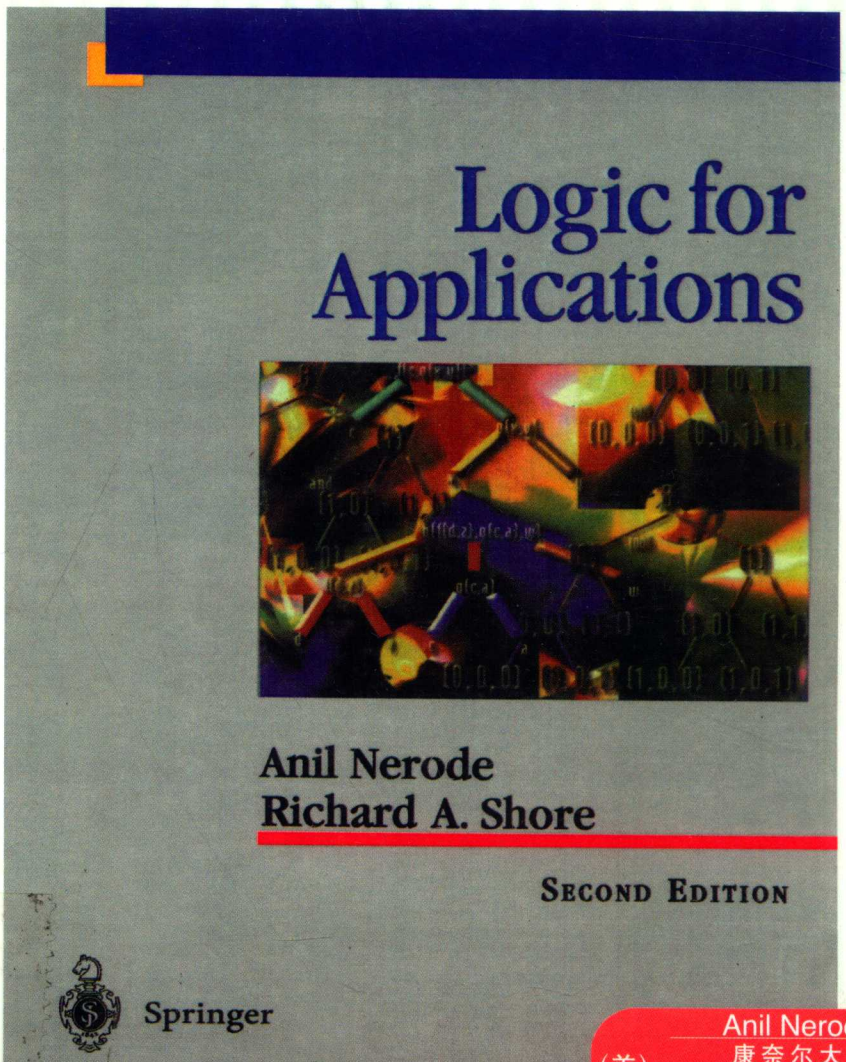
HZ BOOKS
经 典 原 版 书 库

应用逻辑

（英文版 · 第2版）



(美) Anil Nerode 康奈尔大学 Richard A. Shore 康奈尔大学 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

应用逻辑

（英文版 · 第2版）

Logic for Applications

(Second Edition)

(美) Anil Nerode 康奈尔大学
Richard A. Shore 康奈尔大学 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2006-3886

图书在版编目（CIP）数据

应用逻辑（英文版·第2版）/（美）尼罗德（Nerode, A.）等著. -北京：机械工业出版社，2006.9

（经典原版书库）

书名原文：Logic for Applications, Second Edition

ISBN 7-111-19772-0

Ⅰ.应… Ⅱ.尼… Ⅲ.数理逻辑-英文 Ⅳ.O141

中国版本图书馆CIP数据核字（2006）第096589号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

三河市明辉印装有限公司 印刷 · 新华书店北京发行所发行

2006年9月第1版第1次印刷

170mm × 242mm · 29.25印张

定价：49.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

Preface

In writing this book, our goal was to produce a text suitable for a first course in mathematical logic more attuned than the traditional textbooks to the recent dramatic growth in the applications of logic to computer science. Thus, our choice of topics has been heavily influenced by such applications. Of course, we cover the basic traditional topics: syntax, semantics, soundness, completeness and compactness as well as a few more advanced results such as the theorems of Skolem–Löwenheim and Herbrand. Much of our book, however, deals with other less traditional topics. Resolution theorem proving plays a major role in our treatment of logic especially in its application to Logic Programming and PROLOG. We deal extensively with the mathematical foundations of all three of these subjects. In addition, we include two chapters on nonclassical logics — modal and intuitionistic — that are becoming increasingly important in computer science. We develop the basic material on the syntax and semantics (via Kripke frames) for each of these logics. In both cases, our approach to formal proofs, soundness and completeness uses modifications of the same tableau method introduced for classical logic. We indicate how it can easily be adapted to various other special types of modal logics. A number of more advanced topics (including nonmonotonic logic) are also briefly introduced both in the nonclassical logic chapters and in the material on Logic Programming and PROLOG.

The intended audience for this text consists of upper level undergraduate and beginning graduate students of mathematics or computer science. We assume a basic background in abstract reasoning as would be provided by any beginning course in algebra or theoretical computer science as well as the usual familiarity with informal mathematical notation and argument as would be used in any such course. Basic set-theoretic terminology for orderings and trees is given in the opening section I.1.1. We also supply a systematic treatment of elementary set theory in Chapter VI. This material can be used as a reference as needed or covered as part of the course at either its beginning or end.

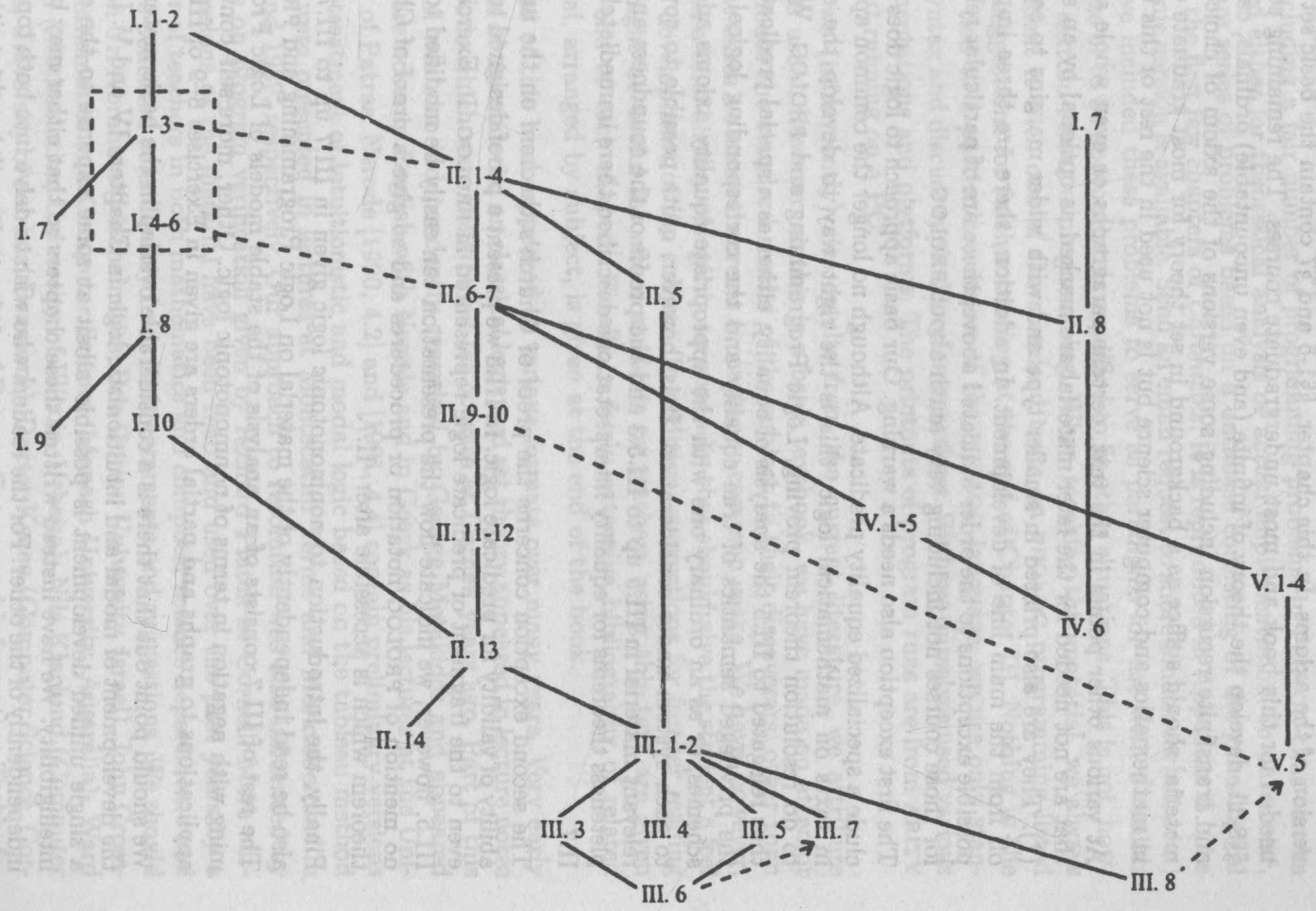
If taught as a course for advanced undergraduates, essentially all the material in Chapters I–III, together with a reasonable amount of programming in PROLOG, can be covered in one semester with three hours of lectures a week. When teaching it in this way, we have had (and recommend) an additional weekly section

devoted to homework problems and programming instruction. Alternatively, the material on resolution theorem proving and Logic Programming can be replaced by the chapters on modal and intuitionistic logic to get a rather different course. For two quarters, one can simply add on one of the nonclassical logics to the first suggested semester course. We have deliberately made these two chapters entirely independent of one another so as to afford a choice. There is, however, much similarity in the developments and, if both are covered, the corresponding sections of the second chapter can be covered more quickly. At the graduate level, essentially the whole book can be covered in a semester.

The text develops propositional logic in its entirety before proceeding to predicate logic. However, depending on the background of the class and the predilections of the instructor, it is possible to combine the treatments. Indeed, for a graduate course or with students with some previous exposure to propositional logic and truth tables, this may well be better. To follow such a path, begin with I.1–2 and then move on to II.1–4. If planning to introduce PROLOG and develop the foundations of Logic Programming, II.5 can be used to explain the syntax and semantics of PROLOG. To start the students on actual programming, an informal introduction to PROLOG implementation and programming can be continued in the recitation section (parts of I.10 are relevant here). Of course, the theoretical underpinnings must wait for the formal development of resolution and unification. With or without the Logic Programming, it is now possible (based on the tableau approach) to prove the soundness and completeness theorems for predicate logic in II.6–7 and to continue through the proof of Herbrand's theorem in II.10. A Hilbert–style proof system is presented without proving any results in I.7 and II.8.

The resolution-style proof system on which PROLOG is based is more intimately tied to its development in the propositional case, as the basic theorems are proven by using Herbrand's theorem to reduce predicate logic results to corresponding ones of propositional logic. Thus, it is necessary, if covering this material, to do the propositional case first. The sections needed for the treatment of PROLOG are I.8, I.10 and II.11–13. The refinements of resolution considered in I.9 are strictly optional. While linear resolution (II.14) is highly relevant to PROLOG, the completeness theorem is one of the most difficult in the text. We have therefore provided in III.1 an alternate approach to the corresponding results for Horn logic and PROLOG that requires only the basic definitions of linear resolution (I.14.1–3).

A diagram of the logical dependencies between sections of the text (except for Chapter VI) is given on the facing page. Unless indicated otherwise by an arrow, the order of dependency runs right to left. Dotted lines (as from the propositional to predicate sections for classical logic) indicate relevance but not strict logical dependence. We should note that the very first section, I.1, simply collects definitions and facts about orderings and trees that are needed at various points in the text. This material can be covered at the beginning or inserted as needed. In particular, König's lemma, I.1.4, is not needed until the treatment of complete


I. 1-2
I. 3
I. 7
I. 4-6
I. 8
I. 9
I. 10
II. 1-4
II. 6-7
II. 5
II. 9-10
II. 11-12
II. 13
II. 14
III. 1-2
III. 3
III. 4
III. 5
III. 7
III. 6
III. 8
I. 7
II. 8
IV. 1-5
IV. 6
V. 1-4
V. 5

systematic tableaux in I.4. Similarly, the material on set–theoretic procedures in Chapter VI (new in this edition) can be covered at any point. Only one section (§6) assumes a familiarity with predicate logic and that is simply to list a formal version of the axioms. In this chapter, §§1-5 and §7 contain the basic set theory needed for this book and most undergraduate courses. The remaining sections, §§8-11, develop the theory of infinite (and even uncountable) ordinals, cardinals and transfinite recursion including some versions of the axiom of choice. This material should suffice as a background in set theory for most graduate courses in mathematics and computer science but is not used in the rest of this book.

At various other points in the text, certain paragraphs or even whole sections that are not needed for the later material are marked as optional by an asterisk (*). They are also printed in smaller type and with wider margins to set them off from the main line of development. In addition, there are three important possible exceptions to the order indicated above; these are of particular relevance for those courses not including very much about PROLOG.

The first exception also needs a warning. Our basic approach to logic does not include a specialized equality predicate. Although no longer the common approach in texts on mathematical logic, this is the right way to develop the subject to do resolution theorem proving, Logic Programming and PROLOG. We have thus relegated to III.5 the analysis of equality, either as a special predicate with the privileged semantics of true equality and the corresponding logical axiom schemes, or as an ordinary one with the appropriate equality axioms added on to each system under consideration. It is, however, quite possible to cover the relevant material in III.5 up to III.5.3 and the proofs of the soundness and completeness theorems for equality interpretations described there immediately after II.7.

The second exception concerns the proof of Church's theorem on the undecidability of validity for predicate logic. In III.8 we present a proof designed to apply even to the fragment of predicate logic represented in PROLOG. In Exercise 3 of III.5, however, we indicate how the presentation can easily be modified to make no mention of PROLOG notation or procedures and so give a proof of Church's theorem which is accessible after II.7.

Finally, the introduction to nonmonotonic logic given in III.7 up to III.7.6 can also be read independently of the material on Logic Programming and PROLOG. The rest of III.7 consists of an analysis of the stable models of Logic Programming with negation in terms of nonmonotonic logic. Other, more self–contained, applications to graphs and partial orders are given in Exercises 8–9 of III.7.

We should point out that there is a considerable overlap in the basic material for the development of modal and intuitionistic logic in Chapters IV and V. Indeed, a single unified development is possible albeit at some expense to the ease of intelligibility. We have instead written these chapters so that either may be read independently of the other. For the readers who wish to delve into both topics, we supply a comparative guide to basic notions of classical, modal and intuitionistic

logic in V.6. We try there to point out the similarities and differences between these logics.

We have included a variety of problems at the end of almost every section of the text including a fair number of programming problems in PROLOG that can be assigned either for theoretical analysis or actual implementation. In particular, there are series of problems based on a database consisting of the genealogical lists in the first few chapters of *Chronicles*. It is reproduced in Appendix B. (An electronic version is available. Send an email request to shore@math.cornell.edu.) We have included these problems to serve as paradigms for use with a similar database or to be appropriately modified to fit other situations. This is not, however, a text on PROLOG programming. When teaching this material, we always supplement it with one of the standard texts on PROLOG programming listed in the suggestions for further reading at the end of Chapter III. Nothing in the text is tied to a particular version of the PROLOG language; we use only standard syntax and discuss typical implementations. We have ourselves used various implementations and platforms. The printouts of program runs are from ARITY PROLOG running on a PC.

When we (infrequently) cite results from the current literature, we attribute them as usual. However, as this is a basic textbook, we have made no attempt to attribute the standard results of the subject to their discoverers, other than when at times we name theorems according to common usage. We have, however, supplied a brief history of logic in an appendix that should give the student a feel for the development of the subject. In addition, suggestions for further reading that might be useful for either students or teachers using this text are given at the end of each chapter. Finally, a fairly extensive bibliography of related material, arranged by subject, is given at the end of the book.

Portions of this book appeared in various formats over many years. Very early versions of the material on classical logic and set theory appeared in lecture notes by Nerode which were distributed for courses at Cornell years ago. Part of this material was also independently reworked by George Metakides and appeared as lecture notes in English with Nerode and in Greek as his lectures at the University of Patras. Nerode [1990, 4.2] and [1991, 4.4] contain preliminary versions of our treatment of intuitionistic and modal logic based on the tableau method which were presented in lectures at Montecatini Terme and Marktoberdorf in 1988 and 1989, respectively. Our approach to resolution was also influenced by courses on program verification given by Richard Platek at Cornell. More current versions of the material have been read and used over the past years by a number of teachers in both mathematics and computer science departments and we have benefited considerably from their comments and suggestions. We should mention Uri Abraham (Mathematics, Ben Gurion University, Israel), John Crossley (Mathematics and Computer Science, Monash University, Australia), Ward Henson (Mathematics, University of Illinois, Urbana), Nils Klarlund (Computer Science, University of Aarhus, Denmark), Dexter Kozen (Computer Science, Cornell),George Metakides (University of Patras, Greece and Information Technolo-

gies Research, EEC) and Jacob Plotkin (Mathematics, Michigan State University). Perry Smith (Math. Reviews) suggested Theorem I.4.11 of this edition and its proof. Bakhadyr Khussainov pointed out an error in the previous edition in the proof of Theorem V.2.20. Warren Goldfarb (Philosophy, Harvard) helped us avoid a number of pitfalls in the historical appendix. Particularly extensive (and highly beneficial) comments were received from Wiktor Marek (Computer Science, University of Kentucky), George Odifreddi (Computer Science, University of Turin, Italy) and Robert Soare (Mathematics and Computer Science, University of Chicago) who used several versions of the text in their courses. We also owe a debt to our graduate students who have served as assistants for our logic courses over the past few years and have made many corrections and suggestions: Jennifer Davoren, Steven Kautz, James Lipton, Sherry Marcus and Duminda Wijesekera.

We gratefully acknowledge the financial support over the past few years of the NSF under grants DMS–8601048, DMS–8902797, DMS-9204308, DMS-9503503; the ARO under grants DAAG29–85–C–0018 and DAAL03–91–C–0027 through ACSyAM at the Mathematical Sciences Institute of Cornell University; and IBM for an equipment grant through Project Ezra at Cornell University. We would also like to thank Arletta Havlik and Graeme Bailey for their help with the TEXing of the original version of the text for the previous edition as well as Geraldine Brady, Jennifer Davoren, Nathaniel Miller, Robert Milnikel, George Odifreddi and David Solomon for their help in proofreading.

Finally, in appreciation of their continuing support, we dedicate this book to our wives, Sally and Naomi.

Cornell University
Ithaca, NY
June 1996

Anil Nerode
Richard A. Shore

Contents

Introduction

In 1920 logic was mostly a philosopher's garden. There were also a few mathematicians there, cultivating the logical roots of the mathematical tree. Today, Recursion Theory, Set Theory, Model Theory and Proof Theory, logic's major subdisciplines, have become full–fledged branches of mathematics. Since the 1970s, the winds of change have been blowing new seeds into the logic garden from computer science, AI and linguistics. These winds have also uncovered a new topography with many prominences and depths, fertile soil for new logical subjects. These days, if you survey international meetings in computer science and linguistics, you will find that the language of mathematical logic is a lingua franca, that methods of mathematical logic are ubiquitous and that understanding new logics and finding feasible algorithms for implementing their inference procedures play a central role in many disciplines. The emerging areas with an important logic component include imperative, declarative and functional programming; verification of programs; interactive, concurrent, distributed, fault tolerant and real time computing; knowledge–based systems; deductive databases; and VLSI design. Various types of logic are now also playing key roles in the modeling of reasoning in special fields from law to medicine.

These applications have widened the horizons of logical research to encompass problems and ideas that were not even considered when logic was motivated only by questions from mathematics and philosophy. Applied logic is now as much a reality as is applied mathematics, with a similarly broad, overlapping but somewhat different area of application. This situation has arisen because of the needs for automated inference in critical, real time and large database information processing applications throughout business, government, science and technology. Mathematical logic, coupled with some of its applications, should be as easily available to college and university students as is applied mathematics. It may well be as important to the future of many previously qualitative disciplines as ordinary applied mathematics has been to the traditionally quantitative ones.

This book is a rigorous elementary introduction to classical predicate logic emphasizing that deduction is a form of computation. We cover the standard topics of soundness, completeness and compactness: our proof methods produce only

valid results, all valid sentences are provable and, if a fact is a logical consequence of an infinite set of axioms, it is actually a consequence of finitely many of them. The need for soundness seems obvious but, as we see in our discussion of PROLOG, even this requirement of simple correctness is often sacrificed on the altar of efficiency in actual implementations. Completeness, on the other hand, is a remarkable result connecting proofs and validity. We can prescribe an effective proof procedure that precisely captures the semantics of first order logic. A valid sentence, i.e., one true for every interpretation of the relations used to state it, always has a proof in a particular formal system and there is an algorithm to find such a proof. Compactness also has surprising applications that deduce results about infinite structures from results about finite ones. To cite just one example, it implies that every planar map is colorable with four colors as every finite planar map is so colorable. We also prove that validity is undecidable: no single algorithm can decide if any given sentence is valid. Thus, although we can, using a particular algorithm, search for a proof of a given sentence φ and be assured of finding one if φ is valid, we cannot know in general whether we are searching in vain.

Our treatment begins in Chapter I with the syntax and semantics of classical propositional logic, that is, the logic of compound sentences formed with connectives such as "and", "or", "if" and "not" but without consideration of the quantifiers "for all" and "there exists". We present a traditional approach to syntax in terms of strings of symbols as well as one based on tree structures. As trees have become basic objects in many computer science areas, the latter approach may well be more accessible (or at least familiar) to many students. Either approach can be adopted. We then introduce the semantic tableau proof method developed by Beth (*Foundations of Mathematics* [1959, 3.2]) and Smullyan (*First Order Logic* [1968, 3.2]) for propositional logic. We have found over the years that the tableaux method is the easiest for students to learn, use and remember. This method seeks to find a proof of a sentence φ by discovering that a systematic search for a counterexample to φ fails in a finite amount of time. The procedure brings out the unadorned reasons for completeness by directly analyzing the subformulas of the formula φ for which a proof is being attempted. It presents the systematic search as a tree–constructing algorithm. The goal of the algorithm is to produce a finite tree beginning with "φ is false" with a contradiction on every branch. Such a tree shows that every analysis of "φ is false" leads to a contradiction. We call this a tableau proof of φ. Employing a systematic search for tableau proofs, we prove the soundness, completeness and compactness theorems.

We then develop the resolution method of theorem proving due to J. A. Robinson [1965, 5.7]. This method has played a crucial role in the development of automated reasoning and theorem proving. After again establishing soundness and completeness, we specialize this method to Horn clauses to develop the mathematical foundations of Logic Programming and PROLOG (still at the propositional level). Logic Programming is a general abstract approach to programming as logical deduction in a restricted setting. PROLOG is a type of programming

language designed to implement the idea that computations are deductions.

In Chapter II we introduce the rest of predicate logic (functions and relations; variables and quantifiers) with explanations of its syntax and semantics. We present a tableau style proof system for predicate logic and prove its soundness and completeness. Our approach naturally leads to Herbrand's theorem which, in a certain sense, reduces predicate logic to propositional logic. Then, following Robinson, we add to resolution the pattern–matching algorithm, called unification, which is originally due to Herbrand. This produces Robinson's system of deduction for predicate logic; it was the first complete redesign of logical inference for the purpose of mechanization of inference on digital computers. It is really better carried out by machines than by hand. Robinson's work made automation of reasoning on digital computers a major area of research. Many of his ideas and much of his terminology have persisted to the present day.

Chapter III is devoted to the specialization of resolution to Horn clauses, a special class of predicate logic formulas that are the domain of Logic Programming and PROLOG. The predicate version of Logic Programming has applications to expert systems, intelligent databases and AI among many others. Logic Programming has a very active research community and has become a separate discipline. In addition to restricting its attention to a limited class of formulas, Logic Programming and PROLOG make various changes in proof procedures to attain computational efficiency. We cover the mathematical foundations of Horn clause logic and then of PROLOG: syntax, semantics, soundness and completeness. We also touch on proofs of termination for PROLOG programs. As an example of current trends, we give an introductory account of the so–called "general logic programs". We present views of implementation and semantics for negation in this setting in terms of both negation as failure and stable models. This area is still in considerable flux. It is one example of the larger evolving subject of nonmonotonic reasoning. Unlike the classical situation, in nonmonotonic logic the addition of new premises may force the withdrawal of conclusions deduced from the previous ones. We include a brief introduction to this area in III.7. We are not programmers, however, and do not attempt to really cover PROLOG programming beyond what is needed to illustrate the underlying logical and mathematical ideas. (References to basic books on PROLOG programming are included in the bibliography.) We do, however, deal with theoretical computability by PROLOG programs as our route to undecidability.

Standard proofs of undecidability for a theory come down to showing how to represent each effectively computable function by a logical formula so that computing the values of the function amounts to deducing instances of that formula. The noncomputability of specific functions such as the halting problem (deciding if a given program halts on a given input) are then translated into the impossibility of deciding the provability of given formulas. In this way, we prove the undecidability of PROLOG and of Horn clause logic (and so *a fortiori* of all of predicate logic) by showing that Horn clause programs and even standard implementations of PROLOG compute all effectively computable functions. As a

definition of an algorithm for an effective computation, we use the model of computation given by programs on register machines. Thus, we simulate each register machine program for computing a recursive function by a PROLOG program computing a coded version of that same function. As it is known that all other models of computation can be simulated by register machines, this suffices to get the desired results on computability and undecidability.

In Chapters IV and V we turn to some nonclassical logics which are becoming increasingly important in understanding and modeling computation and in verifying programs. "Nonclassical" has a technical meaning: the truth of a composite sentence may not depend solely on the truth of its parts and, indeed, even the truth of simple statements may depend on context, time, beliefs, etc. Although this attitude is not the traditional one in mathematics, it reflects many real life situations as well as many important problems in computer science. The truth of an implication often has temporal components. Usually, sentences are evaluated within some context. If our knowledge or beliefs change, so may our evaluation of the truth of some sentence. The analysis of programs depends on the states of knowledge of the computer over time, on what may happen and on what must happen. We touch briefly on one form of such logic (nonmonotonic logic in which later information may invalidate earlier conclusions) in Chapter III. The last two chapters are devoted to a systematic study of two such logics: modal and intuitionistic.

Intuitionism incorporates a constructive view of mathematics into the underlying logic. We can claim that we have a proof of A or B only if we have a proof of one of them. We can claim to have a proof of "there exists an x with property P" only if we can actually exhibit an object c and a proof that c has property P. Modal logic attempts to capture notions of necessity and possibility to serve as a basis for the analyses of systems with temporal, dynamic or belief-based components. We describe the semantics of both of these logics in terms of Kripke frames. These are sets of classical models together with a partial ordering or some other relation on the models; it is this relation that embodies the nonclassical aspects of Kripke semantics. We then formulate tableau systems that generalize the classical ones and faithfully reflect the semantics expressed by Kripke frames. Once again, soundness and completeness play a central role in our exposition. The two logics are presented independently but a comparative guide is supplied in V.6.

For good or ill, the philosophical tenets of intuitionism play no role here nor do the philosophers' analyses of time and necessity. Rather, we explain Kripke frames as a way of modeling the notion of a consequence of partial information and modal operators as simply expressing relations among sets of models. This explanation fits the prospective use of intuitionistic logic, as Scott has suggested, as a language for Scott domains and information systems, or for Horn clause logic which is a subtler use of both classical and intuitionistic logic. It also fits the applications of modal logic to program analysis and verification, as initiated by

Hoare with dynamic logic and continued by many in the field.

We have included in this edition a new chapter (Chapter VI) on the elements of set theory. This material can be used as a reference for standard set–theoretic notations and concepts such as functions, relations, orderings, sequences and the like. It is, however, also a self–contained introduction to axiomatic set theory. In §1-6 of this chapter, we present the axioms for set theory and develop enough of elementary set theory to formalize basic number theory including, for example, the uniqueness of the natural numbers up to isomorphism as a structure satisfying Peano's famous axioms for successor as well as the existence and uniqueness of functions defined by induction or recursion on the natural numbers. We also present the set basic theoretic notions such as functions, sequences, orders and cardinality up to the countable sets needed in this book. In the rest of the chapter, we establish the principle of transfinite induction and develop the basic theory of ordinals, cardinals (including the uncountable ones and their arithmetic) and some of the usual variants of the axiom of choice. Together, this material should supply a sufficient background in set theory for almost any graduate course of studies in computer science or mathematics. The exposition is independent of the rest of the book and repeats a small amount of material from I.1 and the historical appendix.

Finally, we believe that knowing the historical context in which mathematical logic and its applications have developed is important for a full understanding and appreciation of the subject. Thus, we supply in an appendix a brief historical view of the origins and development of logic from the Greeks to the middle of the twentieth century. Parts of this survey may be fully appreciated only after reading the text (especially the first two chapters) but it can be profitably consulted before, after or while reading the book. It is intended only as a tourist brochure, a guide to the terrain. To supplement this guide, we have included a fairly extensive bibliography of historical references and sources for additional information on many topics in logic, including several not covered in the text. When possible, we have confined our suggestions to historical material, handbooks, surveys and basic texts at a level suitable for a reader who has finished this book. Some newer subjects, however, also require references to the current literature. This bibliography is arranged as several (partially annotated) bibliographies on individual subjects. References are made accordingly. Thus, for example, Thomas [1939, 1.1] refers to the item *Selections Illustrating the History of Greek Mathematics with an English Translation* by Ivor Thomas published in 1939 which is listed in Bibliography 1.1, Sourcebooks for the History of Mathematics. We have also included at the end of each chapter suggestions for further reading which are keyed to these bibliographies.