Logic For Computer Science

Foundations of Automatic Theorem Proving

October 2013

Jean Gallier
University of Pennsylvania
Department of Computer and Information Science
Levine Hall
3330 Walnut Street
Philadelphia, Pa 19104
USA
e-mail: `jean@cis.upenn.edu`

To Anne, my wife,

Mia, Philippe and Sylvie, my children,

and my mother, Simone

Preface (Dover Edition)

Logic in Computer Science, initially published in 1985 has been out of print since 2000. Nevertheless, there has been continued demand for my book, and this prompted me in 2003 to correct all known mistakes and typos and to put a copy available online. I am honored that Dover proposed reprinting a hard copy of my book, and I hope that this will be attractive to readers interested in mathematical sciences and philosophy.

This republication of my book is a slighty revised version of the 1985 edition of my logic book. Many typos and errors have been corrected and the line drawings have been improved. Most mistakes were minor, except for a subtle error in Theorem 4.3.3. Indeed, the second part of the theorem about the complexity of the proof tree T obtained from a resolution refutation D is false: It is not necessarily the case that the number of leaves of T is less than or equal to the number of resolution steps in D. As a consequence, Lemma 4.3.4 is also false.

In this revised edition, I simply removed all statements about the complexity of the conversion of resolution refutations into proof trees. Hopefully, this part of the book is now correct, as well as the rest of it!

Some parts of the book have now aged, in particular, the parts about PROLOG. Also, twenty eight years later, I would prefer to present some of the material in a different order and in a different manner. In particular, I would separate more clearly the material on the resolution method (Chapter 4) from the more proof-theory oriented material. However, I consider this too big a task and this mildly revised version will have to do (at least, for now!).

Ideally, a number of topics should also be covered, for example, some basics of constructive logic, linear logic, temporal logic and model checking. Again, this is too formidable a task for a single individual and I hope that readers enjoy this new version of my book anyway.

It should be noted that this book is of “preTEX vintage.” This means that LaTEX was not available at the time this book was written, which implies that I had to write macros (stolen and adapted from D. Knuth blue TEX book) to do chapter headings, etc. Fortunately, Dover converted my TEX files to LaTEX; many thanks for accomplishing this tedious task. Nevertheless, I am grateful to Knuth for producing TEX. Without it, this book would not be alive.

In retrospect, I realize how much I was inspired by, and thus, how much I owe, Gerhard Gentzen, Jacques Herbrand, Stephen Kleene, Gaisi Takeuti, Raymond Smullyan, Peter Andrews and last but not least, Jean-Yves Girard. You have my deepest respect for your seminal and inspiring work and I thank all of you for what you taught me.

Philadelphia, October 2013 Jean Gallier

Preface (1985 Edition)

This book is intended as an introduction to mathematical logic, with an emphasis on proof theory and procedures for constructing formal proofs of formulae algorithmically.

This book is designed primarily for computer scientists, and more generally, for mathematically inclined readers interested in the formalization of proofs, and the foundations of automatic theorem-proving.

The book is self contained, and the level corresponds to senior undergraduates and first year graduate students. However, there is enough material for at least a two semester course, and some Chapters (Chapters 6,7,9,10) contain material which could form the basis of seminars. It would be helpful, but not indispensable, if the reader has had an undergraduate-level course in set theory and/or modern algebra.

Since the main emphasis of the text is on the study of proof systems and algorithmic methods for constructing proofs, it contains some features rarely found in other texts on logic. Four of these features are:

(1) The use of Gentzen Systems;

(2) A Justification of the Resolution method via a translation from a Gentzen System;

(3) A presentation of SLD-resolution and a presentation of the foundations of PROLOG;

(4) Fast decisions procedures based on congruence closures.

A fruitful way to use this text is to teach PROLOG concurently with the material in the book, and ask the student to implement in PROLOG some of the procedures given in the text, in order to design a simple theorem-prover.

Even though the main emphasis of the book is on the design of procedures for constructing formal proofs, the treatment of the semantics is perfectly rigorous. The following paradigm has been followed: Having defined the syntax of the language, it is shown that the set of well-formed formulae is a freely generated inductive set. This is an important point, which is often glossed over. Then, the concepts of satisfaction and validity are defined by recursion over the freely generated inductive set (using the "unique homomorphic extension theorem", which can be rigorously justified). Finally, the proof theory is developped, and procedures for constructing proofs are given. Particular attention is given to the complexity of such procedures.

In our opinion, the choice of Gentzen systems over other formal systems is pedagogically very advantageous. Gentzen-style rules reflect directly the semantics of the logical connectives, lead naturally to mechanical proof procedures, and have the advantage of having duality "built in". Furthermore, in our opinion, Gentzen systems are more convenient than tableaux systems or natural deduction systems for proof-theoretical investigations (cut-free proofs in particular). In three years of teaching, I have found that Gentzen-like systems were very much appreciated by students.

Another good reason for using a formal system inspired from Gentzen (a sequent calculus), is that the completeness theorem is obtained in a natural and simple way. Furthermore, this approach even yields a program (the *search* procedure) for constructing a proof tree for a valid formula. In fact, in our presentation of the completeness theorem (inspired by Kleene, Kleene 1967), the search for a proof tree is described by a program written in pseudo-PASCAL. We also show how a proof procedure for first-order logic with equality can be developed incrementally, starting with the propositional case.

The contents of the book are now outlined.

Chapter 1 sets the goals of the book.

Chapter 2 has been included in order to make the book as self contained as possible, and it covers the mathematical preliminaries needed in the text. It is recommended to refer to this Chapter only when needed, as opposed to reading it entirely before proceeding to Chapter 3.

Propositional logic is studied in Chapter 3. This includes the syntax and semantics of propositional logic. Gentzen systems are introduced as a method for attempting to falsify a proposition. The completeness theorem is shown as well as some of its consequences (the conjunctive and disjunctive normal forms). By introducing infinite sequents, the extended completeness theorem and the compactness theorem are obtained. An informal exposition of the complexity classes P, NP, and of the concept of NP-completeness is given at the end of the Chapter.

The resolution method for propositional logic is presented in Chapter 4. This Chapter uses a new approach for proving the completeness of resolution. Completeness is obtained by defining a special Gentzen System whose completeness follows easily from the results of Chapter 3, and giving an algorithm for converting proofs in the special Gentzen Systems into resolution refutations. Some complexity issues are also examined.

Chapter 5 is devoted to first-order logic. The syntax and semantics are presented. This includes the notions of first-order languages, structures, and models. Gentzen systems are extended to deal with quantifiers and equality. The concept of a Hintikka set is also introduced. It is shown that every Hintikka set is satisfiable in a model whose domain is (a quotient of) a term algebra. This result, together with a generalization of the *search* procedure, is used to derive the main theorems of first-order logic: completeness, compactness, model existence, Löwenheim-Skolem theorems. One of the main themes in this Chapter is that the *search* procedure is a "Hintikka set builder".

Chapter 6 is devoted to Gentzen's "Cut elimination Theorem" and some of its applications. A simple semantic proof derived from the completeness theorem is given for the Gentzen System LK. An entirely proof-theoretic (constructive) argument is also given for a simpler system $G1^{nnf}$. This proof due to Schwichtenberg has the advantage that it also yields a precise upper bound on the length of cut-free proofs obtained from a proof with cut. This result is then extended to a system with equality.

A constructive proof of Craig's Interpolation Theorem is given, and Beth's Definability Theorem and Robinson's Joint Consistency Theorem are also proved. This Chapter contains more advanced material than the previous Chapters.

Chapter 7 is devoted to Gentzen's "Sharpened Hauptsatz", Herbrand's Theorem, and the Skolem-Herbrand-Gödel Theorem. As Chapter 6, this Chapter contains more advanced material. Gentzen's "Sharpened Hauptsatz" for prenex sequents is proved constructively, using proof transformation techniques. A version of the "Sharpened Hauptsatz" is also proved constructively for sequents consisting of formulae in NNF. To prove this result, a new Gentzen system with quantifier rules applying to certain subformulae is defined. This version of the "Sharpened Hauptsatz" for sequents in NNF appears to be new. Using these results, constructive versions of Herbrand's Theorem are proved, as well as Andrews's version of the Skolem-Herbrand-Gödel Theorem (Andrews 1981). The class of primitive recursive functions and the class of recursive functions are also briefly introduced.

In Chapter 8, the resolution method for first-order logic is presented. A recursive unification algorithm inspired from Robinson's algorithm (Robinson 1965) is presented. Using results from Chapter 4 and the Skolem-Herbrand-Gödel Theorem, the completeness of first-order resolution is shown, using the "lifting technique". Paramodulation is also briefly discussed.

Chapter 9 is devoted to SLD-resolution and the foundations of PROLOG. Using techniques from Chapter 4, the completeness of SLD-resolution is shown, by translating proofs in a certain Gentzen system into SLD-refutations. This approach appears to be new. Logic programs are defined, and a model-theoretic semantics is given. It is shown that SLD-resolution is a sound and complete computational proof procedure for logic programs. Most of this material can only be found in research papers, and should be useful to readers interested in logic programming.

In Chapter 10 (the last Chapter), a brief presentation of many-sorted first-order logic is given. This presentation should be sufficient preparation for readers interested in the definition of abstract data types, or computing with rewrite rules. Finally, an extension of the congruence closure method of Nelson and Oppen (Nelson and Oppen 1980) to the many-sorted case and its application to fast decision procedures for testing the validity of quantifier-free formulae are presented.

This book grew out of a number of class notes written for a graduate course in logic for computer scientists, taught at the University of Pennsylvania (CIS581). The inspiration for writing the book came from Sheila Greibach (my advisor at UCLA) and Ronald Book (my "supervisor" at UCSB, while I was a "Post Doc"), who convinced me that there is no better way to really know a topic than writing about it.

I wish to thank my colleagues Saul Gorn, Dale Miller and Alex Pelin for reading the manuscript very carefully, and for many helpful comments. I also wish to thank my students William Dowling, Tomas Isakowitz, Harry Kaplan, Larry Krablin, Francois Lang, Karl Schimpf, Jeff Stroomer, Stan Raatz and Todd Rockoff for their help in "debugging" the manuscript. This includes reporting of typos, stylistic improvements, additional exercises, and correction of mistakes.

Philadelphia, July 1985 Jean Gallier

How To Use This Book As A Text

This book is written at the level appropriate to senior undergraduate and first year graduate students in computer science, or mathematics. The prerequesites are the equivalent of undergraduate-level courses in either set theory, abstract algebra, or discrete structures. All the mathematical background necessary for the text itself is contained in Chapter 2, and in the Appendix. Some of the most difficult exercises may require deeper knowledge of abstract algebra.

Most instructors will find it convenient to use Chapter 2 on a "call by need" basis, depending on the background of the students. However, to the authors's experience, it is usually desirable to review the material contained in Sections 2.1, 2.2 and 2.3.

To help the instructor make up a course, at the end of this section we give a graph showing the dependence of the Sections and Chapters. This graph only applies to the text itself, but not to the exercises, which may depend on any earlier Sections.

The core of the subject which, in the author's opinion, should be part of any course on logic for computer science, is composed of Sections 3.1, 3.2, 3.3 (excluding 3.3.5), 3.4, 3.5, 5.1, 5.2, 5.3, 5.4, 5.5.

The Sections which are next in priority (as core Sections) are 3.6, 5.6, 6.1, 6.2, 6.3, 7.1, 7.2, 7.3, 7.5.

More advanced topics suitable for seminars are covered in Sections 6.4, 6.5, 6.6, 6.7, 7.4, 7.6 and in Chapter 10.

Sections marked with a star ($*$) give a glimpse of topics only sketched in this book. They can be omitted at first reading.

Some results from Section 2.4 are required in Chapter 5. However, in order to shorten Chapter 2, this material as well the material on many-sorted algebras has been made into an Appendix. Similarly, to be perfectly rigorous, Chapter 8 depends on Section 7.6 (since the Skolem-Herbrand-Gödel Theorem proved in Section 7.6 is used to prove the completeness of resolution). However, if the emphasis of the course is on theorem-proving techniques rather than on foundations, it is possible to proceed directly from Section 5.5 to Chapter 8 after having covered Chapter 4). The instructor may simply quote the Herbrand-Skolem-Gödel Theorem from Section 7.6, without proof.

Hence, depending on the time available and the level of the class, there is flexibility for focusing more on automatic theorem-proving methods, or more on foundations. A one semester course emphasizing theorem-proving techniques may consist of the core, plus Chapter 4, Chapter 8, and possibly part of Chapter 9. A one semester course emphasizing foundations may consist of the core, plus Chapter 6 and Chapter 7.

The ideal situtation is to teach the course in two semesters, with automatic theorem-proving techniques first. The second semester covers the foundations, and finishes with a more complete coverage of Chapter 9, Chapter 10, and possibly some material on decision procedures, or on rewrite rules.

It is also possible to use Chapter 6 and Chapter 7 as the core of a seminar on analytic versus non-analytic proofs.

Problems are usually found at the end of each Section. The problems range from routine to very difficult. Difficult exercises or exercises requiring knowledge of material not covered in the text are marked with a star ($*$). Very difficult exercises are marked with a double star ($**$). A few programming assignments have been included.

Some historical remarks and suggestions for further reading are included at the end of each Chapter. Finally the end of a proof is indicated by the symbol □ (box). The word "iff" is used as an abbreviation for "if and only if".

DEPENDENCY OF SECTIONS

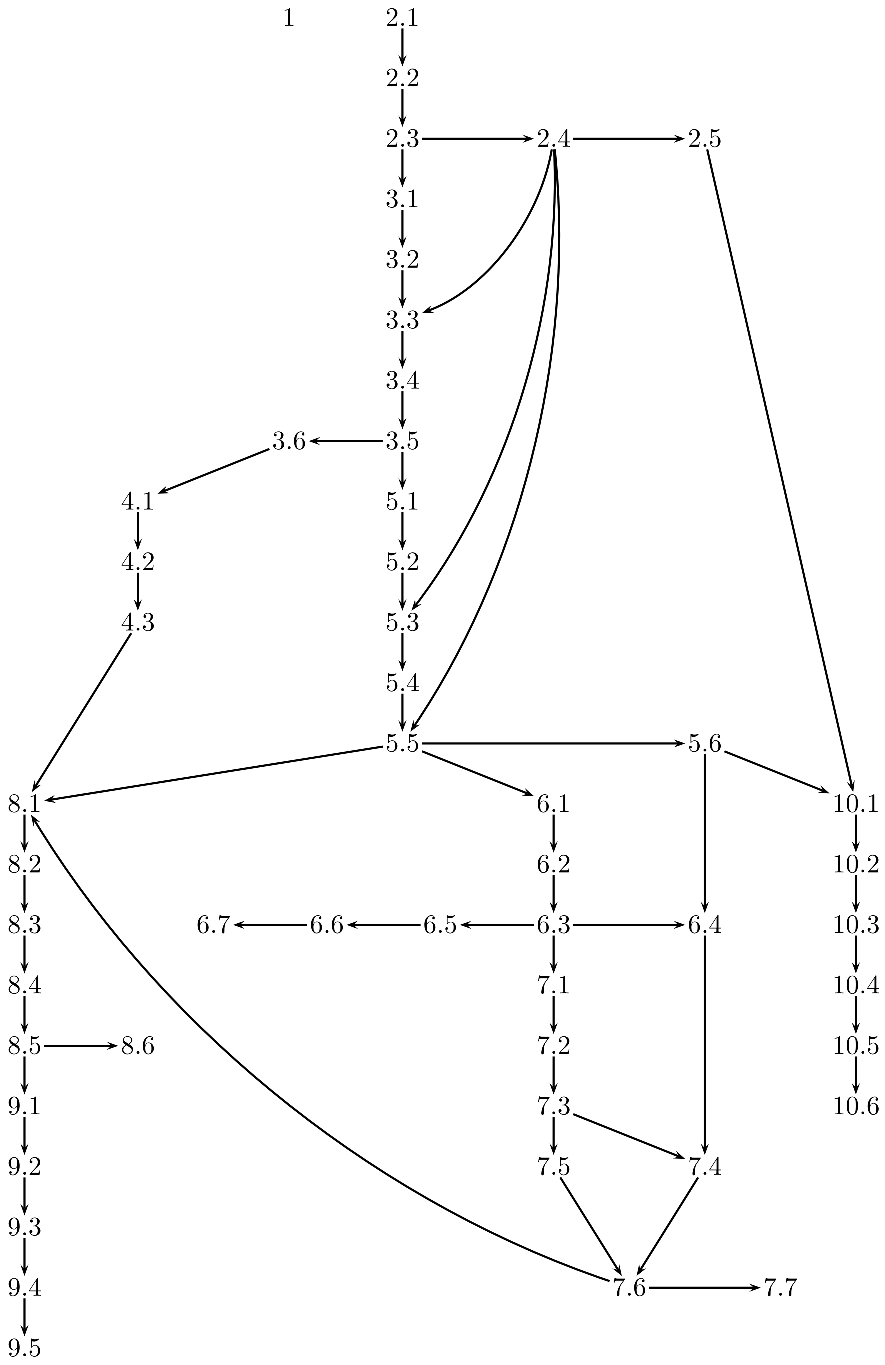

TABLE OF CONTENTS

Chapter 1

Introduction

Logic is concerned mainly with two concepts: *truth* and *provability*. These concepts have been investigated extensively for centuries, by philosophers, linguists, and mathematicians. The purpose of this book is by no means to give a general account of such studies. Instead, the purpose of this book is to focus on a mathematically well defined logical system known as *first-order logic* (and, to some extent, *many-sorted logic*), and prove some basic properties of this system. In particular, we will focus on algorithmic methods for proving theorems (often referred to as *automatic theorem proving*).

Every logical system consists of a *language* used to write statements also called *propositions* or *formulae*. Normally, when one writes a formula, one has some intended *interpretation* of this formula in mind. For example, a formula may assert a true property about the natural numbers, or some property that must be true in a data base. This implies that a formula has a well-defined *meaning* or *semantics*. But how do we define this meaning precisely? In logic, we usually define the meaning of a formula as its *truth value*. A formula can be either true (or valid) or false.

Defining rigorously the notion of truth is actually not as obvious as it appears. We shall present a concept of truth due to Tarski. Roughly speaking, a formula is true if it is satisfied in all possible interpretations. So far, we have used the intuitive meaning of such words as *truth*, *interpretation*, etc. One of the objectives of this book is to define these terms rigorously, for the language of first-order logic (and many-sorted first-order logic). The branch of logic in which abstract structures and the properties true in these structures are studied is known as *model theory*.

Once the concept of truth has been defined rigorously, the next question is to investigate whether it is possible to find methods for deciding in a finite number of steps whether a formula is true (or valid). This is a very difficult task. In fact, by a theorem due to Church, there is no such general method for first-order logic.

However, there is another familiar method for testing whether a formula is true: to give a *proof* of this formula.

Of course, to be of any value, a proof system should be *sound*, which means that every provable formula is true.

We will also define rigorously the notion of proof, and proof system for first-order logic (and many-sorted first-order logic). The branch of logic concerned with the study of proof is known as *proof theory*.

Now, if we have a sound proof system, we know that every provable formula is true. Is the proof system strong enough that it is also possible to prove every true formula (of first-order logic)?

A major theorem of Gödel shows that there are logical proof systems in which every true formula is provable. This is referred to as the *completeness* of the proof system.

To summarize the situation, if one is interested in algorithmic methods for testing whether a formula of first-order logic is valid, there are two logical results of central importance: one positive (Gödel's completeness theorem), the other one negative (Church's undecidability of validity). Roughly speaking, Gödel's completeness theorem asserts that there are logical calculi in which every true formula is provable, and Church's theorem asserts that there is no decision procedure (procedure which always terminates) for deciding whether a formula is true (valid). Hence, any algorithmic procedure for testing whether a formula is true (or equivalently, by Gödel's completeness theorem, provable in a complete system) must run forever when given certain non-true formulae as input.

This book focuses on Gödel's positive result and its applications to automatic theorem proving. We have attempted to present a coherent approach to automatic theorem proving, following a main thread: Gentzen-like sequent calculi. The restriction to the positive result was dictated mostly by the lack of space. Indeed, it should be stressed that Church's negative result is also important, as well as other fundamental negative results due to Gödel. However, the omission of such topics should not be a severe inconvenience to the reader, since there are many texts covering such material (see the notes at the end of Chapter 5).

In spite of the theoretical limitation imposed by Church's result, the goal of automatic theorem proving (for short, atp) is to find *efficient* algorithmic methods for finding proofs of those formulae that are true.

A fairly intuitive method for finding such algorithms is the completeness proof for Gentzen-like sequent calculi. This approach yields a complete procedure (the *search* procedure) for proving valid formulae of first-order logic.

However, the *search* procedure usually requires an enormous amount of space and time and it is not practical. Hence, we will try improve it or find more efficient proof procedures.

For this, we will analyze the structure of proofs carefully. Fundamental results of Gentzen and Herbrand show that if a formula is provable, then it has a proof having a certain form, called a *normal form*.

The existence of such normal forms can be exploited to reduce the size of the search space that needs to be explored in trying to find a proof. Indeed, it is sufficient to look for proofs in normal form.

The existence of normal forms is also fundamental because it reduces the problem of finding a proof of a first-order formula to the problem of finding a proof of a simpler type of formula, called a proposition. Propositions are much simpler than first-order formulae. Indeed, there are algorithms for deciding truth. One of the methods based on this reduction technique is the *resolution method*, which will be investigated in Chapters 4 and 8.

Besides looking for general methods applying to the class of all true (first-order) formulae, it is interesting to consider subclasses for which simpler or more efficient proof procedures exist. Indeed, for certain subclasses there may be decision procedures. This is the case for propositions, and for quantifier-free formulae. Such cases are investigated in Chapters 3 and 10 respectively.

Unfortunately, even in cases in which algorithms exist, another difficulty emerges. A decision procedure may take too much time and space to be practical. For example, even testing whether a proposition is true may be very costly. This will be discussed in Chapter 3.

Automatic theorem proving techniques can be used by computer scientists to axiomatize structures and prove properties of programs working on these structures. Another recent and important role that logic plays in computer science, is its use as a *programming language* and as a *model of computation*. For example, in the programming language PROLOG, programs are specified by sets of assertions. In such a programming language, a computation is in fact a proof, and the output of a program is extracted from the proof. Promoters of such languages claim that since such programs are essentially logical formulae, establishing their correctness is trivial. This is not quite so, because the concept of correctness is relative, and the semantics of a PROLOG program needs to be expressed in a language other than PROLOG. However, using logic as a vehicle for programming is a very interesting idea and should be a selling point for skeptics. This use of logic will be investigated in Chapter 9.