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 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 useable 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 and its proof) 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/`.

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 β-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 Π-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 $\lambda\underline{\omega}$. 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 $\lambda\mathrm{P}$, 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 λC), 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 λC
We demonstrate how propositional logic and predicate logic fit naturally in the λC 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 λC. 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 λC with definitions
Here we extend the type theory λC 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 λD_0, a formal extension of λC treating definitions as first-class citizens.

Chapter 10: Rules and properties of λD
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 λD (i.e. λC + definitions + axioms) and we list the most important properties of the obtained formal system.

Chapter 11: Flag-style natural deduction in λD
In order to demonstrate how λD works in practice, we start with a more thorough investigation of formal logic in λD-style, as a sequel to Chapter 7. We illustrate how derivations in λD 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 λD. Examples show that natural deduction nicely agrees with the ideas and constructions present in type theory.

Chapter 12: Mathematics in λD: a first attempt
In this chapter we put λD 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 λD 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 ι-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 λD
In order to investigate the usefulness of λD 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 λD. 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 ι-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 λD. 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 λD setting.

Chapter 16: Further perspectives
This chapter summarises the useful points of a type-theoretic formalisation of mathematics as, for example, offered by λD. 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 λD
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 λD
We give complete versions of two λD 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 λD
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.

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 λC. The system obtained in Chapter 10, called λD, 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 ι in λD, which is straightforward, since λD permits primitive concepts such as axioms and axiomatic notions. Use of this ι, 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 Netherlands) and was, until his retirement, a lecturer in Logic for Computer Science at the same university. He studied mathematics at Leiden University and obtained 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) normalisation 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* (Kamareddine *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 studied mathematics in Nijmegen and wrote his PhD thesis in the Foundations of

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.