Type theory is a fast-evolving field at the crossroads of logic, computer science and mathematics. This gentle step-by-step introduction is ideal for graduate students and researchers who need to understand the ins and outs of the mathematical machinery, the role of logical rules therein, the essential contribution of definitions and the decisive nature of well-structured proofs.

The authors begin with untyped lambda calculus and proceed to several fundamental type systems, including the well-known and powerful Calculus of Constructions. The book also covers the essence of proof checking and proof development, and the use of dependent type theory to formalise mathematics.

The only prerequisite is a basic knowledge of undergraduate mathematics. Carefully chosen examples illustrate the theory throughout. Each chapter ends with a summary of the content, some historical context, suggestions for further reading and a selection of exercises to help readers familiarise themselves with the material.

Rob Nederpelt was Lecturer in Logic for Computer Science until his retirement. Currently he is a guest researcher in the Faculty of Mathematics and Computer Science at Eindhoven University of Technology, the Netherlands.

Herman Geuvers is Professor in Theoretical Informatics at the Radboud University Nijmegen, and Professor in Proving with Computer Assistance at Eindhoven University of Technology, both in the Netherlands.
TYPE THEORY AND FORMAL PROOF

An Introduction

ROB NEDERPEL'T
Eindhoven University of Technology,
The Netherlands

HERMAN GEUVERS
Radboud University Nijmegen,
and
Eindhoven University of Technology,
The Netherlands
University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.
It furthers the University’s mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org

Information on this title: www.cambridge.org/9781107036505

© Rob Nederpelt and Herman Geuvers 2014

This publication is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without the written
permission of Cambridge University Press.

First published 2014

Printed in the United Kingdom by CPI Group Ltd, Croydon CR0 4YY

A catalogue record for this publication is available from the British Library

Library of Congress Cataloguing in Publication data


Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party internet websites referred to in this publication, and does not guarantee that any content on such websites is, or will remain, accurate or appropriate.
To the memory of N.G. de Bruijn
## Contents

*Foreword, by Henk Barendregt*  
page xiii

*Preface*  
xv

*Acknowledgements*  
xxvii

*Greek alphabet*  
xxviii

### 1 Untyped lambda calculus

1.1 Input–output behaviour of functions  
1

1.2 The essence of functions  
2

1.3 Lambda-terms  
4

1.4 Free and bound variables  
8

1.5 Alpha conversion  
9

1.6 Substitution  
11

1.7 Lambda-terms modulo $\alpha$-equivalence  
14

1.8 Beta reduction  
16

1.9 Normal forms and confluence  
19

1.10 Fixed Point Theorem  
24

1.11 Conclusions  
26

1.12 Further reading  
27

Exercises  
29

### 2 Simply typed lambda calculus

2.1 Adding types  
33

2.2 Simple types  
34

2.3 Church-typing and Curry-typing  
36

2.4 Derivation rules for Church’s $\lambda \rightarrow$  
39

2.5 Different formats for a derivation in $\lambda \rightarrow$  
44

2.6 Kinds of problems to be solved in type theory  
46

2.7 Well-typedness in $\lambda \rightarrow$  
47

2.8 Type Checking in $\lambda \rightarrow$  
50

2.9 Term Finding in $\lambda \rightarrow$  
51
Contents

2.10 General properties of $\lambda\to$ 53
2.11 Reduction and $\lambda\to$ 59
2.12 Consequences 63
2.13 Conclusions 64
2.14 Further reading 65
Exercises 66

3 Second order typed lambda calculus 69
3.1 Type-abstraction and type-application 69
3.2 $\Pi$-types 71
3.3 Second order abstraction and application rules 72
3.4 The system $\lambda^2$ 73
3.5 Example of a derivation in $\lambda^2$ 76
3.6 Properties of $\lambda^2$ 78
3.7 Conclusions 80
3.8 Further reading 80
Exercises 82

4 Types dependent on types 85
4.1 Type constructors 85
4.2 Sort-rule and var-rule in $\lambda\omega$ 88
4.3 The weakening rule in $\lambda\omega$ 90
4.4 The formation rule in $\lambda\omega$ 93
4.5 Application and abstraction rules in $\lambda\omega$ 94
4.6 Shortened derivations 95
4.7 The conversion rule 97
4.8 Properties of $\lambda\omega$ 99
4.9 Conclusions 100
4.10 Further reading 100
Exercises 101

5 Types dependent on terms 103
5.1 The missing extension 103
5.2 Derivation rules of $\lambda P$ 105
5.3 An example derivation in $\lambda P$ 107
5.4 Minimal predicate logic in $\lambda P$ 109
5.5 Example of a logical derivation in $\lambda P$ 115
5.6 Conclusions 118
5.7 Further reading 119
Exercises 121

6 The Calculus of Constructions 123
6.1 The system $\lambda C$ 123
6.2 The $\lambda$-cube 125
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Properties of $\lambda C$</td>
<td>128</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusions</td>
<td>132</td>
</tr>
<tr>
<td>6.5</td>
<td>Further reading</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>134</td>
</tr>
<tr>
<td>7</td>
<td>The encoding of logical notions in $\lambda C$</td>
<td>137</td>
</tr>
<tr>
<td>7.1</td>
<td>Absurdity and negation in type theory</td>
<td>137</td>
</tr>
<tr>
<td>7.2</td>
<td>Conjunction and disjunction in type theory</td>
<td>139</td>
</tr>
<tr>
<td>7.3</td>
<td>An example of propositional logic in $\lambda C$</td>
<td>144</td>
</tr>
<tr>
<td>7.4</td>
<td>Classical logic in $\lambda C$</td>
<td>146</td>
</tr>
<tr>
<td>7.5</td>
<td>Predicate logic in $\lambda C$</td>
<td>150</td>
</tr>
<tr>
<td>7.6</td>
<td>An example of predicate logic in $\lambda C$</td>
<td>154</td>
</tr>
<tr>
<td>7.7</td>
<td>Conclusions</td>
<td>157</td>
</tr>
<tr>
<td>7.8</td>
<td>Further reading</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>162</td>
</tr>
<tr>
<td>8</td>
<td>Definitions</td>
<td>165</td>
</tr>
<tr>
<td>8.1</td>
<td>The nature of definitions</td>
<td>165</td>
</tr>
<tr>
<td>8.2</td>
<td>Inductive and recursive definitions</td>
<td>167</td>
</tr>
<tr>
<td>8.3</td>
<td>The format of definitions</td>
<td>168</td>
</tr>
<tr>
<td>8.4</td>
<td>Instantiations of definitions</td>
<td>170</td>
</tr>
<tr>
<td>8.5</td>
<td>A formal format for definitions</td>
<td>172</td>
</tr>
<tr>
<td>8.6</td>
<td>Definitions depending on assumptions</td>
<td>174</td>
</tr>
<tr>
<td>8.7</td>
<td>Giving names to proofs</td>
<td>175</td>
</tr>
<tr>
<td>8.8</td>
<td>A general proof and a specialised version</td>
<td>178</td>
</tr>
<tr>
<td>8.9</td>
<td>Mathematical statements as formal definitions</td>
<td>180</td>
</tr>
<tr>
<td>8.10</td>
<td>Conclusions</td>
<td>182</td>
</tr>
<tr>
<td>8.11</td>
<td>Further reading</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>185</td>
</tr>
<tr>
<td>9</td>
<td>Extension of $\lambda C$ with definitions</td>
<td>189</td>
</tr>
<tr>
<td>9.1</td>
<td>Extension of $\lambda C$ to the system $\lambda D_0$</td>
<td>189</td>
</tr>
<tr>
<td>9.2</td>
<td>Judgements extended with definitions</td>
<td>190</td>
</tr>
<tr>
<td>9.3</td>
<td>The rule for adding a definition</td>
<td>192</td>
</tr>
<tr>
<td>9.4</td>
<td>The rule for instantiating a definition</td>
<td>193</td>
</tr>
<tr>
<td>9.5</td>
<td>Definition unfolding and $\delta$-conversion</td>
<td>197</td>
</tr>
<tr>
<td>9.6</td>
<td>Examples of $\delta$-conversion</td>
<td>200</td>
</tr>
<tr>
<td>9.7</td>
<td>The conversion rule extended with $\Delta$</td>
<td>202</td>
</tr>
<tr>
<td>9.8</td>
<td>The derivation rules for $\lambda D_0$</td>
<td>203</td>
</tr>
<tr>
<td>9.9</td>
<td>A closer look at the derivation rules of $\lambda D_0$</td>
<td>204</td>
</tr>
<tr>
<td>9.10</td>
<td>Conclusions</td>
<td>206</td>
</tr>
<tr>
<td>9.11</td>
<td>Further reading</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>208</td>
</tr>
</tbody>
</table>
Contents

10 Rules and properties of $\lambda D$
10.1 Descriptive versus primitive definitions 211
10.2 Axioms and axiomatic notions 212
10.3 Rules for primitive definitions 214
10.4 Properties of $\lambda D$ 215
10.5 Normalisation and confluence in $\lambda D$ 219
10.6 Conclusions 221
10.7 Further reading 221
Exercises 223

11 Flag-style natural deduction in $\lambda D$
11.1 Formal derivations in $\lambda D$ 225
11.2 Comparing formal and flag-style $\lambda D$ 228
11.3 Conventions about flag-style proofs in $\lambda D$ 229
11.4 Introduction and elimination rules 232
11.5 Rules for constructive propositional logic 234
11.6 Examples of logical derivations in $\lambda D$ 237
11.7 Suppressing unaltered parameter lists 239
11.8 Rules for classical propositional logic 240
11.9 Alternative natural deduction rules for $\lor$ 243
11.10 Rules for constructive predicate logic 246
11.11 Rules for classical predicate logic 249
11.12 Conclusions 252
11.13 Further reading 253
Exercises 254

12 Mathematics in $\lambda D$: a first attempt
12.1 An example to start with 257
12.2 Equality 259
12.3 The congruence property of equality 262
12.4 Orders 264
12.5 A proof about orders 266
12.6 Unique existence 268
12.7 The descriptor $\iota$ 271
12.8 Conclusions 274
12.9 Further reading 275
Exercises 276

13 Sets and subsets 279
13.1 Dealing with subsets in $\lambda D$ 279
13.2 Basic set-theoretic notions 282
13.3 Special subsets 287
13.4 Relations 288
Contents

13.5 Maps 291
13.6 Representation of mathematical notions 295
13.7 Conclusions 296
13.8 Further reading 297
Exercises 302

14 Numbers and arithmetic in λD 305
14.1 The Peano axioms for natural numbers 305
14.2 Introducing integers the axiomatic way 308
14.3 Basic properties of the ‘new’ N 313
14.4 Integer addition 316
14.5 An example of a basic computation in λD 320
14.6 Arithmetical laws for addition 322
14.7 Closure under addition for natural and negative numbers 324
14.8 Integer subtraction 327
14.9 The opposite of an integer 330
14.10 Inequality relations on Z 332
14.11 Multiplication of integers 335
14.12 Divisibility 338
14.13 Irrelevance of proof 340
14.14 Conclusions 341
14.15 Further reading 343
Exercises 344

15 An elaborated example 349
15.1 Formalising a proof of Bézout’s Lemma 349
15.2 Preparatory work 352
15.3 Part I of the proof of Bézout’s Lemma 354
15.4 Part II of the proof 357
15.5 Part III of the proof 360
15.6 The holes in the proof of Bézout’s Lemma 363
15.7 The Minimum Theorem for Z 364
15.8 The Division Theorem 369
15.9 Conclusions 371
15.10 Further reading 373
Exercises 376

16 Further perspectives 379
16.1 Useful applications of λD 379
16.2 Proof assistants based on type theory 380
16.3 Future of the field 384
16.4 Conclusions 386
16.5 Further reading 387
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Logic in $\lambda D$</td>
<td>391</td>
</tr>
<tr>
<td>A.1</td>
<td>Constructive propositional logic</td>
<td>391</td>
</tr>
<tr>
<td>A.2</td>
<td>Classical propositional logic</td>
<td>393</td>
</tr>
<tr>
<td>A.3</td>
<td>Constructive predicate logic</td>
<td>395</td>
</tr>
<tr>
<td>A.4</td>
<td>Classical predicate logic</td>
<td>396</td>
</tr>
<tr>
<td>B</td>
<td>Arithmetical axioms, definitions and lemmas</td>
<td>397</td>
</tr>
<tr>
<td>C</td>
<td>Two complete example proofs in $\lambda D$</td>
<td>403</td>
</tr>
<tr>
<td>C.1</td>
<td>Closure under addition in $\mathbb{N}$</td>
<td>403</td>
</tr>
<tr>
<td>C.2</td>
<td>The Minimum Theorem</td>
<td>405</td>
</tr>
<tr>
<td>D</td>
<td>Derivation rules for $\lambda D$</td>
<td>409</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>411</td>
</tr>
<tr>
<td>Index of names</td>
<td></td>
<td>419</td>
</tr>
<tr>
<td>Index of definitions</td>
<td></td>
<td>421</td>
</tr>
<tr>
<td>Index of symbols</td>
<td></td>
<td>423</td>
</tr>
<tr>
<td>Index of subjects</td>
<td></td>
<td>425</td>
</tr>
</tbody>
</table>
This book, *Type Theory and Formal Proof: An Introduction*, is a gentle, yet profound, introduction to systems of types and their inhabiting lambda-terms. The book appears shortly after *Lambda Calculus with Types* (Barendregt et al., 2013). Although these books have a partial overlap, they have very different goals. The latter book studies the mathematical properties of some formalisms of types and lambda-terms. The book in your hands is focused on the use of types and lambda-terms for the complete formalisation of mathematics. For this reason it also treats higher order and dependent types. The act of defining new concepts, essential for mathematical reasoning, forms an integral part of the book. Formalising makes it possible that arbitrary mathematical concepts and proofs be represented on a computer and enables a machine verification of the well-formedness of definitions and of the correctness of proofs. The resulting technology elevates the subject of mathematics and its applications to its maximally complete and reliable form.

The endeavour to reach this level of precision was started by Aristotle, by his introduction of the axiomatic method and quest for logical rules. For classical logic Frege completed this quest (and Heyting for the intuitionistic logic of Brouwer). Frege did not get far with his intended formalisation of mathematics: he used an inconsistent foundation. In 1910 Whitehead and Russell introduced types to remedy this. These authors made proofs largely formal, except that substitutions still had to be understood and performed by the reader. In 1940 Church published a system with types, based on a variant of those of Whitehead and Russell, in which the mechanism of substitution was captured by lambda-terms and conversion. Around 1970 de Bruijn essentially extended the formalism of types by introducing *dependent types* with the explicit goal to formalise and verify mathematics. By 2004 this technique was perfected and George Gonthier established, using the mathematical assistant Coq, a full formalisation of the Four Colour Theorem.

The learning curve to formalise remains steep, however. One still needs to be
an expert in a mathematical assistant in order to apply the technique. I hope and expect that this book will contribute to the familiarisation of formalising mathematical proofs and to improvements in the mathematical assistants, bringing this technique within the reach of the working mathematician and computer scientist.

Henk Barendregt
Preface

Aim and scope

The aim of the book is, firstly, to give an introduction to type theory, an evolving scientific field at the crossroads of logic, computer science and mathematics. Secondly, the book explains how type theory can be used for the verification of mathematical expressions and reasonings.

Type theory enables one to provide a ‘coded’ version – i.e. a full formalisation – of many mathematical topics. The formal system underlying type theory forces the user to work in a very precise manner. The real power of type theory is that well-formedness of the formalised expressions implies logical and mathematical correctness of the original content.

An attractive property of type theory is that it becomes possible and feasible to do the encoding in a ‘natural’ manner, such that one follows (and recognises) the way in which these subjects were presented originally. Another important feature of type theory is that proofs are treated as first-class citizens, in the sense that proofs do not remain meta-objects, but are coded as expressions (terms) of the same form as the rest of the formalisation.

The authors intend to address a broad audience, ranging from university students to professionals. The exposition is gentle and gradual, developing the material at a steady pace, with ample examples and comments, cross-references and motivations. Theoretical issues relevant for logic and computer science alternate with practical applications in the area of fundamental mathematical subjects.

History Important investigations in the machinery of logic were made by F.L.G. Frege, as early as the end of the nineteenth century (Frege, 1893). Formal mathematics started with B. Russell in the first decade of the twentieth century, by the publication of the famous Principia Mathematica (The Principles of Mathematics, see Russell, 1903). Other contributions were made by D. Hilbert (Hilbert, 1927) in the 1920s. An important step in the description of
the essential mechanisms behind the mathematical way of thought was made by A. Church in the 1940s (Church, 1940). He invented the lambda calculus, an abstract mechanism for dealing with functions, and he introduced ‘simple type theory’ as the language for higher order logic.

At the end of the 1960s N.G. de Bruijn designed his ‘mathematical language’, Automath (de Bruijn, 1970), which he and his group tested thoroughly by translating a broad corpus of mathematical texts in it, and verifying it by means of a computer program. In the same period, the Polish Mizar group (Mizar, 1989) developed a language and a program to develop and store mathematical theories; they founded their efforts, however, not so much in a type-theoretic theory.

From approximately the 1980s there was an explosion of work in the area of type theory based on earlier work in the 1970s by J.-Y. Girard (Girard, 1986) and P. Martin-Löf (Martin-Löf, 1980). We mention the inspiring work of H.P. Barendregt, whose lambda-cube and the notion of Pure Type Systems based on that are by now a standard in the world of type theory (Barendregt, 1981, 1992).

The present book has been built on both Automath and the lambda-cube, which have been combined into a novel, concise system that enjoys the advantages of both respected predecessors.

**Rationale** Topics such as proven correctness and complete formalisation are essential in many areas of modern science. Type theory as an all-encompassing formalism has become more and more a standard benchmark for what formalisation of logico-mathematical content really means, and the more so because it also includes the essence of what a formal proof is. Thus, type theory is a valuable expedient to transform ‘correctness’ into a mechanisable issue, which is of great importance, in particular in mathematical proof development and correct computer programming.

There are many developments that build on the inherent force of type theory. We mention work on proving program correctness; on correct program construction; on automation of reasoning; on formalisation and archiving of mathematical subjects, including on-line consultable libraries of knowledge; on proof checking, (assistance for) proof development and construction. More about these subjects can be found in Chapter 16, in particular Sections 16.2 and 16.3.

For the benefit of any interested person who desires to get insight into the ‘big points’ of type theory, we note the following. Notwithstanding the momentum that formalisation of mathematics has gained, especially in computer science where it is used for the verification of software and systems by means of proof assistants, formalising is a considerable effort. For students and also for
Preface

interested researchers, it is still a major investment to understand and use the systems. We expect this situation to improve in the future and this is where our book aims to fill a gap, by being a gentle introduction to the art of formalising mathematics on the basis of type theory, suitable for proof assistants and other systems. We believe that this book will be very useful for anyone starting to use such a system.

Approach This textbook describes a concise version of type theory that is immediately usable to represent mathematical theories and to verify them. The representation is close to the manner in which mathematicians write and think, and therefore easier to master and employ. It is a good means for students of mathematics and computer science to make a personal acquaintance with the ins and outs of the mathematical machinery, the role of logical rules therein, the essential contribution of definitions and the decisive nature of well-structured proofs.

For that purpose we build the material from scratch, gradually enlarging the influence of the types in the various systems described. The text starts with the untyped lambda calculus and then introduces several fundamental type systems, culminating in the well-known and powerful Calculus of Constructions. We continue by extending that system with a formal definition system and consecutively test the newly obtained system with several mathematical subjects, until we finally present a substantial piece of mathematics (Bézout’s theorem) in the format described, in order to give a practical demonstration of how the formal system works, and how close the formal translation remains to the usual mathematical way of expressing such an item.

The main thread that runs through all the chapters is the development of a convincing and viable formal type system for mathematics. At the end of each chapter, the results are summarised in a section entitled Conclusions. In the final section of each chapter, called Further Reading, we look around, sketching a broader picture: we give a short historical justification of the topics described, an overview of the essential aspects of related research (past and present) not dealt with in the chapter, and suggest other literature as further reading to the interested reader.

Following each chapter there is a series of exercises, enabling the reader to get acquainted with the presented material. The exercises concentrate on the subjects treated in the chapter text as such (not in the Further Reading). Since we aim at a ‘generic’ approach to type theory as the basis of logical proofs and of mathematics in general, the exercises do not refer to, or make use of, specific proof assistants or software tools: we regard it as sensible to remain independent of the actual technical developments.

For answers to selected exercises, see www.win.tue.nl/~wsinrpn/.
Preface

Summary of contents

Chapter 1: Untyped lambda calculus
We start with an exposition of untyped lambda calculus, a fundamental topic originating from A. Church in the 1930s, which may be regarded as the calculus underlying the behaviour of functions, including variable binding and substitution – essential concepts in mathematics and computer science. The standard subjects in this area are discussed in detail, including $\beta$-reduction, normal forms, confluence, fixed points and the related theorems. We list the positive and negative aspects of this calculus.

Chapters 2 to 6
The drawbacks of the untyped calculus lead to a notion of type, which plays the main role in the rest of the book. In Chapters 2 to 6 we present the standard hierarchy of typed lambda calculi, as elaborated by H.P. Barendregt. In these chapters we introduce several systems, each with its own rationale, and contrast them with previous and coming ones as to their relative ‘power’. Moreover, the reader becomes acquainted with derivation rules and their use, and with basic logical entities and their formal role in reasonings. The relevant properties of these systems are reviewed, with a selection of instructive proofs of these properties.

Chapter 2: Simply typed lambda calculus
In Chapter 2 we develop the simply typed lambda calculus in the explicit version, due to A. Church, which is called $\lambda\to$. We also mention the implicit version of H.B. Curry. We give a derivation system for $\lambda\to$ and examples of its use. The properties of the system are given, and contrasted with the properties of the untyped lambda calculus.

Chapter 3: Second order typed lambda calculus
We extend Church’s $\lambda\to$ with terms depending on types, leading to the system $\lambda^2$, enjoying second order abstraction and application, and having $\Pi$-types. Again, examples show its usefulness.

Chapter 4: Types dependent on types
We extend $\lambda\to$ in another direction, adding types depending on types. Therefore we develop the notions ‘type constructor’ and ‘kind’. Thus we obtain the system $\lambda\omega$. We also adapt the derivation rules and include a conversion rule.

Chapter 5: Types dependent on terms
A third extension of $\lambda\to$ leads to $\lambda\Pi$, which enables us to formalise predicates. The far-reaching propositions-as-types concept, implying the Curry–Howard isomorphism, is one of the topics that pop up in a straightforward fashion. We discuss the correspondence with basic mathematical and logical notions.
Chapter 6: The Calculus of Constructions
Chapters 2 to 5 culminate in the powerful Calculus of Constructions (or $\lambda C$), the theory on which the well-known proof assistant Coq has been built. In this chapter we explain the hierarchy and the structure present in the Barendregt cube, and we add the corresponding derivation system. The relevant properties are listed and contrasted with earlier results in the book.

Chapter 7: The encoding of logical notions in $\lambda C$
We demonstrate how propositional logic and predicate logic fit naturally in the $\lambda C$ framework. For each of the usual logical connectives and quantifiers we give a type-theoretic encoding, for which we employ several times the second order possibilities of $\lambda C$. Constructive logic is separated from classical logic, which needs a single axiom. A number of examples show how logical proofs can be embedded in type theory, thus deepening the reader’s insight into logical reasonings and proofs.

Chapter 8: Definitions
In this chapter we look into the nature, the usage and the usefulness of definitions in logic and mathematics. We argue why one tends to give a specific object or notion a name of its own, often in a context of assumptions, and how these names are used afterwards. We explain the general format underlying the definition mechanism and discuss the various manners of instantiating these definitions. The differences and correspondences between variables, parameters and constants are reviewed, and we point at the possibility in type theory of giving names to proofs.

Chapter 9: Extension of $\lambda C$ with definitions
Here we extend the type theory $\lambda C$ with formal definitions, which is essential for making type theory practically useful. We formalise the common kind of definitions, which name a notion that is specified by a description. We discuss and analyse the extra derivation rules needed for the formal treatment of definitions: one for adding a definition, and one for instantiating a definition. We also introduce and elaborate a reduction mechanism for enabling the ‘unfolding’ of definitions, with a discussion of related notions. Finally, we obtain the system $\lambda D_0$, a formal extension of $\lambda C$ treating definitions as first-class citizens.

Chapter 10: Rules and properties of $\lambda D$
We introduce a second kind of definition, the primitive ones, which can be used for axioms and axiomatic notions. Their formal representation resembles the one for descriptive definitions, which is particularly apparent in the extra derivation rules necessary to encapsulate these primitive definitions. We thus obtain the system $\lambda D$ (i.e. $\lambda C$ + definitions + axioms) and we list the most important properties of the obtained formal system.
Preface

Chapter 11: Flag-style natural deduction in $\lambda D$
In order to demonstrate how $\lambda D$ works in practice, we start with a more thorough investigation of formal logic in $\lambda D$-style, as a sequel to Chapter 7. We illustrate how derivations in $\lambda D$ can be turned into a flag-style linear format, close to the familiar representation used in mathematics books, and therefore easy to understand. Natural deduction, the logical system that reflects the reasoning patterns employed by mathematicians, can be clearly presented in this format, as we demonstrate: the introduction and elimination rules for the standard connectives and quantifiers can be straightforwardly translated into $\lambda D$. Examples show that natural deduction nicely agrees with the ideas and constructions present in type theory.

Chapter 12: Mathematics in $\lambda D$: a first attempt
In this chapter we put $\lambda D$ to the test in the area of mathematics. A simple example, consisting of a theorem and a short proof, leads to investigations about equality, Leibniz’s law, and orders, all directly transposable in the $\lambda D$ setting. Next, we discuss unique existence and the possibility to attach a name to uniquely existing objects and notions. In order to formalise this, we axiomatically introduce Frege’s $\iota$-descriptor.

Chapter 13: Sets and subsets
We discuss how to deal with sets and subsets in type theory. This is not straightforward, since sets and types are different concepts. For example, an element can be a member of several sets, whereas in the standard version of type theory that we develop in this book, each ‘object’ has a unique type (up to conversion). A crucial property of types is that it is decidable whether ‘$a$ has type $T$’. For sets this is not the case: ‘$a \in X$’ is in general undecidable. This means we have to make a choice as to how to deal with sets in type theory. We make the choice to represent subsets as predicates over a type, which works out well, as we show with a number of examples. At the end of the chapter we compare our choice with other options.

Chapter 14: Numbers and arithmetic in $\lambda D$
In order to investigate the usefulness of $\lambda D$ for formalising mathematics in a systematic manner, we focus on the integer numbers. Starting with Peano’s axioms for the natural numbers, we give similar axioms for the integers, which turn out to be directly transposable into $\lambda D$. The natural numbers then form a subset, with the desired properties. Thereby we obtain a viable approach to many parts of number theory. In order to demonstrate this, we develop basic arithmetic for the integers, concerning addition, subtraction, multiplication and the like. When formalising recursion, we make good use of the $\iota$-descriptor introduced in a previous chapter. Inequalities, such as $\leq$, and divisibility follow relatively easy. We demonstrate the flexibility of the obtained formalisation.
by giving ample examples. In developing formal arithmetic, a long sequence of provable lemmas passes by, which shows the strength of the proposed encodings in type theory.

Chapter 15: An elaborated example
In order to demonstrate the reach and the power of the approach developed in this book, we formalise a non-trivial theorem and its proof in $\lambda D$. Therefore we take a version of Bézout’s Lemma: if two positive natural numbers are relatively prime, then there is a linear combination of the two which is equal to 1. We split the proof into a number of parts, which are formalised one by one. In the presentation we suppress a number of obvious details which are left to the reader, in order to keep a clear view of the overall picture. Our development of the proof of Bézout’s Lemma shows that the chosen road is viable and feasible. Two auxiliary theorems (the Minimum Theorem and the Division Theorem) are considered in detail and also transposed in a $\lambda D$ setting.

Chapter 16: Further perspectives
This chapter summarises the useful points of a type-theoretic formalisation of mathematics as, for example, offered by $\lambda D$. It also focuses on the general principles of type theory-based proof assistants and their power concerning proof checking and interactive proving. Finally, we outline our view on the future of the field, motivated by recent developments.

Appendix A: Logic in $\lambda D$
Summary of the natural deduction rules described and used in the book.

Appendix B: Arithmetical axioms, definitions and lemmas
For the reader’s convenience, we list the lemmas concerning arithmetic in $\mathbb{Z}$, as they are given in the text of Chapter 14.

Appendix C: Two complete example proofs in $\lambda D$
We give complete versions of two $\lambda D$ proofs dealt with before, namely of the Closure Property of Addition in $\mathbb{N}$, and of the Division Theorem, in order to show what such formal proofs look like when all relevant details have been worked out.

Appendix D: Derivation rules for $\lambda D$
This appendix summarises the derivation system as developed in the book, for easy reference.

Indexes
We add four sets: an Index of names (which contains the names of the persons mentioned in the main text of the book), an Index of definitions (listing the formal definitions presented in the figures of Chapters 8 to 15), an Index of symbols and an Index of subjects.
Preface

What’s new?

The book has several new aspects in its presentation of type theory:

(1) A main novelty is that we present type theory in *flag-derivation style*. In modern presentations of type theory, one usually sees derivations presented in ‘tree style’, which enables a concise and precise way of defining the rules of type theory, and we will also employ it for that purpose. A tree style, however, is very impractical for giving real derivations. Derivation trees become too wide and big and enforce all kinds of basic typings to be derived several times (in different branches of the derivation tree). The flag style allows easy reuse of context elements. It also allows us, in combination with definitions (see below), to reuse derived results. Altogether it is very close to the usual ‘book style’, which builds up the mathematics in a linear order.

(2) Also new is the inclusion of *definitions*, which are often regarded as informal abbreviation mechanisms on the meta-level. Contrary to this, we give a formal treatment of definitions and show how they are used in practical derivations. Primitive notions, which are treated in a similar manner to definitions in our presentation of type theory, are used for adding elementary concepts and axioms to type theory.

Inductive notions can be defined by means of higher order logic and our definition mechanism allows us to give them a name. One can also define recursive functions over these inductive notions. Therefore we do not have inductive definitions as a basic concept, since this is not needed.

(3) We continually take care to give a *stepwise and gentle explanation* of the rules of type theory, illustrated with many examples of how these are used to formalise mathematics. In particular, given that it is not very common to devote a lot of attention to the notion of ‘definition’, our explicit description in Chapter 8 of the use and meaning of definitions stands out as a new look at what they are and intend to be. We contrast definitions with assumptions and differentiate between variables, parameters and constants. Thus we discuss a number of well-known mathematical concepts with a ‘linguistic-philosophical flavour’, which is not usual.

(4) The manner in which we represent definitions leads to a relatively small but powerful *extension* of \(\lambda C\). The system obtained in Chapter 10, called \(\lambda D\), has a simple, convincing format and is new: it has not yet been described as such in the literature. In Chapter 12 we introduce the descriptor \(\iota\) in \(\lambda D\), which is straightforward, since \(\lambda D\) permits primitive concepts such as axioms and axiomatic notions. Use of this \(\iota\), which enables uniquely identifiable objects, is not very common in dependent type theory. We note, however, that L.S. van Benthem Jutting already did this in his Automath translation of E. Landau’s
book on Analysis (van Benthem Jutting, 1977), and that it is also present in the HOL system (see HOL system, 1988).

(5) At the end of Chapter 13 we give an overview discussion of how to deal with subsets in type theory. The new aspect is that we take a pragmatic viewpoint on subsets by making a conscious choice with a view to our aims.

Readership

Although the style of the book is expository, using a gradual pace of explanation with a continuous care for good understanding, including many examples, the subject has intrinsic difficulties because of the far-reaching and rather complex interweaving of type-related notions, which is inherent to type theory. Therefore, we consider this to be an advanced textbook. The intended readership may include certain undergraduate students, although the primary readership will range from graduate students to specialists:

– Undergraduate students in mathematics and computer science (second or third year) should be able to follow most of the text, from the beginning to the end. The main insight they get is to learn thoroughly what a proof ‘is’, how it should be read and how it can be obtained. Moreover, they see which ‘fine structure’ is present (albeit mostly hidden) behind a mathematical text; that extreme precision in mathematics is feasible; that logic can be framed as a highly systematic deduction apparatus; that proofs can be considered as mathematical objects, on a par with the usual mathematical entities; that definitions are an indispensable asset in the formal mathematical language.

– Graduate students in mathematics and computer science may profit more deeply from this book. They will enjoy the same benefits as sketched above, but they also learn from this book what the essence of types is, in particular of higher order and dependent types. This offers a useful lead into further investigations in type systems for programming languages. They get acquainted with important notions connected to function evaluation, by studying (typed and untyped) lambda calculus and their properties. They see, moreover, that (and how) mathematics can be formalised smoothly and hence that type theory is suitable for checking with computer assistance, for example by means of a proof assistant such as Coq.

– Specialists and researchers in computer science and mathematics not acquainted with type theory can get a good and thorough impression of what it’s all about. They can easily browse through the text to pick up the essentials that interest them, in particular with respect to the important range of type systems combined in Barendregt’s cube; the Calculus of Constructions and its properties; the essence and the value of (formal) definitions and how
they can form part of a fully formal derivation system; the various approaches
to formalising Set Theory; the approach to integers in the Peano style; and
a worked-out example of a real formalised proof of a well-known, non-trivial
theorem. As a by-product, the reader will understand the basic features of the
system Automath (de Bruijn, 1970). A researcher may also be inspired by the
possibility of formalising mathematics as demonstrated in this book, or by the
overview of applications and perspectives at the end of the book.

Technical level

Since the book concerns a relatively new and self-contained subject, developed
from scratch, no more prerequisites are required than a good knowledge of basic
mathematical material such as (undergraduate) algebra and analysis. Some
knowledge of logical systems and some experience with logico-mathematical
reasoning and/or proofs may help, but it is not mandatory.

About the authors

Rob Nederpelt (born 1942) is a guest researcher in the faculty of Mathematics
and Computer Science at the Eindhoven University of Technology (the Nether-
lands) and was, until his retirement, a lecturer in Logic for Computer Science
at the same university. He studied mathematics at Leiden University and ob-
tained his PhD at Eindhoven University in 1973, with N.G. de Bruijn as his
thesis supervisor. The subject of his thesis was (weak and strong) normalisa-
tion in a typed lambda calculus narrowly related to the mathematical language
Automath.

He has taught many courses at Eindhoven University, first in mathematics,
later in logic, theoretical computer science, type theory and the language of
mathematics.

His research interest is primarily logic, in particular type theory and typed
lambda calculus. See www.win.tue.nl/~wsinrpn/publications.htm for his
list of publications. It contains many papers and three books: The Language of
Mathematics (Nederpelt, 1987), A Modern Perspective on Type Theory (Ka-
mareddine et al., 2004), Logical Reasoning (Nederpelt & Kamareddine, 2011).
He has also been one of the editors of Selected Papers on Automath (Nederpelt
et al., 1994).

Herman Geuvers (born 1964) is professor in Theoretical Informatics at the
Radboud University Nijmegen, and in Proving with Computer Assistance at
the Eindhoven University of Technology, both in the Netherlands. He has stud-
ied mathematics in Nijmegen and wrote his PhD thesis in the Foundations of
Preface

Computer Science (Radboud University Nijmegen, the Netherlands; 1993) under the supervision of H.P. Barendregt on the Curry–Howard formulas-as-types interpretation that relates logic and type theory.

He has taught many courses, mainly on topics such as logic, theoretical computer science, type theory, proof assistants and semantics of programming languages. He has been a lecturer in type theory and proof assistants at various international PhD summer schools.

His research is in type theory, proof assistants and formalising mathematics. He has published over 60 papers (www.cs.ru.nl/~herman/pubs.html), has been a member of various programme committees and has organised various scientific events. Moreover, he co-edited the book Selected Papers on Automath (Nederpelt et al., 1994). He was the leader of the ‘FTA project’ at the Radboud University Nijmegen, to formalise a constructive proof of the Fundamental Theorem of Algebra in Coq. This led to CoRN, the Constructive Coq Repository of formalised mathematics (http://corn.cs.ru.nl) at Nijmegen, which is a large library of formalised algebra and analysis by means of the proof assistant Coq.
Acknowledgements

This book is dedicated to N.G. de Bruijn (1918–2012). His pioneering work in the 1960s on the ‘mathematical language’ Automath and on formalising mathematics in type theory has greatly influenced the development of this book. His creative mind has been a great inspiration for us.

We thank H.P. Barendregt for sharing his deep insights in typed lambda calculus and for his thorough and convincing description of many important subjects in the field.

Since one of us is a student of N.G. de Bruijn, and the other one of H.P. Barendregt, the publication of this book is a way of thanking our teachers, and paying tribute to them.

We are particularly grateful to J.R. Hindley and C. Hemerik for their comments on the text. We thank F. Dechesne and H.L. de Champeaux for their useful remarks, A. Visser and R. Iemhoff for their valuable help and P. van Tilburg for his nice and convenient tool for making flag derivations in \LaTeX.

We thank Eindhoven University of Technology for kindly offering us space, time and equipment to prepare the text. We are also grateful to Cambridge University Press, in particular to D. Tranah and G. Smith.

Rob Nederpelt, Herman Geuvers
For convenience, we list the letters of the Greek alphabet (small and capital) together with their names and English pronunciation.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Name</th>
<th>Pronunciation</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>alpha</td>
<td>/ælfə/</td>
</tr>
<tr>
<td>β</td>
<td>beta</td>
<td>/biːta/</td>
</tr>
<tr>
<td>γ</td>
<td>gamma</td>
<td>/ɡæmə/</td>
</tr>
<tr>
<td>δ</td>
<td>delta</td>
<td>/dɛltə/</td>
</tr>
<tr>
<td>ε</td>
<td>epsilon</td>
<td>/ɛpsɪlən/</td>
</tr>
<tr>
<td>ζ</td>
<td>zeta</td>
<td>/ziːta/</td>
</tr>
<tr>
<td>η</td>
<td>eta</td>
<td>/iːta/</td>
</tr>
<tr>
<td>θ</td>
<td>theta</td>
<td>/θiːta/</td>
</tr>
<tr>
<td>i</td>
<td>iota</td>
<td>/aɪəta/</td>
</tr>
<tr>
<td>κ</td>
<td>kappa</td>
<td>/ˈkæpə/</td>
</tr>
<tr>
<td>λ</td>
<td>lambda</td>
<td>/ˈlaɪmədə/</td>
</tr>
<tr>
<td>μ</td>
<td>nu</td>
<td>/njuː/</td>
</tr>
<tr>
<td>ν</td>
<td>nu</td>
<td>/njuː/</td>
</tr>
<tr>
<td>ξ</td>
<td>xi</td>
<td>/gziː/</td>
</tr>
<tr>
<td>Ψ</td>
<td>psi</td>
<td>/psiː/</td>
</tr>
<tr>
<td>Ω</td>
<td>omega</td>
<td>/ˈɔmɪɡə/</td>
</tr>
<tr>
<td>Π</td>
<td>pi</td>
<td>/piː/</td>
</tr>
<tr>
<td>Ρ</td>
<td>rho</td>
<td>/rəʊ/</td>
</tr>
<tr>
<td>Σ</td>
<td>sigma</td>
<td>/sɪɡmə/</td>
</tr>
<tr>
<td>Τ</td>
<td>tau</td>
<td>/toʊ/</td>
</tr>
<tr>
<td>Ψ</td>
<td>psi</td>
<td>/psiː/</td>
</tr>
<tr>
<td>Κ</td>
<td>kappa</td>
<td>/ˈkæpə/</td>
</tr>
<tr>
<td>Χ</td>
<td>chi</td>
<td>/kiː/</td>
</tr>
<tr>
<td>Ψ</td>
<td>psi</td>
<td>/psiː/</td>
</tr>
</tbody>
</table>