Preface

This book is an introductory text on mathematical logic and type theory. It is aimed primarily at providing an introduction to logic for students of mathematics, computer science, or philosophy who are at the college junior, senior, or introductory graduate level. It can also be used as an introduction to type theory by researchers at more advanced levels.

The first part of the book (Chapters 1 and 2, supplemented by parts of Chapters 3 and 4) is suitable for use as a text in a one-semester introduction to mathematical logic, and the second part (Chapters 5 – 7) for a second semester course which introduces type theory (more specifically, typed λ-calculus) as a language for formalizing mathematics and proves the classical incompleteness and undecidability theorems in this context. Persons who wish to learn about type theory and have had a prior introduction to logic will have no difficulty starting with Chapter 5.

The book is oriented toward persons who wish to study logic as a vehicle for formal reasoning. It may be particularly useful to those who are interested in issues related to the problem of constructing formal proofs as models of informal reasoning or for use in computerized systems which involve automated deduction. Proofs, which are often the chief end products and principal manifestations of mathematical reasoning, constitute highly significant pathways to truth and are a central concern of this book. Our choice of the title *To Truth Through Proof* is motivated by the consideration that while in most realms one needs more than logic to achieve an understanding of what is true, in mathematics the primary and ultimate tool for establishing truth is logic.

Of course, the study of logic involves reasoning about reasoning, and it is not surprising that complex questions arise. To achieve deep understanding and proper perspective, one must study a variety of logical systems as mathematical objects and look at them from a variety of points of view. We are thus led to study the interplay between syntax and semantics, questions of consistency and independence, formal rules of reasoning, various formats for proofs and refutations, ways of representing basic mathematical concepts in a formal system, the notion of computability, and the completeness, incompleteness, and undecidability theorems which illuminate both the power and the limitations of logic.

One of the basic tasks of mathematical logic is the formalization of mathematical reasoning. Both type theory (otherwise known as higher-order logic) and axiomatic set theory can be used as formal languages for this

purpose, and it is really an accident of intellectual history that at present most logicians and mathematicians are more familiar with axiomatic set theory than with type theory. It is hoped that this book will help to remedy this situation.

In logic as in other realms, there is a natural tendency for people to prefer that with which they are most familiar. However, those familiar with both type theory and axiomatic set theory recognize that in some ways the former provides a more natural vehicle than the latter for formalizing what mathematicians actually do. Both logical systems are necessarily more restrictive than naive axiomatic set theory with the unrestricted Comprehension Axiom, since the latter theory is inconsistent. Axiomatic set theory achieves consistency by restricting the Comprehension Axiom and introducing the distinction between sets and classes. Mathematicians often find this distinction unnatural, ignore the technicalities about existence axioms, and leave it to the specialists to show that their reasoning can be justified. Type theory achieves consistency by distinguishing between different types of objects (such as numbers, sets of numbers, collections of sets of numbers, functions from numbers to numbers, sets of such functions, etc.). Mathematicians make such distinctions too, and even use different letters or alphabets to help them distinguish between different types of objects, so the restrictions which type theory introduces are already implicit in mathematical practice. While some formulations of type theory may seem cumbersome, the formulation $\mathcal{Q}_0$ introduced in Chapter 5 is a very rich and expressive language, with functions (which need not be regarded as sets of ordered pairs) of all types as primitive objects, so most of what mathematicians write can be translated into $\mathcal{Q}_0$ very directly. $\mathcal{Q}_0$ is a version of typed λ-calculus, and the availability of λ-notation in $\mathcal{Q}_0$ enables definitions to be handled very conveniently and eliminates the need for axioms asserting the existence of sets and functions.

One's choice of a formal language will generally depend on what one wishes to do with it. If one is choosing a language for expressing mathematics in computerized systems which deal with mathematics on a theoretical level (such as mathematically sophisticated information retrieval systems, proof checkers, or deductive aids), there are several reasons why type theory may be preferable to set theory. First, the primitive notation of set theory is so economical that this language is only practical to actually use if one expands it by introducing various abbreviations. This is usually done in an informal way (i.e., in the meta-language), but in an automated system it is important that the formal language be the one actually used to express mathematical statements, since this is the language which must be studied rigorously and

manipulated systematically. Thus, the formal language should not only be adequate in principle for expressing mathematical ideas, but it should also be convenient in practice. Indeed, the translation from familiar mathematical notations to the formal language should be as simple and direct as possible. Second, in set theory one can write all sorts of expressions which are not legal in type theory, and which a mathematician would reject as nonsense almost instinctively. A computer program for manipulating mathematical expressions would certainly be very awkward and inefficient if it did not have some way of distinguishing between different types of mathematical objects and rejecting nonsensical expressions. Third, one of the basic operations in automated theorem proving is that of finding appropriate substitutions for variables, and an understanding of the types of various entities allows one to avoid much useless search. Finally, it has been found (see [Andrews *et al.*, 1984] [Andrews *et al.*, 1996]) that unification algorithms for type theory (such as [Huet, 1975] and [Jensen and Pietrzykowski, 1976]) are powerful tools in automating the proofs of simple mathematical theorems.

As noted in [Hanna and Daeche, 1985], [Hanna and Daeche, 1986], and [Gordon, 1986], typed λ-calculus is particularly well suited for specifying and verifying hardware and software. Familiarity with typed λ-calculus also provides fundamental background for the study of denotational semantics for functional programming languages.

While Gödel's Completeness Theorem guarantees that all valid wffs of first-order logic have proofs in first-order logic, extraordinary reductions in the lengths of such proofs can sometimes be achieved by using higher-order logic. (See §54.)

It is easy to discuss the semantics of type theory, and to explain the crucial distinction between standard and nonstandard models which sheds so much light on the mysteries associated with the incompleteness theorems, Skolem's paradox, etc. As will be seen in Chapter 7, in the context of the language $\mathcal{Q}_0$ it is easy to present the incompleteness theorems very elegantly and to show that their significance extends far beyond the realm of formalizing arithmetic.

Obviously, serious students of mathematical logic ought to know about both type theory and axiomatic set theory. Pedagogically, it makes good sense to introduce them to elementary logic (propositional calculus and first-order logic), then type theory, then set theory. In spite of the superficial appearance of simplicity possessed by set theory, its models are more complicated, and from a logical point of view it is a more powerful and complex language. In [Marshall and Chuaqui, 1991] it is argued that the set-theoretical sentences which are preserved under isomorphisms are of fundamental im-

portance, and it is shown that these are the sentences which are equivalent to sentences of type theory.

A brief explanation of the labeling system used in this book may be helpful. The sections in Chapter 1 (for example) are labeled §10, §11, ..., §16. Of course, an alternative labeling would be §1.0, §1.1, ..., §1.6, but the periods are quite redundant once one understands the labeling system. These tags should be regarded as labels, not numbers, although they do have a natural ordering. Similarly, the theorems in §11 (for example) are labeled 1100, 1101, 1102, ..., rather than 1.1.00, 1.1.01, 1.1.02, ... Exercises are labeled in a similar way, but their labels start with an X. Thus, the exercises for section 11 are labeled X1100, X1101, X1102, ... Labeling starts with 0 rather than 1 because 0 is the initial ordinal. (Chapter 0 is the Introduction.)

The reader should examine the table of contents in parallel with the discussion of the individual chapters below. Much that is in this book is simply what one would expect in an introductory text on mathematical logic, so the discussion will concentrate on features one might not expect.

Chapter 1 introduces the student to an axiomatic system $\mathcal{P}$ of propositional calculus and later to more general considerations about propositional calculus. In §14 a convenient two-dimensional notation is introduced to represent wffs in negation normal form. This enables one to readily comprehend the structure of such wffs, find disjunctive and conjunctive normal forms by inspection, and check whether the wffs are contradictory by examining the "vertical paths" through them.

Chapter 2 contains what everyone ought to know about first-order logic. Students can learn a great deal about how to construct proofs by doing the exercises at the end of §21. The discussion of prenex normal form in §22 shows students how to pull out the quantifiers of a wff all at once as well as how to bring them out past each connective one step at a time. This discussion is facilitated by the use of the notation $\exists\!\!\!/$ for the ambiguous quantifier.

In §25 Gödel's Completeness Theorem is proved by an elegant generalization of Henkin's method due to Smullyan. The notion of consistency in Henkin's proof is replaced by abstract consistency, and with very little extra effort a proof is obtained for Smullyan's Unifying Principle, from which readily follow Gödel's Completeness Theorem in §25, Gentzen's Cut-Elimination Theorem in §31, the completeness of the method of semantic tableaux in §32, the completeness of a method for refuting universal sentences in §34, and the Craig–Lyndon Interpolation Theorem in §41. In their first encounter with the proof of Gödel's Completeness Theorem, students may need a less abstract approach than that of Smullyan's Unifying Prin-

ciple, so as an alternative in §25A an elegant and direct simplified proof of the Completeness Theorem is given. It actually parallels the abstract approach very closely and provides a good introduction to the abstract approach. The Löwenheim–Skolem Theorem and the Compactness Theorem are derived in the usual way from the Completeness Theorem. It is shown how the Compactness Theorem can be applied to extend the Four Color Theorem of graph theory from finite to infinite graphs and to prove the existence of nonstandard models for number theory.

Chapter 3 is concerned with the logical principles underlying various techniques for proving or refuting sentences of first-order logic, and provides an introduction to the fundamental ideas used in various approaches to automated theorem proving.

The chapter starts in §30 with a system of natural deduction which is essentially a summary of the most useful derived rules of inference that were obtained in §21. Methods of proving existential formulas are discussed. A cut-free system of logic and semantic tableaux are then introduced in §31 and §32 as systems in which one can search more systematically for a proof (or refutation). When one constructs a semantic tableau, one often notices that it would be nice to postpone deciding how to instantiate universal quantifiers by just instantiating them with variables for which substitutions could be made later. This is facilitated when one eliminates existential quantifiers to obtain a universal sentence by Skolemization, which is the topic of §33. It is noted that several methods of Skolemization can be used.

§34 introduces a method for refuting universal sentences which avoids the branching on disjunctions of semantic tableaux and uses the cut rule to reduce formulas to the contradictory empty disjunction. This method has many elements in common with the resolution method [Robinson, 1965] of automated theorem-proving. Students are often successful at establishing theorems by this method even though they were unable to find proofs for them in natural deduction format.

The proof methods just discussed all yield highly redundant proof structures in which various subformulas may occur many times. §35 introduces a version of Herbrand's Theorem which enables one to establish that a wff is contradictory simply by displaying a suitable compound instance of it.

One of the basic processes of theorem-proving in first- or higher-order logic is that of substituting terms for variables (or instantiating quantified variables) in such a way that certain expressions become the same. §36 provides a description of Robinson's Unification Algorithm for first-order logic and a proof of its correctness. Since substitutions have been defined (in §10) to be functions of a certain sort which map formulas to formulas,

certain facts about substitutions (such as the associativity of composition) can be used without special justification.

Chapter 4 discusses a few additional topics in first-order logic. In §40 the topic of Duality is introduced with a parable about two scholars who argue about an ancient text on logic because of a fundamental ambiguity in the notations for truth and falsehood. Of course, much of this material could appropriately be discussed immediately after §14.

Chapter 5 introduces a system $\mathcal{Q}_0$ of typed λ-calculus which is essentially the system introduced by Alonzo Church in [Church, 1940], except that (following [Henkin, 1963]) equality relations rather than quantifiers and propositional connectives are primitive. Type theory is first introduced in §50 with a discussion of a rather traditional formulation $\mathcal{F}^\omega$ of type theory in which propositional connectives and quantifiers are primitive. It is shown that equality can be defined in such a system in two quite natural ways. It is then shown that this system can be simplified to obtain naive axiomatic set theory, which is inconsistent because Russell's paradox can be derived in it. Thus, one needs some device such as type distinctions or restrictions on the Comprehension Axiom to obtain a consistent formalization of naive set theory. The discussion turns to finding as simple, natural, and expressive a formulation of type theory as possible. (The choice of equality as the primitive logical notion is influenced not only by the desire for simplicity but also by the desire for a natural semantics, as discussed at the end of [Andrews, 1972].) This leads to an exposition of the basic ideas underlying $\mathcal{Q}_0$. While these ideas usually come to be regarded as very natural, they often seem novel at this stage, and need to be absorbed thoroughly before one proceeds to the next section.

By the end of §52 the student should be ready to prove theorems in $\mathcal{Q}_0$, so notations for some simple but basic mathematical ideas are introduced, and in the exercises the student is asked to give formal proofs of some simple mathematical theorems about sets and functions. An exercise in §53 asks for a proof of a concise type-theoretic formulation of Cantor's Theorem.

Henkin's Completeness Theorem is proved for $\mathcal{Q}_0$ in §55. Skolem's paradox (which was first discussed in §25) is discussed again in the context of type theory and resolved with the aid of the important distinction between standard and nonstandard models. It is shown that theories which have infinite models must have nonstandard models.

Chapter 6 is intended to make it clear that mathematics really can be formalized within the system $\mathcal{Q}_0^\infty$ which is the result of adding an axiom of infinity to $\mathcal{Q}_0$. It is shown how cardinal numbers can be defined, Peano's Postulates can be derived, and recursive functions can be represented very

elegantly in $\mathcal{Q}_0^\infty$. Proofs of theorems of $\mathcal{Q}_0^\infty$ are presented in sufficient detail so that students should have no difficulty providing formal justifications for each step. Naturally, each teacher will decide how (or whether) to treat these proofs in class. It is good experience for students to present some proofs, or parts of proofs, in class. By the end of this chapter students should have a firm grasp of the connection between abbreviated formal proofs and the informal proofs seen in other mathematics courses, and should therefore be much more confident in dealing with such proofs.

Chapter 7 presents the classical incompleteness, undecidability and undefinability results for $\mathcal{Q}_0^\infty$. In §70 it is shown that the numerical functions representable in any consistent recursively axiomatized extension of $\mathcal{Q}_0^\infty$ are precisely the recursive functions, and this leads to an argument for Church's Thesis. In §71 Gödel's First and Second Incompleteness Theorems are presented for $\mathcal{Q}_0^\infty$, along with Löb's Theorem about the sentence which says "I am provable". A number of consequences of Gödel's Second Incompleteness Theorem are discussed. In §72 the Gödel–Rosser Incompleteness Theorem is presented for $\mathcal{Q}_0^\infty$, and it is shown how this implies that there is no recursively axiomatized extension of $\mathcal{Q}_0$ whose theorems are precisely the wffs valid in all standard models. §73 is concerned with the unsolvability of the decision problems for $\mathcal{Q}_0$ and $\mathcal{Q}_0^\infty$, and the undefinability of truth. §74 is a brief epilogue reflecting on the elusiveness of truth.

The exercises at the end of each section generally provide opportunities for using material from that section. However, students need to learn how to decide for themselves what techniques to use to solve problems. Therefore, at the end of the book there is a collection of Supplementary Exercises which are not explicitly tied to any particular section.

Some exercises (see §21 and §52, for example) involve applying rules of inference to prove theorems of the formal system being discussed. A computer program called ETPS (which is reviewed in [Goldson and Reeves, 1993] and [Goldson *et al.*, 1993]) has been developed to facilitate work on such exercises. The student using ETPS issues commands to apply rules of inference in specified ways, and the computer handles the details of writing the appropriate lines of the proof and checking that the rules can be used in this way. Proofs, and the active lines of the proof, are displayed using the special symbols of logic in proof windows which are automatically updated as the proof is constructed. The program thus allows students to concentrate on the essential logical problems underlying the proofs, and it gives them immediate feedback for both correct and incorrect actions. Experience shows that ETPS enables students to construct rigorous proofs of more difficult theorems than they would otherwise find tractable.

ETPS permits students to work forwards, backwards, or in a combination of these modes, and provides facilities for rearranging proofs, deleting parts of proofs, displaying only those parts of proofs under active consideration, saving incomplete proofs, and printing proofs on paper. A convenient formula editor permits the student to extract needed formulas which occur anywhere in the proof, and build new formulas from them. Teachers who set up ETPS for use by a class can take advantage of its facilities for keeping records of completed exercises and processing information for grades.

Information about obtaining ETPS without cost can be obtained from http://gtps.math.cmu.edu/tps.html; alternatively, connect to the web page http://www.cmu.edu/ for Carnegie Mellon University or http://www.cs.cmu.edu/afs/cs/project/pal/www/pal.html for CMU's Pure and Applied Logic Program and from there find the author's web page.

I have used versions of this book for a course in mathematical logic at Carnegie Mellon University for about thirty years. Experience has shown that students are best prepared for the course if they have had at least one rigorous mathematics course or a course in philosophy or computer science that has prepared them to appreciate the importance of understanding the nature of deductive reasoning and the art of proving theorems. I have found that if one covers the material rather thoroughly in class, it is difficult to cover all of Chapters 1 – 4 in the first semester, so I normally cover Chapter 1, most of Chapter 2 (using §25A and omitting §24 and most of §26), and also §30, §33, and §34. Naturally, in a course primarily composed of graduate students, or students with more previous exposure to logic, one could move faster, ask students to read some material outside class, and cover all of Chapters 1 – 4 in one semester. In the second semester I normally cover Chapters 5 – 7 fairly completely.

A few comments about notation may be helpful. "U is a subset of V" is written as "$U \subseteq V$", and "U is a proper subset of V" as "$U \subset V$". The composition of functions f and g is written "$f \circ g$". "iff" is an abbreviation for "if and only if". Ends of proofs are marked with the sign ■.

I wish to thank the many people who contributed directly or indirectly to the development of this book. Alonzo Church, John Kemeny, Raymond Smullyan, and Abraham Robinson taught me logic. It will be obvious to all who have read them that this book has been deeply influenced by [Church, 1940] and [Church, 1956]. Numerous students at Carnegie Mellon helped shape the book with their questions, comments, and interests. Special thanks go to Ferna Hartman for her heroic work typing the difficult manuscript.

Bibliography

[Andrews *et al.*, 1984] Peter B. Andrews, Dale A. Miller, Eve Longini Cohen, and Frank Pfenning. Automating Higher-Order Logic. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years*, Contemporary Mathematics series, vol. 29, pages 169–192. American Mathematical Society, 1984.

[Andrews *et al.*, 1996] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16:321–353, 1996.

[Andrews, 1972] Peter B. Andrews. General Models and Extensionality. *Journal of Symbolic Logic*, 37:395–397, 1972.

[Church, 1940] Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Church, 1956] Alonzo Church. *Introduction to Mathematical Logic*. Princeton University Press, Princeton, N.J., 1956.

[Goldson and Reeves, 1993] Doug Goldson and Steve Reeves. Using Programs to Teach Logic to Computer Scientists. *Notices of the American Mathematical Society*, 40:143–148, 1993.

[Goldson *et al.*, 1993] Douglas Goldson, Steve Reeves, and Richard Bornat. A Review of Several Programs for the Teaching of Logic. *The Computer Journal*, 36:373–386, 1993.

[Gordon, 1986] Mike Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177. North-Holland, 1986.

[Hanna and Daeche, 1985] F. K. Hanna and N. Daeche. Specification and Verification using Higher-Order Logic. In Koomen and Moto-oka, editors, *Computer Hardware Description Languages and their Applications*, pages 418–433. North Holland, 1985.

[Hanna and Daeche, 1986] F. K. Hanna and N. Daeche. Specification and Verification Using Higher-Order Logic: A Case Study. In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 179–213. North-Holland, 1986.

[Henkin, 1963] Leon Henkin. A Theory of Propositional Types. *Fundamenta Mathematicae*, 52:323–344, 1963.

[Huet, 1975] Gérard P. Huet. A Unification Algorithm for Typed λ-Calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Jensen and Pietrzykowski, 1976] D.C. Jensen and T. Pietrzykowski. Mechanizing ω-Order Type Theory Through Unification. *Theoretical Computer Science*, 3:123–171, 1976.

[Marshall and Chuaqui, 1991] M. Victoria Marshall and Rolando Chuaqui. Sentences of type theory: the only sentences preserved under isomorphisms. *Journal of Symbolic Logic*, 56:932–948, 1991.

[Robinson, 1965] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12:23–41, 1965.

http://www.springer.com/978-1-4020-0763-7

An Introduction to Mathematical Logic and Type Theory
To Truth Through Proof
Andrews, P.B.
2002, XVIII, 390 p., Hardcover
ISBN: 978-1-4020-0763-7