Mathematical Logic and Computation

Jeremy Avigad

Contents

Preface

In the phrase *mathematical logic*, the word “mathematical” is ambiguous. It can be taken to specify the methods used, so that the phrase refers to the mathematical study of the principles of reasoning. It can be taken to demarcate the type of reasoning considered, so that the phrase refers to the study of specifically mathematical reasoning. Or it can be taken to indicate the purpose, so that the phrase refers to the study of logic with an eye toward mathematical applications.

In the title of this book, the word “mathematical” is intended in the first two senses but not the third. In other words, mathematical logic is viewed here as a mathematical study of the methods of mathematical reasoning. The subject is interesting in its own right, and it has mathematical applications. But it also has applications in computer science, for example, to the verification of hardware and software and to the mechanization of mathematical reasoning. It can inform the philosophy of mathematics as well, by providing idealized models of what it means to do mathematics.

One thing that distinguishes logic as a discipline is its focus on language. The subject starts with formal expressions that are supposed to model the informal language we use to define objects, state claims, and prove them. At that point, two distinct perspectives emerge. From a *semantic* perspective, the formal expressions are used to say things about abstract mathematical objects and structures. They can be used to characterize classes of structures like groups, rings, and fields; to characterize particular structures, like the Euclidean plane or the real numbers; or to describe relationships within a particular structure. From that point of view, mathematical logic is the science of reference, definability, and truth, clarifying the semantic notions that determine the relationship between mathematical language and the mathematical structures it describes.

This book adopts a more *syntactic* perspective, in which the primary objects of interest are the expressions themselves. From that point of view, formal languages are used to reason and calculate, and what we care about are the rules that govern their proper use. We will use formal systems to understand patterns of mathematical inference and the structure of mathematical definitions and proofs, and we will be interested in the things that we can do with these syntactic representations. We won't shy away from the use of semantic methods, but our goal is to use semantics to illuminate syntax rather than the other way around.

There are a number of reasons why a syntactic approach is valuable. The mathematical theory of syntax is independently interesting and informative. A focus on syntactic objects is also more closely aligned with computer science, since these objects can be represented as data and acted on by algorithms. Finally, there are philosophical benefits: because a general theory of finite strings of symbols is all that is needed to work with expressions, a syntactic perspective provides a means of studying mathematical reasoning—including the use of infinitary objects and methods—without importing strong mathematical presuppositions at the outset.

Another notable feature of this book is its focus on computation. On the one hand, we expect mathematics to give us a broad conceptual understanding. At the empirical borders of the subject, this serves to organize and explain our scientific observations, but our desire for understanding is not limited to empirical phenomena. On the other hand, we also expect mathematics to tell us how to calculate trajectories and probabilities so that we can make better predictions and decisions, and act rationally toward securing our pragmatic goals. There is a tension between conceptual understanding and calculation: computation is important, but we often see further, and reason more efficiently, by suppressing computational detail.

The tension is partially captured by the logician's distinction between *classical* logic, on the one hand, and *intuitionistic* or *constructive* logic on the other. Classical logic is meant to support a style of reasoning that supports abstraction and idealization, whereas intuitionistic logic is more directly suited to computational interpretation. Mathematics today is resolutely classical, but mathematics without calculation is almost a contradiction in terms. And while the end goal of computer science is practical computation, abstraction is essential to designing and reasoning about computational systems. Here we will study both classical and constructive logic, with an eye toward understanding the relationships between the two.

The connections between logic and computation run deep: we can compute with formal expressions, we can reason formally about computation, and we can extract computational content from formal derivations. The style of presentation I have adopted here runs the risk of being judged too mathematical by computer scientists and too computationally minded for pure mathematicians, but I have aimed to strike a balance and, hopefully, bring the two communities closer together.

Audience The material here should be accessible at an advanced undergraduate or introductory graduate level. I have tried to keep the presentation self-contained, but the emphasis is on proving things about logical systems rather than illustrating and motivating their use. Readers who have had a prior introduction to logic will be able to navigate the material here more quickly and comfortably.

Notation Most of the notation in this book is conventional, but challenges inevitably arise when juxtaposing material from fields where conventions differ. The most striking instance of such a challenge is the use of *binders* like quantifiers and lambda abstraction: mathematical logicians generally take such operations to bind tightly while computer scientists typically give them the widest scope possible. Another mismatch arises with respect to function application, since theoretical computer scientists often write $f\,x$ instead of $f(x)$. So, where a mathematical logician would write

$$\exists x\, A(x) \to \exists x\, (A(x) \wedge \forall y\, (R(y, x) \to \neg A(y))),$$

a computer scientist might write

$$(\exists x.\ A\,x) \to \exists x.\ A\,x \wedge \forall y.\ R\,y\,x \to \neg A\,y.$$

As a compromise, this book uses the mathematician's notation for symbolic logic but adopts the computer scientist's conventions for computational systems like the simply typed lambda calculus. The differences are most pronounced in Chapters 14 and 17, where logic and type theory come together.

Teaching The material in this book can be used to teach a number of different courses, and the dependencies between topics are mild. Chapters 1–7 provide a thorough introduction to the syntax and semantics of first-order logic. Classically minded mathematicians can easily tune out anything to do with intuitionistic logic, while computer scientists and other interested parties can spend more time with it. Ignoring intuitionistic logic might leave time for the cut elimination theorem and its applications, while skipping cut elimination might leave time to dabble in algebraic semantics.

For students already familiar with propositional and first-order logic, Chapters 8–14 provide a fairly self-contained presentation of formal arithmetic, computability, and the incompleteness theorems. A course on computability and incompleteness can start with Chapter 8, skip Section 8.5, continue to Chapter 11, refer back to Sections 10.2 and 10.5 for background on arithmetic definability, skip Sections 11.8 and 11.9, and then move on to Chapter 12.

A course on proof theory can focus on the cut elimination theorem and its applications (Chapters 6 and 7) and the simply typed lambda calculus (Chapter 13). Chapter 14 can serve as the focus of a course on computational interpretations of arithmetic, building on material in Chapters 8–11 and 13; for that purpose, Sections 8.5, 9.5, 10.6–10.7, and 11.6–11.9 can be omitted. Similarly, Chapter 16 can serve as the focus for an introduction to subsystems of second-order arithmetic and reverse mathematics, supported by a quick tour of Chapters 8–11 and Sections 15.4 and 15.5. Sections 11.8 and 11.9, in particular, were written with Chapter 16 in mind.

Most of the material after Chapter 7 is not strongly tied to any particular deductive system. Natural deduction is sometimes used by default, but that choice is not essential to the exposition.

Acknowledgments I have had the privilege of learning logic with colleagues, mentors, and countless students over the years. Among those to whom I am indebted for comments on drafts of this book are Seulkee Baek, Mario Carneiro, Lucas Clark, Ulrich Kohlenbach, Gustavo Lacerda, Fernando Langlois, Steffen Lempp, Paulo Oliva, Jason Rute, Alex Smith, and Henry Towsner. This list is woefully incomplete. I owe a special thanks to Jasmin Blanchette, Madeleine Harlow, and Stephen Mackereth for exceptionally close readings and detailed corrections. The derivations in this book have been typeset in LaTeX by Samuel Buss's *Bussproofs* package.

The notes at the end of each chapter implicitly acknowledge another debt. I myself learned the topics presented here from a number of excellent sources, including Shoenfield's *Mathematical Logic*, *The Handbook of Mathematical Logic*, Chang and Keisler's *Model Theory*, Troelstra and Van Dalen's *Constructive Mathematics*, Troelstra and Schwichtenberg's *Basic Proof Theory*, Hájek and Pudlák's *Metamathematics of First-Order Arithmetic*, Soare's *Recursively Enumerable Sets and Degrees*, Girard's *Proofs and Types*, Hindley and Seldin's *Lambda-Calculus and Combinators*, Troelstra's epic *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, and Simpson's *Subsystems of Second-Order Arithmetic*. I have drawn extensively on all of these sources and many others. I hope that my presentation does the material justice and helps keep it accessible to the next generation of logicians.

I am indebted to numerous colleagues, including Steve Awodey and Wilfried Sieg, for sharing their insights and expertise. I am especially indebted to Jack Silver and Solomon Feferman, who introduced me to logic and set me on my way. Finally, this book is dedicated to my wife, Elinor, and to my daughters, Rebecca, Jordana, and Ariella. They are the ones who make it all worthwhile.