt te redeneren over de effecten in een programma en deze te beheren.

Effect handlers bieden een manier om effecten te coderen en te manipuleren op een modulair componeerbare manier, waardoor meer modulaire en herbruikbare programma's kunnen worden gemaakt. Dit is bijzonder interessant wanneer complexe effecten moeten worden gecontroleerd, zoals in parallelle of gedistribueerde systemen.

In het algemeen biedt het gebruik van effect handlers een declaratieve en gestructureerde benadering van het beheer van effecten, waardoor programmeurs meer modulariteit, onderhoudbaarheid en voorspelbaarheid bekomen in hun programma's.

2.2 Algebraïsche effecten

Algebraïsche effecten, een theoretisch kader voor het beschrijven van en redeneren over computationele effecten in programmeertalen. Algebraïsche effect handlers kunnen effecten modelleren zoals onder andere I/O, het veranderen van de staat van variabelen en niet-determinisme.

Algebraïsche effecten zijn beperkt in hun vermogen om bepaalde soorten effecten te modelleren. Meer bepaald kunnen algebraïsche effecten geen effecten voorstellen die de controlestroom van een programma veranderen, zoals uitzonderingen, met uitzondering van zwakke uitzonderingen, niet-deterministische vertakkingen en onderbrekingen. Deze beperking vloeit voort uit de aard van algebraïsche effecten, die gericht zijn op het modelleren van effecten die abstract kunnen worden voorgesteld als algebraïsche bewerkingen en compositioneel kunnen worden gecombineerd met andere effecten. Controlestroom-veranderende effecten daarentegen vereisen meer verfijnde mechanismen voor hun behandeling, die buiten de mogelijkheden van algebraïsche effecten vallen.

2.3 Calculus

...

2.3.1 Syntax

...

2.3.2 Operationele semantiek

...

2.3.3 Type- en effect-systeem

...

2.3.4 Metatheorie

...

2.4 Scoped Effecten

...

2.5 Parallelle Effecten

...

Hoofdstuk 3

Startpunt

3.1 λ_{sc}: A Calculus for Scoped Effects & Handlers

TODO

3.1.1 Veranderingen tegenover een calculus voor algebraïsche effecten

TODO

3.2 λ_p: Parallel Algebraic Effect Handlers

TODO

3.2.1 Veranderingen tegenover een calculus voor algebraïsche effecten

TODO

Hoofdstuk 4

Motivatie en Uitdagingen

TODO

4.1 Beperkingen op parallelle uitvoering effecten

TODO

4.1.1 Voorbeeld

4.2 Parallelle semantiek inbouwen in de calculus

TODO

Hoofdstuk 5

Syntaxis

TODO

Hoofdstuk 6

Operationele Semantiek

TODO

6.1 Letrec: Recursieve functies

TODO

6.2 Parallelle semantiek

TODO

Hoofdstuk 7

Type- en Effect-Systeem

TODO

7.1 Lijsten

TODO

Hoofdstuk 8

Uitgewerkte Voorbeelden

TODO

8.1 Voorbeelden met Scoped Effecten

TODO

8.2 Voorbeelden met Scoped Effecten

TODO

8.3 Voorbeelden met Beide Effecten

TODO

Hoofdstuk 9

Metatheorie

TODO

9.1 Lemma's

TODO

9.2 Behoud

TODO

9.3 Vooruitgang

TODO

Hoofdstuk 10

Evaluatie

TODO

10.1 Bijdrage

TODO

10.2 Bruikbaarheid

TODO

10.3 Correctheid

TODO

10.4 Implementatie

TODO

10.5 Backwards compatibility

TODO

10.5.1 λ_{sc}

TODO

10.5.2 λ_p

TODO

Hoofdstuk 11

Gerelateerd Werk

TODO

Hoofdstuk 12

Besluit

TODO

12.1 Resultaten

TODO

12.2 Beperkingen

TODO

12.3 Toekomstig werk

TODO

Bijlagen

Bibliografie

[1] A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming*, 84:108–123, 1 2015.

[2] R. Bosman, B. van den Berg, W. Tang, and T. Schrijvers. A calculus for scoped effects& handlers. 2022.

[3] S. Liang, P. Hudak, and M. Jones. Monad transformers and modular interpreters. pages 333–343. Association for Computing Machinery, 1995.

[4] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[5] M. Piróg, T. Schrijvers, N. Wu, and M. Jaskelioff. Syntax and semantics for operations with scopes. pages 809–818. Association for Computing Machinery, 2018.

[6] M. Pretnar. An introduction to algebraic effects and handlers. invited tutorial paper. *Electronic Notes in Theoretical Computer Science*, 319:19–35, 12 2015.

[7] N. Wu, T. Schrijvers, and R. Hinze. Effect handlers in scope. *SIGPLAN Not.*, 49:1–12, 9 2014.

[8] N. Xie, D. D. Johnson, D. Maclaurin, and A. Paszke. Parallel algebraic effect handlers. arXiv, 2021.

[9] Z. Yang, M. Paviotti, N. Wu, B. van den Berg, and T. Schrijvers. Structured handling of scoped effects: Extended version, 2022.