A Problem Course in Mathematical Logic

Version 1.6

Stefan Bilaniuk

Department of Mathematics
Trent University
Peterborough, Ontario
Canada K9J 7B8
E-mail address: sbilaniuk@trentu.ca

1991 *Mathematics Subject Classification.* 03

Key words and phrases. logic, computability, incompleteness

Abstract. This is a text for a problem-oriented course on mathematical logic and computability.

Copyright © 1994-2003 Stefan Bilaniuk.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

This work was typeset with LaTeX, using the $\mathcal{AMS}$-LaTeX and $\mathcal{AMS}$Fonts packages of the American Mathematical Society.

Contents

Preface v

Introduction ix

Part I. Propositional Logic 1

Chapter 1. Language 3

Chapter 2. Truth Assignments 7

Chapter 3. Deductions 11

Chapter 4. Soundness and Completeness 15

Hints for Chapters 1–4 17

Part II. First-Order Logic 21

Chapter 5. Languages 23

Chapter 6. Structures and Models 33

Chapter 7. Deductions 41

Chapter 8. Soundness and Completeness 47

Chapter 9. Applications of Compactness 53

Hints for Chapters 5–9 59

Part III. Computability 65

Chapter 10. Turing Machines 67

Chapter 11. Variations and Simulations 75

Chapter 12. Computable and Non-Computable Functions 81

Chapter 13. Recursive Functions 87

Chapter 14. Characterizing Computability 95

Hints for Chapters 10–14 101

Part IV. Incompleteness 109

Chapter 15. Preliminaries 111

Chapter 16. Coding First-Order Logic 113

Chapter 17. Defining Recursive Functions In Arithmetic 117

Chapter 18. The Incompleteness Theorem 123

Hints for Chapters 15–18 127

Appendices 131

Appendix A. A Little Set Theory 133

Appendix B. The Greek Alphabet 135

Appendix C. Logic Limericks 137

Appendix D. GNU Free Documentation License 139

Appendix. Bibliography 147

Appendix. Index 149

Preface

This book is a free text intended to be the basis for a problem-oriented course(s) in mathematical logic and computability for students with some degree of mathematical sophistication. Parts I and II cover the basics of propositional and first-order logic respectively, Part III covers the basics of computability using Turing machines and recursive functions, and Part IV covers Gödel's Incompleteness Theorems. They can be used in various ways for courses of various lengths and mixes of material. The author typically uses Parts I and II for a one-term course on mathematical logic, Part III for a one-term course on computability, and/or much of Part III together with Part IV for a one-term course on computability and incompleteness.

In keeping with the modified Moore-method, this book supplies definitions, problems, and statements of results, along with some explanations, examples, and hints. The intent is for the students, individually or in groups, to learn the material by solving the problems and proving the results for themselves. Besides constructive criticism, it will probably be necessary for the instructor to supply further hints or direct the students to other sources from time to time. Just how this text is used will, of course, depend on the instructor and students in question. However, it is probably *not* appropriate for a conventional lecture-based course nor for a really large class.

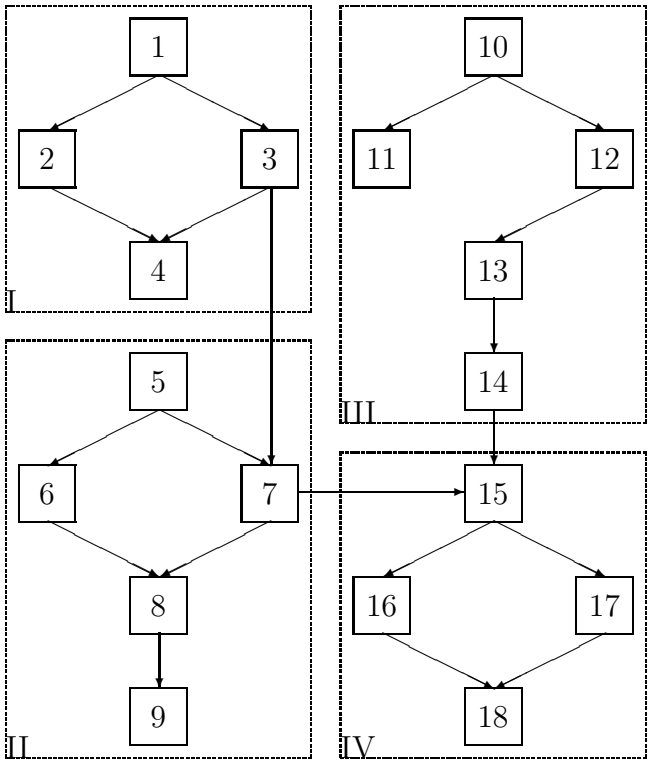
The material presented in this text is somewhat stripped-down. Various concepts and topics that are often covered in introductory mathematical logic and computability courses are given very short shrift or omitted entirely.[1] Instructors might consider having students do projects on additional material if they wish to to cover it.

Prerequisites. The material in this text is largely self-contained, though some knowledge of (very basic) set theory and elementary number theory is assumed at several points. A few problems and examples draw on concepts from other parts of mathematics; students who are

[1]Future versions of both volumes may include more – or less! – material. Feel free to send suggestions, corrections, criticisms, and the like — I'll feel free to ignore them or use them.

not already familiar with these should consult texts in the appropriate subjects for the necessary definitions. What is really needed to get anywhere with all of the material developed here is competence in handling abstraction and proofs, including proofs by induction. The experience provided by a rigorous introductory course in abstract algebra, analysis, or discrete mathematics ought to be sufficient.

Chapter Dependencies. The following diagram indicates how the parts and chapters depend on one another, with the exception of a few isolated problems or subsections.



Acknowledgements. Various people and institutions deserve some credit for this text.

Foremost are all the people who developed the subject, even though almost no attempt has been made to give due credit to those who developed and refined the ideas, results, and proofs mentioned in this work. In mitigation, it would often be difficult to assign credit fairly because many people were involved, frequently having interacted in complicated ways. Those interested in who did what should start by consulting other texts or reference works covering similar material. In

particular, a number of the key papers in the development of modern mathematical logic can be found in [**9**] and [**6**].

Others who should be acknowledged include my teachers and colleagues; my students at Trent University who suffered, suffer, and will suffer through assorted versions of this text; Trent University and the taxpayers of Ontario, who paid my salary; Ohio University, where I spent my sabbatical in 1995–96; all the people and organizations who developed the software and hardware with which this book was prepared. Gregory H. Moore, whose mathematical logic course convinced me that I wanted to do the stuff, deserves particular mention.

Any blame properly accrues to the author.

Availability. The URL of the home page for *A Problem Course In Mathematical Logic*, with links to LaTeX, PostScript, and Portable Document Format (pdf) files of the latest available release is:

`http://euclid.trentu.ca/math/sb/pcml/`

Please note that to typeset the LaTeX source files, you will need the $\mathcal{AMS}$-LaTeX and $\mathcal{AMS}$Fonts packages in addition to LaTeX.

If you have any problems, feel free to contact the author for assistance, preferably by e-mail:

Stefan Bilaniuk
Department of Mathematics
Trent University
Peterborough, Ontario
K9J 7B8
e-mail: `sbilaniuk@trentu.ca`

Conditions. See the *GNU Free Documentation License* in Appendix D for what you can do with this text. The gist is that you are free to copy, distribute, and use it unchanged, but there are some restrictions on what you can do if you wish to make changes. If you wish to use this text in a manner not covered by the *GNU Free Documentation License*, please contact the author.

Author's Opinion. It's not great, but the price is right!

Introduction

What sets mathematics aside from other disciplines is its reliance on proof as the principal technique for determining truth, where science, for example, relies on (carefully analyzed) experience. So what is a proof? Practically speaking, a proof is any reasoned argument accepted as such by other mathematicians.[2] A more precise definition is needed, however, if one wishes to discover what mathematical reasoning can – or cannot – accomplish in principle. This is one of the reasons for studying mathematical logic, which is also pursued for its own sake and in order to find new tools to use in the rest of mathematics and in related fields.

In any case, mathematical logic is concerned with formalizing and analyzing the kinds of reasoning used in the rest of mathematics. The point of mathematical logic is not to try to do mathematics *per se* completely formally — the practical problems involved in doing so are usually such as to make this an exercise in frustration — but to study formal logical systems as mathematical objects in their own right in order to (informally!) prove things about them. For this reason, the formal systems developed in this part and the next are optimized to be easy to prove things about, rather than to be easy to use. Natural deductive systems such as those developed by philosophers to formalize logical reasoning are equally capable in principle and much easier to actually use, but harder to prove things about.

Part of the problem with formalizing mathematical reasoning is the necessity of precisely specifying the language(s) in which it is to be done. The natural languages spoken by humans won't do: they are so complex and continually changing as to be impossible to pin down completely. By contrast, the languages which underly formal logical systems are, like programming languages, rigidly defined but much simpler and less flexible than natural languages. A formal logical system also requires the careful specification of the allowable rules of reasoning,

[2]If you are not a mathematician, gentle reader, you are hereby temporarily promoted.

plus some notion of how to interpret statements in the underlying language and determine their truth. The real fun lies in the relationship between interpretation of statements, truth, and reasoning.

The *de facto* standard for formalizing mathematical systems is first-order logic, and the main thrust of this text is studying it with a view to understanding some of its basic features and limitations. More specifically, Part I of this text is concerned with propositional logic, developed here as a warm-up for the development of first-order logic proper in Part II.

Propositional logic attempts to make precise the relationships that certain connectives like *not*, *and*, *or*, and *if ... then* are used to express in English. While it has uses, propositional logic is not powerful enough to formalize most mathematical discourse. For one thing, it cannot handle the concepts expressed by the quantifiers *all* and *there is*. First-order logic adds these notions to those propositional logic handles, and suffices, in principle, to formalize most mathematical reasoning. The greater flexibility and power of first-order logic makes it a good deal more complicated to work with, both in syntax and semantics. However, a number of results about propositional logic carry over to first-order logic with little change.

Given that first-order logic can be used to formalize most mathematical reasoning it provides a natural context in which to ask whether such reasoning can be automated. This question is the *Entscheidungsproblem*[3]:

ENTSCHEIDUNGSPROBLEM. Given a set Σ of hypotheses and some statement φ, is there an effective method for determining whether or not the hypotheses in Σ suffice to prove φ?

Historically, this question arose out of David Hilbert's scheme to secure the foundations of mathematics by axiomatizing mathematics in first-order logic, showing that the axioms in question do not give rise to any contradictions, and that they suffice to prove or disprove every statement (which is where the Entscheidungsproblem comes in). If the answer to the Entscheidungsproblem were "yes" in general, the effective method(s) in question might put mathematicians out of business... Of course, the statement of the problem begs the question of what "effective method" is supposed to mean.

In the course of trying to find a suitable formalization of the notion of "effective method", mathematicians developed several different

[3] *Entscheidungsproblem* ≡ decision problem.

abstract models of computation in the 1930's, including recursive functions, λ-calculus, Turing machines, and grammars[4]. Although these models are very different from each other in spirit and formal definition, it turned out that they were all essentially equivalent in what they could do. This suggested the (empirical, not mathematical!) principle:

CHURCH'S THESIS. A function is effectively computable in principle in the real world if and only if it is computable by (any) one of the abstract models mentioned above.

Part III explores two of the standard formalizations of the notion of "effective method", namely Turing machines and recursive functions, showing, among other things, that these two formalizations are actually equivalent. Part IV then uses the tools developed in Parts II ands III to answer the Entscheidungsproblem for first-order logic. The answer to the general problem is negative, by the way, though decision procedures do exist for propositional logic, and for some particular first-order languages and sets of hypotheses in these languages.

Prerequisites. In principle, not much is needed by way of prior mathematical knowledge to define and prove the basic facts about propositional logic and computability. Some knowledge of the natural numbers and a little set theory suffices; the former will be assumed and the latter is very briefly summarized in Appendix A. ([**10**] is a good introduction to basic set theory in a style not unlike this book's; [**8**] is a good one in a more conventional mode.) Competence in handling abstraction and proofs, especially proofs by induction, will be needed, however. In principle, the experience provided by a rigorous introductory course in algebra, analysis, or discrete mathematics ought to be sufficient.

Other Sources and Further Reading. [**2**], [**5**], [**7**], [**12**], and [**13**] are texts which go over large parts of the material covered here (and often much more besides), while [**1**] and [**4**] are good references for more advanced material. A number of the key papers in the development of modern mathematical logic and related topics can be found in [**9**] and [**6**]. Entertaining accounts of some related topics may be found in [**11**],

[4]The development of the theory of computation thus actually began before the development of electronic digital computers. In fact, the computers and programming languages we use today owe much to the abstract models of computation which preceded them. For example, the standard von Neumann architecture for digital computers was inspired by Turing machines and the programming language LISP borrows much of its structure from λ-calculus.

[**14**] and[**15**]. Those interested in natural deductive systems might try [**3**], which has a very clean presentation.

Part I

Propositional Logic

CHAPTER 1

Language

Propositional logic (sometimes called sentential or predicate logic) attempts to formalize the reasoning that can be done with connectives like *not*, *and*, *or*, and *if . . . then*. We will define the formal language of propositional logic, $\mathcal{L}_P$, by specifying its symbols and rules for assembling these symbols into the formulas of the language.

DEFINITION 1.1. The *symbols* of $\mathcal{L}_P$ are:

(1) Parentheses: (and).
(2) Connectives: $\neg$ and $\rightarrow$.
(3) Atomic formulas: A_0, A_1, A_2, . . . , A_n, . . .

We still need to specify the ways in which the symbols of $\mathcal{L}_P$ can be put together.

DEFINITION 1.2. The *formulas* of $\mathcal{L}_P$ are those finite sequences or strings of the symbols given in Definition 1.1 which satisfy the following rules:

(1) Every atomic formula is a formula.
(2) If α is a formula, then $(\neg\alpha)$ is a formula.
(3) If α and β are formulas, then $(\alpha \rightarrow \beta)$ is a formula.
(4) No other sequence of symbols is a formula.

We will often use lower-case Greek characters to represent formulas, as we did in the definition above, and upper-case Greek characters to represent sets of formulas.[1] All formulas in Chapters 1–4 will be assumed to be formulas of $\mathcal{L}_P$ unless stated otherwise.

What do these definitions mean? The parentheses are just punctuation: their only purpose is to group other symbols together. (One could get by without them; see Problem 1.6.) $\neg$ and $\rightarrow$ are supposed to represent the connectives *not* and *if . . . then* respectively. The atomic formulas, A_0, A_1, . . . , are meant to represent statements that cannot be broken down any further using our connectives, such as "The moon is made of cheese." Thus, one might translate the the English sentence "If the moon is red, it is not made of cheese" into the formula

[1]The Greek alphabet is given in Appendix B.

$(A_0 \to (\neg A_1))$ of $\mathcal{L}_P$ by using A_0 to represent "The moon is red" and A_1 to represent "The moon is made of cheese." Note that the truth of the formula depends on the interpretation of the atomic sentences which appear in it. Using the interpretations just given of A_0 and A_1, the formula $(A_0 \to (\neg A_1))$ is true, but if we instead use A_0 and A_1 to interpret "My telephone is ringing" and "Someone is calling me", respectively, $(A_0 \to (\neg A_1))$ is false.

Definition 1.2 says that that every atomic formula is a formula and every other formula is built from shorter formulas using the connectives and parentheses in particular ways. For example, A_{1123}, $(A_2 \to (\neg A_0))$, and $(((\neg A_1) \to (A_1 \to A_7)) \to A_7)$ are all formulas, but X_3, (A_5), $)\neg A_{41}$, $A_5 \to A_7$, and $(A_2 \to (\neg A_0)$ are not.

PROBLEM 1.1. *Why are the following* not *formulas of $\mathcal{L}_P$? There might be more than one reason...*

(1) A_{-56}
(2) $(Y \to A)$
(3) $(A_7 \leftarrow A_4)$
(4) $A_7 \to (\neg A_5))$
(5) $(A_8 A_9 \to A_{1043998}$
(6) $(((\neg A_1) \to (A_\ell \to A_7) \to A_7)$

PROBLEM 1.2. *Show that every formula of $\mathcal{L}_P$ has the same number of left parentheses as it has of right parentheses.*

PROBLEM 1.3. *Suppose α is any formula of $\mathcal{L}_P$. Let $\ell(\alpha)$ be the length of α as a sequence of symbols and let $p(\alpha)$ be the number of parentheses (counting both left and right parentheses) in α. What are the minimum and maximum values of $p(\alpha)/\ell(\alpha)$?*

PROBLEM 1.4. *Suppose α is any formula of $\mathcal{L}_P$. Let $s(\alpha)$ be the number of atomic formulas in α (counting repetitions) and let $c(\alpha)$ be the number of occurrences of $\to$ in α. Show that $s(\alpha) = c(\alpha) + 1$.*

PROBLEM 1.5. *What are the possible lengths of formulas of $\mathcal{L}_P$? Prove it.*

PROBLEM 1.6. *Find a way for doing without parentheses or other punctuation symbols in defining a formal language for propositional logic.*

PROPOSITION 1.7. *Show that the set of formulas of $\mathcal{L}_P$ is countable.*

Informal Conventions. At first glance, $\mathcal{L}_P$ may not seem capable of breaking down English sentences with connectives other than *not* and *if ... then*. However, the sense of many other connectives can be

captured by these two by using suitable circumlocutions. We will use the symbols $\wedge$, $\vee$, and $\leftrightarrow$ to represent *and*, *or*,[2] and *if and only if* respectively. Since they are not among the symbols of $\mathcal{L}_P$, we will use them as abbreviations for certain constructions involving only $\neg$ and $\rightarrow$. Namely,

- $(\alpha \wedge \beta)$ is short for $(\neg(\alpha \rightarrow (\neg\beta)))$,
- $(\alpha \vee \beta)$ is short for $((\neg\alpha) \rightarrow \beta)$, and
- $(\alpha \leftrightarrow \beta)$ is short for $((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$.

Interpreting A_0 and A_1 as before, for example, one could translate the English sentence "The moon is red and made of cheese" as $(A_0 \wedge A_1)$. (Of course this is really $(\neg(A_0 \rightarrow (\neg A_1)))$, *i.e.* "It is not the case that if the moon is green, it is not made of cheese.") $\wedge$, $\vee$, and $\leftrightarrow$ were not included among the official symbols of $\mathcal{L}_P$ partly because we can get by without them and partly because leaving them out makes it easier to prove things about $\mathcal{L}_P$.

PROBLEM 1.8. *Take a couple of English sentences with several connectives and translate them into formulas of $\mathcal{L}_P$. You may use $\wedge$, $\vee$, and $\leftrightarrow$ if appropriate.*

PROBLEM 1.9. *Write out $((\alpha \vee \beta) \wedge (\beta \rightarrow \alpha))$ using only $\neg$ and $\rightarrow$.*

For the sake of readability, we will occasionally use some informal conventions that let us get away with writing fewer parentheses:

- We will usually drop the outermost parentheses in a formula, writing $\alpha \rightarrow \beta$ instead of $(\alpha \rightarrow \beta)$ and $\neg\alpha$ instead of $(\neg\alpha)$.
- We will let $\neg$ take precedence over $\rightarrow$ when parentheses are missing, so $\neg\alpha \rightarrow \beta$ is short for $((\neg\alpha) \rightarrow \beta)$, and fit the informal connectives into this scheme by letting the order of precedence be $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$.
- Finally, we will group repetitions of $\rightarrow$, $\vee$, $\wedge$, or $\leftrightarrow$ to the right when parentheses are missing, so $\alpha \rightarrow \beta \rightarrow \gamma$ is short for $(\alpha \rightarrow (\beta \rightarrow \gamma))$.

Just like formulas using $\vee$, $\wedge$, or $\neg$, formulas in which parentheses have been omitted as above are not official formulas of $\mathcal{L}_P$, they are convenient abbreviations for official formulas of $\mathcal{L}_P$. Note that a precedent for the precedence convention can be found in the way that $\cdot$ commonly takes precedence over $+$ in writing arithmetic formulas.

PROBLEM 1.10. *Write out $\neg(\alpha \leftrightarrow \neg\delta) \wedge \beta \rightarrow \neg\alpha \rightarrow \gamma$ first with the missing parentheses included and then as an official formula of $\mathcal{L}_P$.*

[2]We will use *or* inclusively, so that "A or B" is still true if both of A and B are true.

The following notion will be needed later on.

DEFINITION 1.3. Suppose φ is a formula of $\mathcal{L}_P$. The set of *subformulas* of φ, $\mathcal{S}(\varphi)$, is defined as follows.

(1) If φ is an atomic formula, then $\mathcal{S}(\varphi) = \{\varphi\}$.
(2) If φ is $(\neg\alpha)$, then $\mathcal{S}(\varphi) = \mathcal{S}(\alpha) \cup \{(\neg\alpha)\}$.
(3) If φ is $(\alpha \to \beta)$, then $\mathcal{S}(\varphi) = \mathcal{S}(\alpha) \cup \mathcal{S}(\beta) \cup \{(\alpha \to \beta)\}$.

For example, if φ is $(((\neg A_1) \to A_7) \to (A_8 \to A_1))$, then $\mathcal{S}(\varphi)$ includes A_1, A_7, A_8, $(\neg A_1)$, $(A_8 \to A_1)$, $((\neg A_1) \to A_7)$, and $(((\neg A_1) \to A_7) \to (A_8 \to A_1))$ itself.

Note that if you write out a formula with all the official parentheses, then the subformulas are just the parts of the formula enclosed by matching parentheses, plus the atomic formulas. In particular, every formula is a subformula of itself. Note that some subformulas of formulas involving our informal abbreviations $\vee$, $\wedge$, or $\leftrightarrow$ will be most conveniently written using these abbreviations. For example, if ψ is $A_4 \to A_1 \vee A_4$, then

$$\mathcal{S}(\psi) = \{\, A_1,\ A_4,\ (\neg A_1),\ (A_1 \vee A_4),\ (A_4 \to (A_1 \vee A_4)) \,\}.$$

(As an exercise, where did $(\neg A_1)$ come from?)

PROBLEM 1.11. *Find all the subformulas of each of the following formulas.*

(1) $(\neg((\neg A_{56}) \to A_{56}))$
(2) $A_9 \to A_8 \to \neg(A_{78} \to \neg\neg A_0)$
(3) $\neg A_0 \wedge \neg A_1 \leftrightarrow \neg(A_0 \vee A_1)$

Unique Readability. The slightly paranoid — er, truly rigorous — might ask whether Definitions 1.1 and 1.2 actually ensure that the formulas of $\mathcal{L}_P$ are unambiguous, *i.e.* can be read in only one way according to the rules given in Definition 1.2. To actually prove this one must add to Definition 1.1 the requirement that all the symbols of $\mathcal{L}_P$ are distinct and that no symbol is a subsequence of any other symbol. With this addition, one can prove the following:

THEOREM 1.12 (Unique Readability Theorem). *A formula of $\mathcal{L}_P$ must satisfy exactly one of conditions 1–3 in Definition 1.2.*

CHAPTER 2

Truth Assignments

Whether a given formula φ of $\mathcal{L}_P$ is true or false usually depends on how we interpret the atomic formulas which appear in φ. For example, if φ is the atomic formula A_2 and we interpret it as "2+2 = 4", it is true, but if we interpret it as "The moon is made of cheese", it is false. Since we don't want to commit ourselves to a single interpretation — after all, we're really interested in general logical relationships — we will define how any assignment of *truth values* T ("true") and F ("false") to atomic formulas of $\mathcal{L}_P$ can be extended to all other formulas. We will also get a reasonable definition of what it means for a formula of $\mathcal{L}_P$ to follow logically from other formulas.

DEFINITION 2.1. A *truth assignment* is a function v whose domain is the set of all formulas of $\mathcal{L}_P$ and whose range is the set $\{T, F\}$ of truth values, such that:

(1) $v(A_n)$ is defined for every atomic formula A_n.
(2) For any formula α,

$$v(\,(\neg\alpha)\,) = \begin{cases} T & \text{if } v(\alpha) = F \\ F & \text{if } v(\alpha) = T. \end{cases}$$

(3) For any formulas α and β,

$$v(\,(\alpha \to \beta)\,) = \begin{cases} F & \text{if } v(\alpha) = T \text{ and } v(\beta) = F \\ T & \text{otherwise.} \end{cases}$$

Given interpretations of all the atomic formulas of $\mathcal{L}_P$, the corresponding truth assignment would give each atomic formula representing a true statement the value T and every atomic formula representing a false statement the value F. Note that we have not defined how to handle any truth values besides T and F in $\mathcal{L}_P$. Logics with other truth values have uses, but are not relevant in most of mathematics.

For an example of how non-atomic formulas are given truth values on the basis of the truth values given to their components, suppose v is a truth assignment such that $v(A_0) = T$ and $v(A_1) = F$. Then $v(\,((\neg A_1) \to (A_0 \to A_1))\,)$ is determined from $v(\,(\neg A_1)\,)$ and $v(\,(A_0 \to$

$A_1)\,)$ according to clause 3 of Definition 2.1. In turn, $v(\,(\neg A_1)\,)$ is determined from of $v(A_1)$ according to clause 2 and $v(\,(A_0 \to A_1)\,)$ is determined from $v(A_1)$ and $v(A_0)$ according to clause 3. Finally, by clause 1, our truth assignment must be defined for all atomic formulas to begin with; in this case, $v(A_0) = T$ and $v(A_1) = F$. Thus $v(\,(\neg A_1)\,) = T$ and $v(\,(A_0 \to A_1)\,) = F$, so $v(\,((\neg A_1) \to (A_0 \to A_1))\,) = F$.

A convenient way to write out the determination of the truth value of a formula on a given truth assignment is to use a *truth table*: list all the subformulas of the given formula across the top in order of length and then fill in their truth values on the bottom from left to right. Except for the atomic formulas at the extreme left, the truth value of each subformula will depend on the truth values of the subformulas to its left. For the example above, one gets something like:

A_0	A_1	$(\neg A_1)$	$(A_0 \to A_1)$	$(\neg A_1) \to (A_0 \to A_1))$
T	F	T	F	F

PROBLEM 2.1. *Suppose v is a truth assignment such that $v(A_0) = v(A_2) = T$ and $v(A_1) = v(A_3) = F$. Find $v(\alpha)$ if α is:*

(1) $\neg A_2 \to \neg A_3$
(2) $\neg A_2 \to A_3$
(3) $\neg(\neg A_0 \to A_1)$
(4) $A_0 \vee A_1$
(5) $A_0 \wedge A_1$

The use of finite truth tables to determine what truth value a particular truth assignment gives a particular formula is justified by the following proposition, which asserts that only the truth values of the atomic sentences in the formula matter.

PROPOSITION 2.2. *Suppose δ is any formula and u and v are truth assignments such that $u(A_n) = v(A_n)$ for all atomic formulas A_n which occur in δ. Then $u(\delta) = v(\delta)$.*

COROLLARY 2.3. *Suppose u and v are truth assignments such that $u(A_n) = v(A_n)$ for every atomic formula A_n. Then $u = v$,* i.e. *$u(\varphi) = v(\varphi)$ for every formula φ.*

PROPOSITION 2.4. *If α and β are formulas and v is a truth assignment, then:*

(1) *$v(\neg\alpha) = T$ if and only if $v(\alpha) = F$.*
(2) *$v(\alpha \to \beta) = T$ if and only if $v(\beta) = T$ whenever $v(\alpha) = T$;*
(3) *$v(\alpha \wedge \beta) = T$ if and only if $v(\alpha) = T$ and $v(\beta) = T$;*
(4) *$v(\alpha \vee \beta) = T$ if and only if $v(\alpha) = T$ or $v(\beta) = T$; and*
(5) *$v(\alpha \leftrightarrow \beta) = T$ if and only if $v(\alpha) = v(\beta)$.*

Truth tables are often used even when the formula in question is not broken down all the way into atomic formulas. For example, if α and β are any formulas and we know that α is true but β is false, then the truth of $(\alpha \to (\neg\beta))$ can be determined by means of the following table:

α	β	$(\neg\beta)$	$(\alpha \to (\neg\beta))$
T	F	T	T

DEFINITION 2.2. If v is a truth assignment and φ is a formula, we will often say that v *satisfies* φ if $v(\varphi) = T$. Similarly, if Σ is a set of formulas, we will often say that v satisfies Σ if $v(\sigma) = T$ for every $\sigma \in \Sigma$. We will say that φ (respectively, Σ) is *satisfiable* if there is some truth assignment which satisfies it.

DEFINITION 2.3. A formula φ is a *tautology* if it is satisfied by every truth assignment. A formula ψ is a *contradiction* if there is no truth assignment which satisfies it.

For example, $(A_4 \to A_4)$ is a tautology while $(\neg(A_4 \to A_4))$ is a contradiction, and A_4 is a formula which is neither. One can check whether a given formula is a tautology, contradiction, or neither, by grinding out a complete truth table for it, with a separate line for each possible assignment of truth values to the atomic subformulas of the formula. For $A_3 \to (A_4 \to A_3)$ this gives

A_3	A_4	$A_4 \to A_3$	$A_3 \to (A_4 \to A_3)$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

so $A_3 \to (A_4 \to A_3)$ is a tautology. Note that, by Proposition 2.2, we need only consider the possible truth values of the atomic sentences which actually occur in a given formula.

One can often use truth tables to determine whether a given formula is a tautology or a contradiction even when it is not broken down all the way into atomic formulas. For example, if α is any formula, then the table

α	$(\alpha \to \alpha)$	$(\neg(\alpha \to \alpha))$
T	T	F
F	T	F

demonstrates that $(\neg(\alpha \to \alpha))$ is a contradiction, no matter which formula of $\mathcal{L}_P$ α actually is.

PROPOSITION 2.5. *If α is any formula, then $((\neg\alpha) \vee \alpha)$ is a tautology and $((\neg\alpha) \wedge \alpha)$ is a contradiction.*

PROPOSITION 2.6. *A formula β is a tautology if and only if $\neg\beta$ is a contradiction.*

After all this warmup, we are finally in a position to define what it means for one formula to follow logically from other formulas.

DEFINITION 2.4. A set of formulas Σ *implies* a formula φ, written as $\Sigma \models \varphi$, if every truth assignment v which satisfies Σ also satisfies φ. We will often write $\Sigma \not\models \varphi$ if it is not the case that $\Sigma \models \varphi$. In the case where Σ is empty, we will usually write $\models \varphi$ instead of $\emptyset \models \varphi$.

Similarly, if Δ and Γ are sets of formulas, then Δ *implies* Γ, written as $\Delta \models \Gamma$, if every truth assignment v which satisfies Δ also satisfies Γ.

For example, $\{ A_3, (A_3 \to \neg A_7) \} \models \neg A_7$, but $\{ A_8, (A_5 \to A_8) \} \not\models A_5$. (There is a truth assignment which makes A_8 and $A_5 \to A_8$ true, but A_5 false.) Note that a formula φ is a tautology if and only if $\models \varphi$, and a contradiction if and only if $\models (\neg\varphi)$.

PROPOSITION 2.7. *If Γ and Σ are sets of formulas such that $\Gamma \subseteq \Sigma$, then $\Sigma \models \Gamma$.*

PROBLEM 2.8. *How can one check whether or not $\Sigma \models \varphi$ for a formula φ and a finite set of formulas Σ?*

PROPOSITION 2.9. *Suppose Σ is a set of formulas and ψ and ρ are formulas. Then $\Sigma \cup \{\psi\} \models \rho$ if and only if $\Sigma \models \psi \to \rho$.*

PROPOSITION 2.10. *A set of formulas Σ is satisfiable if and only if there is no contradiction χ such that $\Sigma \models \chi$.*

CHAPTER 3

Deductions

In this chapter we develop a way of defining logical implication that does not rely on any notion of truth, but only on manipulating sequences of formulas, namely formal proofs or deductions. (Of course, any way of defining logical implication had better be compatible with that given in Chapter 2.) To define these, we first specify a suitable set of formulas which we can use freely as premisses in deductions.

DEFINITION 3.1. The three *axiom schema* of $\mathcal{L}_P$ are:

A1: $(\alpha \to (\beta \to \alpha))$
A2: $((\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)))$
A3: $(((\neg\beta) \to (\neg\alpha)) \to (((\neg\beta) \to \alpha) \to \beta))$.

Replacing α, β, and γ by particular formulas of $\mathcal{L}_P$ in any one of the schemas A1, A2, or A3 gives an *axiom* of $\mathcal{L}_P$.

For example, $(A_1 \to (A_4 \to A_1))$ is an axiom, being an instance of axiom schema A1, but $(A_9 \to (\neg A_0))$ is not an axiom as it is not the instance of any of the schema. As had better be the case, every axiom is always true:

PROPOSITION 3.1. *Every axiom of $\mathcal{L}_P$ is a tautology.*

Second, we specify our one (and only!) rule of inference.[1]

DEFINITION 3.2 (Modus Ponens). Given the formulas φ and $(\varphi \to \psi)$, one may infer ψ.

We will usually refer to Modus Ponens by its initials, MP. Like any rule of inference worth its salt, MP preserves truth.

PROPOSITION 3.2. *Suppose φ and ψ are formulas. Then $\{\, \varphi, (\varphi \to \psi) \,\} \models \psi$.*

With axioms and a rule of inference in hand, we can execute formal proofs in $\mathcal{L}_P$.

[1]Natural deductive systems, which are usually more convenient to actually execute deductions in than the system being developed here, compensate for having few or no axioms by having many rules of inference.

DEFINITION 3.3. Let Σ be a set of formulas. A *deduction* or *proof* from Σ in $\mathcal{L}_P$ is a finite sequence $\varphi_1\varphi_2\dots\varphi_n$ of formulas such that for each $k \leq n$,

(1) φ_k is an axiom, or
(2) $\varphi_k \in \Sigma$, or
(3) there are $i, j < k$ such that φ_k follows from φ_i and φ_j by MP.

A formula of Σ appearing in the deduction is called a *premiss*. Σ *proves* a formula α, written as $\Sigma \vdash \alpha$, if α is the last formula of a deduction from Σ. We'll usually write $\vdash \alpha$ for $\emptyset \vdash \alpha$, and take $\Sigma \vdash \Delta$ to mean that $\Sigma \vdash \delta$ for every formula $\delta \in \Delta$.

In order to make it easier to verify that an alleged deduction really is one, we will number the formulas in a deduction, write them out in order on separate lines, and give a justification for each formula. Like the additional connectives and conventions for dropping parentheses in Chapter 1, this is not officially a part of the definition of a deduction.

EXAMPLE 3.1. Let us show that $\vdash \varphi \to \varphi$.

(1) $(\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$	A2
(2) $\varphi \to ((\varphi \to \varphi) \to \varphi)$	A1
(3) $(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	1,2 MP
(4) $\varphi \to (\varphi \to \varphi)$	A1
(5) $\varphi \to \varphi$	3,4 MP

Hence $\vdash \varphi \to \varphi$, as desired. Note that indication of the formulas from which formulas 3 and 5 beside the mentions of MP.

EXAMPLE 3.2. Let us show that $\{\, \alpha \to \beta,\, \beta \to \gamma \,\} \vdash \alpha \to \gamma$.

(1) $(\beta \to \gamma) \to (\alpha \to (\beta \to \gamma))$	A1
(2) $\beta \to \gamma$	Premiss
(3) $\alpha \to (\beta \to \gamma)$	1,2 MP
(4) $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	A2
(5) $(\alpha \to \beta) \to (\alpha \to \gamma)$	4,3 MP
(6) $\alpha \to \beta$	Premiss
(7) $\alpha \to \gamma$	5,6 MP

Hence $\{\, \alpha \to \beta,\, \beta \to \gamma \,\} \vdash \alpha \to \gamma$, as desired.

It is frequently convenient to save time and effort by simply referring to a deduction one has already done instead of writing it again as part of another deduction. If you do so, please make sure you appeal only to deductions that have already been carried out.

EXAMPLE 3.3. Let us show that $\vdash (\neg\alpha \to \alpha) \to \alpha$.

(1) $(\neg\alpha \to \neg\alpha) \to ((\neg\alpha \to \alpha) \to \alpha)$	A3

(2) $\neg\alpha \to \neg\alpha$ Example 3.1
(3) $(\neg\alpha \to \alpha) \to \alpha$ 1,2 MP

Hence $\vdash (\neg\alpha \to \alpha) \to \alpha$, as desired. To be completely formal, one would have to insert the deduction given in Example 3.1 (with φ replaced by $\neg\alpha$ throughout) in place of line 2 above and renumber the old line 3.

PROBLEM 3.3. *Show that if α, β, and γ are formulas, then*

(1) $\{\, \alpha \to (\beta \to \gamma),\, \beta \,\} \vdash \alpha \to \gamma$
(2) $\vdash \alpha \vee \neg\alpha$

EXAMPLE 3.4. Let us show that $\vdash \neg\neg\beta \to \beta$.

(1) $(\neg\beta \to \neg\neg\beta) \to ((\neg\beta \to \neg\beta) \to \beta)$ A3
(2) $\neg\neg\beta \to (\neg\beta \to \neg\neg\beta)$ A1
(3) $\neg\neg\beta \to ((\neg\beta \to \neg\beta) \to \beta)$ 1,2 Example 3.2
(4) $\neg\beta \to \neg\beta$ Example 3.1
(5) $\neg\neg\beta \to \beta$ 3,4 Problem 3.3.1

Hence $\vdash \neg\neg\beta \to \beta$, as desired.

Certain general facts are sometimes handy:

PROPOSITION 3.4. *If $\varphi_1\varphi_2\ldots\varphi_n$ is a deduction of $\mathcal{L}_P$, then $\varphi_1\ldots\varphi_\ell$ is also a deduction of $\mathcal{L}_P$ for any ℓ such that $1 \le \ell \le n$.*

PROPOSITION 3.5. *If $\Gamma \vdash \delta$ and $\Gamma \vdash \delta \to \beta$, then $\Gamma \vdash \beta$.*

PROPOSITION 3.6. *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \alpha$, then $\Delta \vdash \alpha$.*

PROPOSITION 3.7. *If $\Gamma \vdash \Delta$ and $\Delta \vdash \sigma$, then $\Gamma \vdash \sigma$.*

The following theorem often lets one take substantial shortcuts when trying to show that certain deductions exist in $\mathcal{L}_P$, even though it doesn't give us the deductions explicitly.

THEOREM 3.8 (Deduction Theorem). *If Σ is any set of formulas and α and β are any formulas, then $\Sigma \vdash \alpha \to \beta$ if and only if $\Sigma \cup \{\alpha\} \vdash \beta$.*

EXAMPLE 3.5. Let us show that $\vdash \varphi \to \varphi$. By the Deduction Theorem it is enough to show that $\{\varphi\} \vdash \varphi$, which is trivial:

(1) φ Premiss

Compare this to the deduction in Example 3.1.

PROBLEM 3.9. *Appealing to previous deductions and the Deduction Theorem if you wish, show that:*

(1) $\{\delta, \neg\delta\} \vdash \gamma$

(2) $\vdash \varphi \to \neg\neg\varphi$
(3) $\vdash (\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$
(4) $\vdash (\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$
(5) $\vdash (\beta \to \neg\alpha) \to (\alpha \to \neg\beta)$
(6) $\vdash (\neg\beta \to \alpha) \to (\neg\alpha \to \beta)$
(7) $\vdash \sigma \to (\sigma \vee \tau)$
(8) $\{\alpha \wedge \beta\} \vdash \beta$
(9) $\{\alpha \wedge \beta\} \vdash \alpha$

CHAPTER 4

Soundness and Completeness

How are deduction and implication related, given that they were defined in completely different ways? We have some evidence that they behave alike; compare, for example, Proposition 2.9 and the Deduction Theorem. It had better be the case that if there is a deduction of a formula φ from a set of premisses Σ, then φ is implied by Σ. (Otherwise, what's the point of defining deductions?) It would also be nice for the converse to hold: whenever φ is implied by Σ, there is a deduction of φ from Σ. (So anything which is true can be proved.) The Soundness and Completeness Theorems say that both ways do hold, so $\Sigma \vdash \varphi$ if and only if $\Sigma \models \varphi$, *i.e.* $\vdash$ and $\models$ are equivalent for propositional logic. One direction is relatively straightforward to prove...

THEOREM 4.1 (Soundness Theorem). *If Δ is a set of formulas and α is a formula such that $\Delta \vdash \alpha$, then $\Delta \models \alpha$.*

...but for the other direction we need some additional concepts.

DEFINITION 4.1. A set of formulas Γ is *inconsistent* if $\Gamma \vdash \neg(\alpha \to \alpha)$ for some formula α, and *consistent* if it is not inconsistent.

For example, $\{A_{41}\}$ is consistent by Proposition 4.2, but it follows from Problem 3.9 that $\{A_{13}, \neg A_{13}\}$ is inconsistent.

PROPOSITION 4.2. *If a set of formulas is satisfiable, then it is consistent.*

PROPOSITION 4.3. *Suppose Δ is an inconsistent set of formulas. Then $\Delta \vdash \psi$ for any formula ψ.*

PROPOSITION 4.4. *Suppose Σ is an inconsistent set of formulas. Then there is a finite subset Δ of Σ such that Δ is inconsistent.*

COROLLARY 4.5. *A set of formulas Γ is consistent if and only if every finite subset of Γ is consistent.*

To obtain the Completeness Theorem requires one more definition.

DEFINITION 4.2. A set of formulas Σ is *maximally consistent* if Σ is consistent but $\Sigma \cup \{\varphi\}$ is inconsistent for any $\varphi \notin \Sigma$.

That is, a set of formulas is maximally consistent if it is consistent, but there is no way to add any other formula to it and keep it consistent.

PROBLEM 4.6. *Suppose v is a truth assignment. Show that $\Sigma = \{ \varphi \mid v(\varphi) = T \}$ is maximally consistent.*

We will need some facts concerning maximally consistent theories.

PROPOSITION 4.7. *If Σ is a maximally consistent set of formulas, φ is a formula, and $\Sigma \vdash \varphi$, then $\varphi \in \Sigma$.*

PROPOSITION 4.8. *Suppose Σ is a maximally consistent set of formulas and φ is a formula. Then $\neg\varphi \in \Sigma$ if and only if $\varphi \notin \Sigma$.*

PROPOSITION 4.9. *Suppose Σ is a maximally consistent set of formulas and φ and ψ are formulas. Then $\varphi \to \psi \in \Sigma$ if and only if $\varphi \notin \Sigma$ or $\psi \in \Sigma$.*

It is important to know that any consistent set of formulas can be expanded to a maximally consistent set.

THEOREM 4.10. *Suppose Γ is a consistent set of formulas. Then there is a maximally consistent set of formulas Σ such that $\Gamma \subseteq \Sigma$.*

Now for the main event!

THEOREM 4.11. *A set of formulas is consistent if and only if it is satisfiable.*

Theorem 4.11 gives the equivalence between $\vdash$ and $\models$ in slightly disguised form.

THEOREM 4.12 (Completeness Theorem). *If Δ is a set of formulas and α is a formula such that $\Delta \models \alpha$, then $\Delta \vdash \alpha$.*

It follows that anything provable from a given set of premisses must be true if the premisses are, and *vice versa*. The fact that $\vdash$ and $\models$ are actually equivalent can be very convenient in situations where one is easier to use than the other. For example, most parts of Problems 3.3 and 3.9 are much easier to do with truth tables instead of deductions, even if one makes use of the Deduction Theorem.

Finally, one more consequence of Theorem 4.11.

THEOREM 4.13 (Compactness Theorem). *A set of formulas Γ is satisfiable if and only if every finite subset of Γ is satisfiable.*

We will not look at any uses of the Compactness Theorem now, but we will consider a few applications of its counterpart for first-order logic in Chapter 9.

Hints for Chapters 1–4

Hints for Chapter 1.

1.1. Symbols not in the language, unbalanced parentheses, lack of connectives...

1.2. The key idea is to exploit the recursive structure of Definition 1.2 and proceed by induction on the length of the formula or on the number of connectives in the formula. As this is an idea that will be needed repeatedly in Parts I, II, and IV, here is a skeleton of the argument in this case:

PROOF. By induction on n, the number of connectives (*i.e.* occurrences of $\neg$ and/or $\rightarrow$) in a formula φ of $\mathcal{L}_P$, we will show that any formula φ must have just as many left parentheses as right parentheses.

Base step: ($n = 0$) If φ is a formula with no connectives, then it must be atomic. (Why?) Since an atomic formula has no parentheses at all, it has just as many left parentheses as right parentheses.

Induction hypothesis: ($n \leq k$) Assume that any formula with $n \leq k$ connectives has just as many left parentheses as right parentheses.

Induction step: ($n = k + 1$) Suppose φ is a formula with $n = k + 1$ connectives. It follows from Definition 1.2 that φ must be either

(1) $(\neg\alpha)$ for some formula α with k connectives or
(2) $(\beta \rightarrow \gamma)$ for some formulas β and γ which have $\leq k$ connectives each.

(Why?) We handle the two cases separately:

(1) By the induction hypothesis, α has just as many left parentheses as right parentheses. Since φ, *i.e.* $(\neg\alpha)$, has one more left parenthesis and one more right parentheses than α, it must have just as many left parentheses as right parentheses as well.
(2) By the induction hypothesis, β and γ each have the same number of left parentheses as right parentheses. Since φ, *i.e.* $(\beta \rightarrow \alpha)$, has one more left parenthesis and one more right parnthesis than β and γ together have, it must have just as many left parntheses as right parentheses as well.

It follows by induction that every formula φ of $\mathcal{L}_P$ has just as many left parentheses as right parentheses.

□

1.3. Compute $p(\alpha)/\ell(\alpha)$ for a number of examples and look for patterns. Getting a minimum value should be pretty easy.

1.4. Proceed by induction on the length of or on the number of connectives in the formula.

1.5. Construct examples of formulas of all the short lengths that you can, and then see how you can make longer formulas out of short ones.

1.6. Hewlett-Packard sells calculators that use such a trick. A similar one is used in Definition 5.2.

1.7. Observe that $\mathcal{L}_P$ has countably many symbols and that every formula is a finite sequence of symbols. The relevant facts from set theory are given in Appendix A.

1.8. Stick several simple statements together with suitable connectives.

1.9. This should be straightforward.

1.10. Ditto.

1.11. To make sure you get all the subformulas, write out the formula in official form with all the parentheses.

1.12. Proceed by induction on the length or number of connectives of the formula.

Hints for Chapter 2.

2.1. Use truth tables.

2.2. Proceed by induction on the length of δ or on the number of connectives in δ.

2.3. Use Proposition 2.2.

2.4. In each case, unwind Definition 2.1 and the definitions of the abbreviations.

2.5. Use truth tables.

2.6. Use Definition 2.3 and Proposition 2.4.

2.7. If a truth assignment satisfies every formula in Σ and every formula in Γ is also in Σ, then...

2.8. Grinding out an appropriate truth table will do the job. Why is it important that Σ be finite here?

2.9. Use Definition 2.4 and Proposition 2.4.

2.10. Use Definitions 2.3 and 2.4. If you have trouble trying to prove one of the two directions directly, try proving its contrapositive instead.

Hints for Chapter 3.

3.1. Truth tables are probably the best way to do this.

3.2. Look up Proposition 2.4.

3.3. There are usually many different deductions with a given conclusion, so you shouldn't take the following hints as gospel.

(1) Use A2 and A1.
(2) Recall what $\vee$ abbreviates.

3.4. You need to check that $\varphi_1 \dots \varphi_\ell$ satisfies the three conditions of Definition 3.3; you know $\varphi_1 \dots \varphi_n$ does.

3.5. Put together a deduction of β from Γ from the deductions of δ and $\delta \to \beta$ from Γ.

3.6. Examine Definition 3.3 carefully.

3.7. The key idea is similar to that for proving Proposition 3.5.

3.8. One direction follows from Proposition 3.5. For the other direction, proceed by induction on the length of the shortest proof of β from $\Sigma \cup \{\alpha\}$.

3.9. Again, don't take these hints as gospel. Try using the Deduction Theorem in each case, plus

(1) A3.
(2) A3 and Problem 3.3.
(3) A3.
(4) A3, Problem 3.3, and Example 3.2.
(5) Some of the above parts and Problem 3.3.
(6) Ditto.
(7) Use the definition of $\vee$ and one of the above parts.
(8) Use the definition of $\wedge$ and one of the above parts.
(9) Aim for $\neg\alpha \to (\alpha \to \neg\beta)$ as an intermediate step.

Hints for Chapter 4.

4.1. Use induction on the length of the deduction and Proposition 3.2.

4.2. Assume, by way of contradiction, that the given set of formulas is inconsistent. Use the Soundness Theorem to show that it can't be satisfiable.

4.3. First show that $\{\neg(\alpha \to \alpha)\} \vdash \psi$.

4.4. Note that deductions are finite sequences of formulas.

4.5. Use Proposition 4.4.

4.6. Use Proposition 4.2, the definition of Σ, and Proposition 2.4.

4.7. Assume, by way of contradiction, that $\varphi \notin \Sigma$. Use Definition 4.2 and the Deduction Theorem to show that Σ must be inconsistent.

4.8. Use Definition 4.2 and Problem 3.9.

4.9. Use Definition 4.2 and Proposition 4.8.

4.10. Use Proposition 1.7 and induction on a list of all the formulas of $\mathcal{L}_P$.

4.11. One direction is just Proposition 4.2. For the other, expand the set of formulas in question to a maximally consistent set of formulas Σ using Theorem 4.10, and define a truth assignment v by setting $v(A_n) = T$ if and only if $A_n \in \Sigma$. Now use induction on the length of φ to show that $\varphi \in \Sigma$ if and only if v satisfies φ.

4.12. Prove the contrapositive using Theorem 4.11.

4.13. Put Corollary 4.5 together with Theorem 4.11.

Part II

First-Order Logic

CHAPTER 5

Languages

As noted in the Introduction, propositional logic has obvious deficiencies as a tool for mathematical reasoning. First-order logic remedies enough of these to be adequate for formalizing most ordinary mathematics. It does have enough in common with propositional logic to let us recycle some of the material in Chapters 1–4.

A few informal words about how first-order languages work are in order. In mathematics one often deals with structures consisting of a set of elements plus various operations on them or relations among them. To cite three common examples, a group is a set of elements plus a binary operation on these elements satisfying certain conditions, a field is a set of elements plus two binary operations on these elements satisfying certain conditions, and a graph is a set of elements plus a binary relation with certain properties. In most such cases, one frequently uses symbols naming the operations or relations in question, symbols for variables which range over the set of elements, symbols for logical connectives such as *not* and *for all*, plus auxiliary symbols such as parentheses, to write formulas which express some fact about the structure in question. For example, if $(G, \cdot)$ is a group, one might express the associative law by writing something like

$$\forall x \, \forall y \, \forall z \; x \cdot (y \cdot z) = (x \cdot y) \cdot z \, ,$$

it being understood that the variables range over the set G of group elements. A formal language to do as much will require some or all of these: symbols for various logical notions and for variables, some for functions or relations, plus auxiliary symbols. It will also be necessary to specify rules for putting the symbols together to make formulas, for interpreting the meaning and determining the truth of these formulas, and for making inferences in deductions.

For a concrete example, consider elementary number theory. The set of elements under discussion is the set of natural numbers $\mathbb{N} = \{ 0, 1, 2, 3, 4, \dots \}$. One might need symbols or names for certain interesting numbers, say 0 and 1; for variables over $\mathbb{N}$ such as n and x; for functions on $\mathbb{N}$, say $\cdot$ and $+$; and for relations, say $=$, $<$, and $|$. In addition, one is likely to need symbols for punctuation, such as (and

); for logical connectives, such as $\neg$ and $\rightarrow$; and for quantifiers, such as $\forall$ ("for all") and $\exists$ ("there exists"). A statement of mathematical English such as "For all n and m, if n divides m, then n is less than or equal to m" can then be written as a cool formula like

$$\forall n \forall m \left(n \mid m \rightarrow (n < m \wedge n = m)\right).$$

The extra power of first-order logic comes at a price: greater complexity. First, there are many first-order languages one might wish to use, practically one for each subject, or even problem, in mathematics.[1] We will set up our definitions and general results, however, to apply to a wide range of them.[2]

As with $\mathcal{L}_P$, our formal language for propositional logic, first-order languages are defined by specifying their symbols and how these may be assembled into formulas.

DEFINITION 5.1. The *symbols* of a first-order language $\mathcal{L}$ include:

(1) Parentheses: (and).
(2) Connectives: $\neg$ and $\rightarrow$.
(3) Quantifier: $\forall$.
(4) Variables: v_0, v_1, v_2, ..., v_n, ...
(5) Equality: $=$.
(6) A (possibly empty) set of *constant* symbols.
(7) For each $k \geq 1$, a (possibly empty) set of *k-place function* symbols.
(8) For each $k \geq 1$, a (possibly empty) set of *k-place relation* (or *predicate*) symbols.

The symbols described in parts 1–5 are the *logical* symbols of $\mathcal{L}$, shared by every first-order language, and the rest are the *non-logical* symbols of $\mathcal{L}$, which usually depend on what the language's intended use.

NOTE. It is possible to define first-order languages without $=$, so $=$ is considered a non-logical symbol by many authors. While such languages have some uses, they are uncommon in ordinary mathematics.

Observe that any first-order language $\mathcal{L}$ has countably many logical symbols. It may have uncountably many symbols if it has uncountably many non-logical symbols. *Unless explicitly stated otherwise, we will*

[1] It is possible to formalize almost all of mathematics in a single first-order language, like that of set theory or category theory. However, trying to actually do most mathematics in such a language is so hard as to be pointless.

[2] Specifically, to countable one-sorted first-order languages with equality.

assume that every first-order language we encounter has only countably many non-logical symbols. Most of the results we will prove actually hold for countable and uncountable first-order languages alike, but some require heavier machinery to prove for uncountable languages.

Just as in $\mathcal{L}_P$, the parentheses are just punctuation while the connectives, $\neg$ and $\rightarrow$, are intended to express *not* and *if . . . then*. However, the rest of the symbols are new and are intended to express ideas that cannot be handled by $\mathcal{L}_P$. The quantifier symbol, $\forall$, is meant to represent *for all*, and is intended to be used with the variable symbols, *e.g.* $\forall v_4$. The constant symbols are meant to be names for particular elements of the structure under discussion. k-place function symbols are meant to name particular functions which map k-tuples of elements of the structure to elements of the structure. k-place relation symbols are intended to name particular k-place relations among elements of the structure.[3] Finally, $=$ is a special binary relation symbol intended to represent equality.

Example 5.1. Since the logical symbols are always the same, first-order languages are usually defined by specifying the non-logical symbols. A formal language for elementary number theory like that unofficially described above, call it $\mathcal{L}_{NT}$, can be defined as follows.

- Constant symbols: 0 and 1
- Two 2-place function symbols: $+$ and $\cdot$
- Two binary relation symbols: $<$ and $|$

Each of these symbols is intended to represent the same thing it does in informal mathematical usage: 0 and 1 are intended to be names for the numbers zero and one, $+$ and $\cdot$ names for the operations of addition and multiplications, and $<$ and $|$ names for the relations "less than" and "divides". (Note that we could, in principle, interpret things completely differently – let 0 represent the number forty-one, $+$ the operation of exponentiation, and so on – or even use the language to talk about a different structure – say the real numbers, $\mathbb{R}$, with 0, 1, $+$, $\cdot$, and $<$ representing what they usually do and, just for fun, $|$ interpreted as "is not equal to". More on this in Chapter 6.) We will usually use the same symbols in our formal languages that we use informally for various common mathematical objects. This convention

[3]Intuitively, a relation or predicate expresses some (possibly arbitrary) relationship among one or more objects. For example, "n is prime" is a 1-place relation on the natural numbers, $<$ is a 2-place or binary relation on the rationals, and $\vec{a} \times (\vec{b} \times \vec{c}) = \vec{0}$ is a 3-place relation on $\mathbb{R}^3$. Formally, a k-place relation on a set X is just a subset of X^k, *i.e.* the collection of sequences of length k of elements of X for which the relation is true.

can occasionally cause confusion if it is not clear whether an expression involving these symbols is supposed to be an expression in a formal language or not.

EXAMPLE 5.2. Here are some other first-order languages. Recall that we need only specify the non-logical symbols in each case and note that some parts of Definitions 5.2 and 5.3 may be irrelevant for a given language if it is missing the appropriate sorts of non-logical symbols.

(1) The language of pure equality, $\mathcal{L}_=$:
   - No non-logical symbols at all.

(2) A language for fields, $\mathcal{L}_F$:
   - Constant symbols: 0, 1
   - 2-place function symbols: $+$, $\cdot$

(3) A language for set theory, $\mathcal{L}_S$:
   - 2-place relation symbol: $\in$

(4) A language for linear orders, $\mathcal{L}_O$:
   - 2-place relation symbol: $<$

(5) Another language for elementary number theory, $\mathcal{L}_N$:
   - Constant symbol: 0
   - 1-place function symbol: S
   - 2-place function symbols: $+$, $\cdot$, E

   Here 0 is intended to represent zero, S the successor function, *i.e.* $S(n) = n + 1$, and E the exponential function, *i.e.* $E(n, m) = n^m$.

(6) A "worst-case" countable language, $\mathcal{L}_1$:
   - Constant symbols: c_1, c_2, c_3, ...
   - For each $k \geq 1$, k-place function symbols: f_1^k, f_2^k, f_3^k, ...
   - For each $k \geq 1$, k-place relation symbols: P_1^k, P_2^k, P_3^k, ...

   This language has no use except as an abstract example.

It remains to specify how to form valid formulas from the symbols of a first-order language $\mathcal{L}$. This will be more complicated than it was for $\mathcal{L}_P$. In fact, we first need to define a type of expression in $\mathcal{L}$ which has no counterpart in propositional logic.

DEFINITION 5.2. The *terms* of a first-order language $\mathcal{L}$ are those finite sequences of symbols of $\mathcal{L}$ which satisfy the following rules:

(1) Every variable symbol v_n is a term.
(2) Every constant symbol c is a term.
(3) If f is a k-place function symbol and $t_1, \ldots, t_k$ are terms, then $ft_1 \ldots t_k$ is also a term.
(4) Nothing else is a term.

That is, a term is an expression which represents some (possibly indeterminate) element of the structure under discussion. For example, in $\mathcal{L}_{NT}$ or $\mathcal{L}_N$, $+v_0v_1$ (informally, $v_0 + v_1$) is a term, though precisely which natural number it represents depends on what values are assigned to the variables v_0 and v_1.

PROBLEM 5.1. *Which of the following are terms of one of the languages defined in Examples 5.1 and 5.2? If so, which of these language(s) are they terms of; if not, why not?*

(1) $\cdot v_2$
(2) $+0 \cdot +v_6 11$
(3) $|1 + v_3 0$
(4) $(< E101 \rightarrow +11)$
(5) $+ + \cdot + 00000$
(6) $f_4^3 f_7^2 c_4 v_9 c_1 v_4$
(7) $\cdot v_5(+1v_8)$
(8) $< v_6 v_2$
(9) $1 + 0$

Note that in languages with no function symbols all terms have length one.

PROBLEM 5.2. *Choose one of the languages defined in Examples 5.1 and 5.2 which has terms of length greater than one and determine the possible lengths of terms of this language.*

PROPOSITION 5.3. *The set of terms of a countable first-order language $\mathcal{L}$ is countable.*

Having defined terms, we can finally define first-order formulas.

DEFINITION 5.3. The *formulas* of a first-order language $\mathcal{L}$ are the finite sequences of the symbols of $\mathcal{L}$ satisfying the following rules:

(1) If P is a k-place relation symbol and $t_1, \ldots, t_k$ are terms, then $Pt_1 \ldots t_k$ is a formula.
(2) If t_1 and t_2 are terms, then $= t_1 t_2$ is a formula.
(3) If α is a formula, then $(\neg\alpha)$ is a formula.
(4) If α and β are formulas, then $(\alpha \rightarrow \beta)$ is a formula.
(5) If φ is a formula and v_n is a variable, then $\forall v_n \varphi$ is a formula.
(6) Nothing else is a formula.

Formulas of form 1 or 2 will often be referred to as the *atomic formulas* of $\mathcal{L}$.

Note that three of the conditions in Definition 5.3 are borrowed directy from propositional logic. As before, we will exploit the way

formulas are built up in making definitions and in proving results by induction on the length of a formula. We will also recycle the use of lower-case Greek characters to refer to formulas and of upper-case Greek characters to refer to sets of formulas.

PROBLEM 5.4. *Which of the following are formulas of one of the languages defined in Examples 5.1 and 5.2? If so, which of these language(s) are they formulas of; if not, why not?*

(1) $= 0 + v_7 \cdot 1 v_3$
(2) $(\neg = v_1 v_1)$
(3) $(|v_2 0 \to \cdot 01)$
(4) $(\neg \forall v_5 (= v_5 v_5))$
(5) $< +01 | v_1 v_3$
(6) $(v_3 = v_3 \to \forall v_5\, v_3 = v_5)$
(7) $\forall v_6 (= v_6 0 \to \forall v_9 (\neg | v_9 v_6))$
(8) $\forall v_8 < +11 v_4$

PROBLEM 5.5. *Show that every formula of a first-order language has the same number of left parentheses as of right parentheses.*

PROBLEM 5.6. *Choose one of the languages defined in Examples 5.1 and 5.2 and determine the possible lengths of formulas of this language.*

PROPOSITION 5.7. *A countable first-order language $\mathcal{L}$ has countably many formulas.*

In practice, devising a formal language intended to deal with a particular (kind of) structure isn't the end of the job: one must also specify axioms in the language that the structure(s) one wishes to study should satisfy. Defining satisfaction is officially done in the next chapter, but it is usually straightforward to unofficially figure out what a formula in the language is supposed to mean.

PROBLEM 5.8. *In each case, write down a formula of the given language expressing the given informal statement.*

(1) *"Addition is associative" in $\mathcal{L}_F$.*
(2) *"There is an empty set" in $\mathcal{L}_S$.*
(3) *"Between any two distinct elements there is a third element" in $\mathcal{L}_O$.*
(4) *"$n^0 = 1$ for every n different from 0" in $\mathcal{L}_N$.*
(5) *"There is only one thing" in $\mathcal{L}_=$.*

PROBLEM 5.9. *Define first-order languages to deal with the following structures and, in each case, an appropriate set of axioms in your language:*

(1) *Groups.*
(2) *Graphs.*
(3) *Vector spaces.*

We will need a few additional concepts and facts about formulas of first-order logic later on. First, what are the subformulas of a formula?

PROBLEM 5.10. *Define the set of subformulas of a formula φ of a first-order language $\mathcal{L}$.*

For example, if φ is

$$(((\neg\forall v_1\,(\neg = v_1 c_7)) \to P_3^2 v_5 v_8) \to \forall v_8 (= v_8 f_5^3 c_0 v_1 v_5 \to P_2^1 v_8))$$

in the language $\mathcal{L}_1$, then the set of subformulas of φ, $\mathcal{S}(\varphi)$, ought to include

- $= v_1 c_7$, $P_3^2 v_5 v_8$, $= v_8 f_5^3 c_0 v_1 v_5$, $P_2^1 v_8$,
- $(\neg = v_1 c_7)$, $(= v_8 f_5^3 c_0 v_1 v_5 \to P_2^1 v_8)$,
- $\forall v_1\,(\neg = v_1 c_7)$, $\forall v_8 (= v_8 f_5^3 c_0 v_1 v_5 \to P_2^1 v_8)$,
- $(\neg\forall v_1\,(\neg = v_1 c_7))$,
- $(\neg\forall v_1\,(\neg = v_1 c_7)) \to P_3^2 v_5 v_8)$, and
- $(((\neg\forall v_1\,(\neg = v_1 c_7)) \to P_3^2 v_5 v_8) \to \forall v_8 (= v_8 f_5^3 c_0 v_1 v_5 \to P_2^1 v_8))$ itself.

Second, we will need a concept that has no counterpart in propositional logic.

DEFINITION 5.4. Suppose x is a variable of a first-order language $\mathcal{L}$. Then x *occurs free* in a formula φ of $\mathcal{L}$ is defined as follows:

(1) If φ is atomic, then x occurs free in φ if and only if x occurs in φ.
(2) If φ is $(\neg\alpha)$, then x occurs free in φ if and only if x occurs free in α.
(3) If φ is $(\beta \to \delta)$, then x occurs free in φ if and only if x occurs free in β or in δ.
(4) If φ is $\forall v_k\,\psi$, then x occurs free in φ if and only if x is different from v_k and x occurs free in ψ.

An occurrence of x in φ which is not free is said to be *bound*. A formula σ of $\mathcal{L}$ in which no variable occurs free is said to be a *sentence*.

Part 4 is the key: it asserts that an occurrence of a variable x is bound instead of free if it is in the "scope" of an occurrence of $\forall x$. For example, v_7 is free in $\forall v_5 = v_5 v_7$, but v_5 is not. Different occurences of a given variable in a formula may be free or bound, depending on where they are; *e.g.* v_6 occurs both free and bound in $\forall v_0\,(= v_0 f_3^1 v_6 \to (\neg\forall v_6\, P_9^1 v_6))$.

PROBLEM 5.11. *Give a precise definition of the scope of a quantifier.*

Note the distinction between sentences and ordinary formulas introduced in the last part of Definition 5.4. As we shall see, sentences are often more tractable and useful theoretically than ordinary formulas.

PROBLEM 5.12. *Which of the formulas you gave in solving Problem 5.8 are sentences?*

Finally, we will eventually need to consider a relationship between first-order languages.

DEFINITION 5.5. A first-order language $\mathcal{L}'$ is an *extension* of a first-order language $\mathcal{L}$, sometimes written as $\mathcal{L} \subseteq \mathcal{L}'$, if every non-logical symbol of $\mathcal{L}$ is a non-logical symbol of the same kind of $\mathcal{L}'$.

For example, every first-order language is an extension of $\mathcal{L}_=$.

PROBLEM 5.13. *Which of the languages given in Example 5.2 are extensions of other languages given in Example 5.2?*

PROPOSITION 5.14. *Suppose $\mathcal{L}$ is a first-order language and $\mathcal{L}'$ is an extension of $\mathcal{L}$. Then every formula φ of $\mathcal{L}$ is a formula of $\mathcal{L}'$.*

Common Conventions. As with propositional logic, we will often use abbreviations and informal conventions to simplify the writing of formulas in first-order languages. In particular, we will use the same additional connectives we used in propositional logic, plus an additional quantifier, $\exists$ ("there exists"):

- $(\alpha \wedge \beta)$ is short for $(\neg(\alpha \to (\neg\beta)))$.
- $(\alpha \vee \beta)$ is short for $((\neg\alpha) \to \beta)$.
- $(\alpha \leftrightarrow \beta)$ is short for $((\alpha \to \beta) \wedge (\beta \to \alpha))$.
- $\exists v_k \varphi$ is short for $(\neg\forall v_k(\neg\varphi))$.

($\forall$ is often called the universal quantifier and $\exists$ is often called the existential quantifier.)

Parentheses will often be omitted in formulas according to the same conventions we used in propositional logic, with the modification that $\forall$ and $\exists$ take precedence over all the logical connectives:

- We will usually drop the outermost parentheses in a formula, writing $\alpha \to \beta$ instead of $(\alpha \to \beta)$ and $\neg\alpha$ instead of $(\neg\alpha)$.
- We will let $\forall$ take precedence over $\neg$, and $\neg$ take precedence over $\to$ when parentheses are missing, and fit the informal abbreviations into this scheme by letting the order of precedence be $\forall$, $\exists$, $\neg$, $\wedge$, $\vee$, $\to$, and $\leftrightarrow$.

- Finally, we will group repetitions of $\rightarrow$, $\vee$, $\wedge$, or $\leftrightarrow$ to the right when parentheses are missing, so $\alpha \rightarrow \beta \rightarrow \gamma$ is short for $(\alpha \rightarrow (\beta \rightarrow \gamma))$.

For example, $\exists v_k \neg \alpha \rightarrow \forall v_n \beta$ is short for $((\neg \forall v_k (\neg(\neg \alpha))) \rightarrow \forall v_n \beta)$. On the other hand, we will sometimes add parentheses and arrange things in unofficial ways to make terms and formulas easier to read. In particular we will often write

(1) $f(t_1, \ldots, t_k)$ for $f t_1 \ldots t_k$ if f is a k-place function symbol and $t_1, \ldots, t_k$ are terms,
(2) $s \circ t$ for $\circ st$ if $\circ$ is a 2-place function symbol and s and t are terms,
(3) $P(t_1, \ldots, t_k)$ for $P t_1 \ldots t_k$ if P is a k-place relation symbol and $t_1, \ldots, t_k$ are terms,
(4) $s \bullet t$ for $\bullet st$ if $\bullet$ is a 2-place relation symbol and s and t are terms, and
(5) $s = t$ for $= st$ if s and t are terms, and
(6) enclose terms in parentheses to group them.

Thus, we could write the formula $= +1 \cdot 0 v_6 \cdot 11$ of $\mathcal{L}_{NT}$ as $1 + (0 \cdot v_6) = 1 \cdot 1$.

As was observed in Example 5.1, it is customary in devising a formal language to recycle the same symbols used informally for the given objects. In situations where we want to talk about symbols without committing ourselves to a particular one, such as when talking about first-order languages in general, we will often use "generic" choices:

- a, b, c, ... for constant symbols;
- x, y, z, ... for variable symbols;
- f, g, h, ... for function symbols;
- P, Q, R, ... for relation symbols; and
- r, s, t, ... for generic terms.

These can be thought of as variables in the metalanguage[4] ranging over different kinds objects of first-order logic, much as we're already using lower-case Greek characters as variables which range over formulas. (In fact, we have already used some of these conventions in this chapter...)

Unique Readability. The slightly paranoid might ask whether Definitions 5.1, 5.2 and 5.3 actually ensure that the terms and formulas of a first-order language $\mathcal{L}$ are unambiguous, *i.e.* cannot be read in

[4]The metalanguage is the language, mathematical English in this case, in which we talk *about* a language. The theorems we prove about formal logic are, strictly speaking, metatheorems, as opposed to the theorems proved within a formal logical system. For more of this kind of stuff, read some philosophy...

more than one way. As with $\mathcal{L}_P$, to actually prove this one must assume that all the symbols of $\mathcal{L}$ are distinct and that no symbol is a subsequence of any other symbol. It then follows that:

THEOREM 5.15. *Any term of a first-order language $\mathcal{L}$ satisfies exactly one of conditions 1–3 in Definition 5.2.*

THEOREM 5.16 (Unique Readability Theorem). *Any formula of a first-order language satisfies exactly one of conditions 1–5 in Definition 5.3.*

CHAPTER 6

Structures and Models

Defining truth and implication in first-order logic is a lot harder than it was in propositional logic. First-order languages are intended to deal with mathematical objects like groups or linear orders, so it makes little sense to speak of the truth of a formula without specifying a context. For example, one can write down a formula expressing the commutative law in a language for group theory, $\forall x \, \forall y \, x \cdot y = y \cdot x$, but whether it is true or not depends on which group we're dealing with. It follows that we need to make precise which mathematical objects or structures a given first-order language can be used to discuss and how, given a suitable structure, formulas in the language are to be interpreted. Such a structure for a given language should supply most of the ingredients needed to interpret formulas of the language. Throughout this chapter, let $\mathcal{L}$ be an arbitrary fixed countable first-order language. All formulas will be assumed to be formulas of $\mathcal{L}$ unless stated otherwise.

DEFINITION 6.1. A *structure* $\mathfrak{M}$ for $\mathcal{L}$ consists of the following:

(1) A non-empty set M, often written as $|\mathfrak{M}|$, called the *universe* of $\mathfrak{M}$.
(2) For each constant symbol c of $\mathcal{L}$, an element $c^{\mathfrak{M}}$ of M.
(3) For each k-place function symbol f of $\mathcal{L}$, a function $f^{\mathfrak{M}} : M^k \to M$, *i.e.* a k-place function on M.
(4) For each k-place relation symbol P of $\mathcal{L}$, a relation $P^{\mathfrak{M}} \subseteq M^k$, *i.e.* a k-place relation on M.

That is, a structure supplies an underlying set of elements plus interpretations for the various non-logical symbols of the language: constant symbols are interpreted by particular elements of the underlying set, function symbols by functions on this set, and relation symbols by relations among elements of this set.

It is customary to use upper-case "gothic" characters such as $\mathfrak{M}$ and $\mathfrak{N}$ for structures.

For example, consider $\mathfrak{Q} = (\mathbb{Q}, <)$, where $<$ is the usual "less than" relation on the rationals. This is a structure for $\mathcal{L}_O$, the language for linear orders defined in Example 5.2; it supplies a 2-place relation to

interpret the language's 2-place relation symbol. $\mathfrak{Q}$ is *not* the only possible structure for $\mathcal{L}_O$: $(\mathbb{R}, <)$, $(\{0\}, \emptyset)$, and $(\mathbb{N}, \mathbb{N}^2)$ are three others among infinitely many more. (Note that in these cases the relation symbol $<$ is interpreted by relations on the universe which are not linear orders. One can ensure that a structure satisfy various conditions beyond what Definition 6.1 guarantees by requiring appropriate formulas to be true when interpreted in the structure.) On the other hand, $(\mathbb{R})$ is not a structure for $\mathcal{L}_O$ because it lacks a binary relation to interpret the symbol $<$ by, while $(\mathbb{N}, 0, 1, +, \cdot, |, <)$ is not a structure for $\mathcal{L}_O$ because it has two binary relations where $\mathcal{L}_O$ has a symbol only for one, plus constants and functions for which $\mathcal{L}_O$ lacks symbols.

PROBLEM 6.1. *The first-order languages referred to below were all defined in Example 5.2.*

(1) *Is* $(\emptyset)$ *a structure for* $\mathcal{L}_=$*?*
(2) *Determine whether* $\mathfrak{Q} = (\mathbb{Q}, <)$ *is a structure for each of* $\mathcal{L}_=$*,* $\mathcal{L}_F$*, and* $\mathcal{L}_S$*.*
(3) *Give three different structures for* $\mathcal{L}_F$ *which are not fields.*

To determine what it means for a given formula to be true in a structure for the corresponding language, we will also need to specify how to interpret the variables when they occur free. (Bound variables have the associated quantifier to tell us what to do.)

DEFINITION 6.2. Let $V = \{ v_0, v_1, v_2, \dots \}$ be the set of all variable symbols of $\mathcal{L}$ and suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$. A function $s : V \to |\mathfrak{M}|$ is said to be an *assignment* for $\mathfrak{M}$.

Note that these are *not* truth assignments like those for $\mathcal{L}_P$. An assignment just interprets each variable in the language by an element of the universe of the structure. Also, as long as the universe of the structure has more than one element, any variable can be interpreted in more than one way. Hence there are usually many different possible assignments for a given structure.

EXAMPLE 6.1. Consider the structure $\mathfrak{R} = (\mathbb{R}, 0, 1, +, \cdot)$ for $\mathcal{L}_F$. Each of the following functions $V \to \mathbb{R}$ is an assignment for $\mathfrak{R}$:

(1) $p(v_n) = \pi$ for each n,
(2) $r(v_n) = e^n$ for each n, and
(3) $s(v_n) = n + 1$ for each n.

In fact, *every* function $V \to \mathbb{R}$ is an assignment for $\mathfrak{R}$.

In order to use assignments to determine whether formulas are true in a structure, we need to know how to use an assignment to interpret each term of the language as an element of the universe.

DEFINITION 6.3. Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$ and $s\colon V \to |\mathfrak{M}|$ is an assignment for $\mathfrak{M}$. Let T be the set of all terms of $\mathcal{L}$. Then the *extended assignment* $\mathbf{s}\colon T \to |\mathfrak{M}|$ is defined inductively as follows:

(1) For each variable x, $\mathbf{s}(x) = s(x)$.
(2) For each constant symbol c, $\mathbf{s}(c) = c^{\mathfrak{M}}$.
(3) For every k-place function symbol f and terms $t_1, \ldots, t_k$,

$$\mathbf{s}(ft_1 \ldots t_k) = f^{\mathfrak{M}}(\mathbf{s}(t_1), \ldots, \mathbf{s}(t_k)).$$

EXAMPLE 6.2. Let $\mathfrak{R}$ be the structure for $\mathcal{L}_F$ given in Example 6.1, and let $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{s}$ be the extended assignments corresponding to the assignments p, r, and s defined in Example 6.1. Consider the term $+ \cdot v_6 v_0 + 0 v_3$ of $\mathcal{L}_F$. Then:

(1) $\mathbf{p}(+ \cdot v_6 v_0 + 0 v_3) = \pi^2 + \pi$,
(2) $\mathbf{r}(+ \cdot v_6 v_0 + 0 v_3) = e^6 + e^3$, and
(3) $\mathbf{s}(+ \cdot v_6 v_0 + 0 v_3) = 11$.

Here's why for the last one: since $s(v_6) = 7$, $s(v_0) = 1$, $s(v_3) = 4$, and $\mathbf{s}(0) = 0$ (by part 2 of Definition 6.3), it follows from part 3 of Definition 6.3 that $\mathbf{s}(+ \cdot v_6 v_0 + 0 v_3) = (7 \cdot 1) + (0 + 4) = 7 + 4 = 11$.

PROBLEM 6.2. *$\mathfrak{N} = (\mathbb{N}, 0, S, +, \cdot, E)$ is a structure for $\mathcal{L}_N$. Let $s\colon V \to \mathbb{N}$ be the assignment defined by $s(v_k) = k + 1$. What are $\mathbf{s}(E + v_{19} v_1 \cdot 0 v_{45})$ and $\mathbf{s}(SSS + E 0 v_6 v_7)$?*

PROPOSITION 6.3. $\mathbf{s}$ *is unique,* i.e. *given an assignment s, no other function $T \to |\mathfrak{M}|$ satisfies conditions 1–3 in Definition 6.3.*

With Definitions 6.2 and 6.3 in hand, we can take our first cut at defining what it means for a first-order formula to be true.

DEFINITION 6.4. Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$, s is an assignment for $\mathfrak{M}$, and φ is a formula of $\mathcal{L}$. Then $\mathfrak{M} \models \varphi[s]$ is defined as follows:

(1) If φ is $t_1 = t_2$ for some terms t_1 and t_2, then $\mathfrak{M} \models \varphi[s]$ if and only if $\mathbf{s}(t_1) = \mathbf{s}(t_2)$.
(2) If φ is $Pt_1 \ldots t_k$ for some k-place relation symbol P and terms $t_1, \ldots, t_k$, then $\mathfrak{M} \models \varphi[s]$ if and only if $(\mathbf{s}(t_1), \ldots, \mathbf{s}(t_k)) \in P^{\mathfrak{M}}$, *i.e.* $P^{\mathfrak{M}}$ is true of $(\mathbf{s}(t_1), \ldots, \mathbf{s}(t_k))$.
(3) If φ is $(\neg\psi)$ for some formula ψ, then $\mathfrak{M} \models \varphi[s]$ if and only if it is not the case that $\mathfrak{M} \models \psi[s]$.
(4) If φ is $(\alpha \to \beta)$, then $\mathfrak{M} \models \varphi[s]$ if and only if $\mathfrak{M} \models \beta[s]$ whenever $\mathfrak{M} \models \alpha[s]$, *i.e.* unless $\mathfrak{M} \models \alpha[s]$ but not $\mathfrak{M} \models \beta[s]$.
(5) If φ is $\forall x\, \delta$ for some variable x, then $\mathfrak{M} \models \varphi[s]$ if and only if for all $m \in |\mathfrak{M}|$, $\mathfrak{M} \models \delta[s(x|m)]$, where $s(x|m)$ is the assignment

given by

$$s(x|m)(v_k) = \begin{cases} s(v_k) & \text{if } v_k \text{ is different from } x \\ m & \text{if } v_k \text{ is } x. \end{cases}$$

If $\mathfrak{M} \models \varphi[s]$, we shall say that $\mathfrak{M}$ *satisfies* φ *on assignment* s or that φ *is true in* $\mathfrak{M}$ *on assignment* s. We will often write $\mathfrak{M} \not\models \varphi[s]$ if it is not the case that $\mathfrak{M} \models \varphi[s]$. Also, if Γ is a set of formulas of $\mathcal{L}$, we shall take $\mathfrak{M} \models \Gamma[s]$ to mean that $\mathfrak{M} \models \gamma[s]$ for every formula γ in Γ and say that $\mathfrak{M}$ *satisfies* Γ *on assignment* s. Similarly, we shall take $\mathfrak{M} \not\models \Gamma[s]$ to mean that $\mathfrak{M} \not\models \gamma[s]$ for *some* formula γ in Γ.

Clauses 1 and 2 are pretty straightforward and clauses 3 and 4 are essentially identical to the corresponding parts of Definition 2.1. The key clause is 5, which says that $\forall$ should be interpreted as "for all elements of the universe".

EXAMPLE 6.3. Let $\mathfrak{R}$ be the structure for $\mathcal{L}_F$ and s the assignment for $\mathfrak{R}$ given in Example 6.1, and consider the formula $\forall v_1 (= v_3 \cdot 0 v_1 \rightarrow = v_3 0)$ of $\mathcal{L}_F$. We can verify that $\mathfrak{R} \models \forall v_1 (= v_3 \cdot 0 v_1 \rightarrow = v_3 0)\,[s]$ as follows:

$$\begin{aligned}
&\mathfrak{R} \models \forall v_1 (= v_3 \cdot 0 v_1 \rightarrow = v_3 0)\,[s] \\
\iff &\text{for all } a \in |\mathfrak{R}|,\ \mathfrak{R} \models (= v_3 \cdot 0 v_1 \rightarrow = v_3 0)\,[s(v_1|a)] \\
\iff &\text{for all } a \in |\mathfrak{R}|, \text{ if } \mathfrak{R} \models = v_3 \cdot 0 v_1\,[s(v_1|a)], \\
&\qquad \text{then } \mathfrak{R} \models = v_3 0\,[s(v_1|a)] \\
\iff &\text{for all } a \in |\mathfrak{R}|, \text{ if } \mathbf{s}(v_1|a)(v_3) = \mathbf{s}(v_1|a)(\cdot 0 v_1), \\
&\qquad \text{then } \mathbf{s}(v_1|a)(v_3) = \mathbf{s}(v_1|a)(0) \\
\iff &\text{for all } a \in |\mathfrak{R}|, \text{ if } \mathbf{s}(v_3) = \mathbf{s}(v_1|a)(0) \cdot \mathbf{s}(v_1|a)(v_1), \text{ then } \mathbf{s}(v_3) = 0 \\
\iff &\text{for all } a \in |\mathfrak{R}|, \text{ if } s(v_3) = 0 \cdot a, \text{ then } s(v_3) = 0 \\
\iff &\text{for all } a \in |\mathfrak{R}|, \text{ if } 4 = 0 \cdot a, \text{ then } 4 = 0 \\
\iff &\text{for all } a \in |\mathfrak{R}|, \text{ if } 4 = 0, \text{ then } 4 = 0
\end{aligned}$$

... which last is true whether or not $4 = 0$ is true or false.

PROBLEM 6.4. *Let* $\mathfrak{N}$ *be the structure for* $\mathcal{L}_N$ *in Problem 6.2. Let* $p : V \to \mathbb{N}$ *be defined by* $p(v_{2k}) = k$ *and* $p(v_{2k+1}) = k$. *Verify that*

(1) $\mathfrak{N} \models \forall w\,(\neg Sw = 0)\,[p]$ *and*
(2) $\mathfrak{N} \not\models \forall x \exists y\, x + y = 0\,[p]$.

PROPOSITION 6.5. *Suppose* $\mathfrak{M}$ *is a structure for* $\mathcal{L}$, s *is an assignment for* $\mathfrak{M}$, x *is a variable, and* φ *is a formula of a first-order language* $\mathcal{L}$. *Then* $\mathfrak{M} \models \exists x\, \varphi[s]$ *if and only if* $\mathfrak{M} \models \varphi[s(x|m)]$ *for some* $m \in |\mathfrak{M}|$.

Working with particular assignments is difficult but, while sometimes unavoidable, not always necessary.

DEFINITION 6.5. Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$, and φ a formula of $\mathcal{L}$. Then $\mathfrak{M} \models \varphi$ if and only if $\mathfrak{M} \models \varphi[s]$ for every assignment $s : V \to |\mathfrak{M}|$ for $\mathfrak{M}$. $\mathfrak{M}$ is a *model* of φ or that φ is *true* in $\mathfrak{M}$ if $\mathfrak{M} \models \varphi$. We will often write $\mathfrak{M} \not\models \psi$ if it is not the case that $\mathfrak{M} \models \psi$.

Similarly, if Γ is a set of formulas, we will write $\mathfrak{M} \models \Gamma$ if $\mathfrak{M} \models \gamma$ for every formula $\gamma \in \Gamma$, and say that $\mathfrak{M}$ is a *model* of Γ or that $\mathfrak{M}$ *satisfies* Γ. A formula or set of formulas is *satisfiable* if there is some structure $\mathfrak{M}$ which satisfies it. We will often write $\mathfrak{M} \not\models \Gamma$ if it is not the case that $\mathfrak{M} \models \Gamma$.

NOTE. $\mathfrak{M} \not\models \varphi$ does *not* mean that for every assignment $s : V \to |\mathfrak{M}|$, it is not the case that $\mathfrak{M} \models \varphi[s]$. It only means that that there is *some* assignment $r : V \to |\mathfrak{M}|$ for which $\mathfrak{M} \models \varphi[r]$ is not true.

PROBLEM 6.6. *$\mathfrak{Q} = (\mathbb{Q}, <)$ is a structure for $\mathcal{L}_O$. For each of the following formulas φ of $\mathcal{L}_O$, determine whether or not $\mathfrak{Q} \models \varphi$.*

(1) $\forall v_0 \, \exists v_2 \, v_0 < v_2$
(2) $\exists v_1 \, \forall v_3 \, (v_1 < v_3 \to v_1 = v_3)$
(3) $\forall v_4 \, \forall v_5 \, \forall v_6 (v_4 < v_5 \to (v_5 < v_6 \to v_4 < v_6))$

The following facts are counterparts of sorts for Proposition 2.2. Their point is that what a given assignment does with a given term or formula depends only on the assignment's values on the (free) variables of the term or formula.

LEMMA 6.7. *Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$, t is a term of $\mathcal{L}$, and r and s are assignments for $\mathfrak{M}$ such that $r(x) = s(x)$ for every variable x which occurs in t. Then $\mathbf{r}(t) = \mathbf{s}(t)$.*

PROPOSITION 6.8. *Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$, φ is a formula of $\mathcal{L}$, and r and s are assignments for $\mathfrak{M}$ such that $r(x) = s(x)$ for every variable x which occurs free in φ. Then $\mathfrak{M} \models \varphi[r]$ if and only if $\mathfrak{M} \models \varphi[s]$.*

COROLLARY 6.9. *Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$ and σ is a sentence of $\mathcal{L}$. Then $\mathfrak{M} \models \sigma$ if and only if there is some assignment $s : V \to |\mathfrak{M}|$ for $\mathfrak{M}$ such that $\mathfrak{M} \models \sigma[s]$.*

Thus sentences are true or false in a structure independently of any particular assignment. This does not necessarily make life easier when trying to verify whether a sentence is true in a structure – try doing Problem 6.6 again with the above results in hand – but it does let us

simplify things on occasion when proving things about sentences rather than formulas.

We recycle a sense in which we used $\models$ in propositional logic.

DEFINITION 6.6. Suppose Γ is a set of formulas of $\mathcal{L}$ and ψ is a formula of $\mathcal{L}$. Then Γ *implies* ψ, written as $\Gamma \models \psi$, if $\mathfrak{M} \models \psi$ whenever $\mathfrak{M} \models \Gamma$ for every structure $\mathfrak{M}$ for $\mathcal{L}$.

Similarly, if Γ and Δ are sets of formulas of $\mathcal{L}$, then Γ *implies* Δ, written as $\Gamma \models \Delta$, if $\mathfrak{M} \models \Delta$ whenever $\mathfrak{M} \models \Gamma$ for every structure $\mathfrak{M}$ for $\mathcal{L}$.

We will usually write $\models \ldots$ for $\emptyset \models \ldots$.

PROPOSITION 6.10. *Suppose α and β are formulas of some first-order language. Then $\{ (\alpha \to \beta), \alpha \} \models \beta$.*

PROPOSITION 6.11. *Suppose Σ is a set of formulas and ψ and ρ are formulas of some first-order language. Then $\Sigma \cup \{\psi\} \models \rho$ if and only if $\Sigma \models (\psi \to \rho)$.*

DEFINITION 6.7. A formula ψ of $\mathcal{L}$ is a *tautology* if it is true in every structure, *i.e.* if $\models \psi$. ψ is a *contradiction* if $\neg\psi$ is a tautology, *i.e.* if $\models \neg\psi$.

For some trivial examples, let φ be a formula of $\mathcal{L}$ and $\mathfrak{M}$ a structure for $\mathcal{L}$. Then $\mathfrak{M} \models \{\varphi\}$ if and only if $\mathfrak{M} \models \varphi$, so it must be the case that $\{\varphi\} \models \varphi$. It is also easy to check that $\varphi \to \varphi$ is a tautology and $\neg(\varphi \to \varphi)$ is a contradiction.

PROBLEM 6.12. *Show that $\forall y\, y = y$ is a tautology and that $\exists y \neg y = y$ is a contradiction.*

PROBLEM 6.13. *Suppose φ is a contradiction. Show that $\mathfrak{M} \models \varphi[s]$ is false for every structure $\mathfrak{M}$ and assignment $s : V \to |\mathfrak{M}|$ for $\mathfrak{M}$.*

PROBLEM 6.14. *Show that a set of formulas Σ is satisfiable if and only if there is no contradiction χ such that $\Sigma \models \chi$.*

The following fact is a counterpart of Proposition 2.4.

PROPOSITION 6.15. *Suppose $\mathfrak{M}$ is a structure for $\mathcal{L}$ and α and β are sentences of $\mathcal{L}$. Then:*

(1) $\mathfrak{M} \models \neg\alpha$ *if and only if* $\mathfrak{M} \not\models \alpha$.
(2) $\mathfrak{M} \models \alpha \to \beta$ *if and only if* $\mathfrak{M} \models \beta$ *whenever* $\mathfrak{M} \models \alpha$.
(3) $\mathfrak{M} \models \alpha \vee \beta$ *if and only if* $\mathfrak{M} \models \alpha$ *or* $\mathfrak{M} \models \beta$.
(4) $\mathfrak{M} \models \alpha \wedge \beta$ *if and only if* $\mathfrak{M} \models \alpha$ *and* $\mathfrak{M} \models \beta$.
(5) $\mathfrak{M} \models \alpha \leftrightarrow \beta$ *if and only if* $\mathfrak{M} \models \alpha$ *exactly when* $\mathfrak{M} \models \beta$.
(6) $\mathfrak{M} \models \forall x\, \alpha$ *if and only if* $\mathfrak{M} \models \alpha$.

(7) $\mathfrak{M} \models \exists x\, \alpha$ *if and only if there is some* $m \in |\mathfrak{M}|$ *so that* $\mathfrak{M} \models \alpha\,[s(x|m)]$ *for every assignment* s *for* $\mathfrak{M}$.

PROBLEM 6.16. *How much of Proposition 6.15 must remain true if* α *and* β *are not sentences?*

Recall that by Proposition 5.14 a formula of a first-order language is also a formula of any extension of the language. The following relationship between extension languages and satisfiability will be needed later on.

PROPOSITION 6.17. *Suppose* $\mathcal{L}$ *is a first-order language,* $\mathcal{L}'$ *is an extension of* $\mathcal{L}$*, and* Γ *is a set of formulas of* $\mathcal{L}$*. Then* Γ *is satisfiable in a structure for* $\mathcal{L}$ *if and only if* Γ *is satisfiable in a structure for* $\mathcal{L}'$.

One last bit of terminology...

DEFINITION 6.8. If $\mathfrak{M}$ is a structure for $\mathcal{L}$, then the *theory* of $\mathfrak{M}$ is just the set of all sentences of $\mathcal{L}$ true in $\mathfrak{M}$, *i.e.*

$$\mathrm{Th}(\mathfrak{M}) = \{\, \tau \mid \tau \text{ is a sentence and } \mathfrak{M} \models \tau \,\}.$$

If Δ is a set of sentences and $\mathcal{S}$ is a collection of structures, then Δ is a set of (non-logical) *axioms* for $\mathcal{S}$ if for every structure $\mathfrak{M}$, $\mathfrak{M} \in \mathcal{S}$ if and only if $\mathfrak{M} \models \Delta$.

EXAMPLE 6.4. Consider the sentence $\exists x\, \exists y\, ((\neg x = y) \wedge \forall z\, (z = x \vee z = y))$ of $\mathcal{L}_=$. Every structure of $\mathcal{L}_=$ satisfying this sentence must have exactly two elements in its universe, so $\{\, \exists x\, \exists y\, ((\neg x = y) \wedge \forall z\, (z = x \vee z = y))\,\}$ is a set of non-logical axioms for the collection of sets of cardinality 2:

$$\{\, \mathfrak{M} \mid \mathfrak{M} \text{ is a structure for } \mathcal{L}_= \text{ with exactly 2 elements} \,\}.$$

PROBLEM 6.18. *In each case, find a suitable language and a set of axioms in it for the given collection of structures.*

(1) *Sets of size 3.*
(2) *Bipartite graphs.*
(3) *Commutative groups.*
(4) *Fields of characteristic 5.*

CHAPTER 7

Deductions

Deductions in first-order logic are not unlike deductions in propositional logic. Of course, some changes are necessary to handle the various additional features of propositional logic, especially quantifiers. In particular, one of the new axioms requires a tricky preliminary definition. Roughly, the problem is that we need to know when we can replace occurrences of a variable in a formula by a term without letting any variable in the term get captured by a quantifier.

Throughout this chapter, let $\mathcal{L}$ be a fixed arbitrary first-order language. Unless stated otherwise, all formulas will be assumed to be formulas of $\mathcal{L}$.

DEFINITION 7.1. Suppose x is a variable, t is a term, and φ is a formula. Then *t is substitutable for x in φ* is defined as follows:

(1) If φ is atomic, then t is substitutable for x in φ.
(2) If φ is $(\neg\psi)$, then t is substitutable for x in φ if and only if t is substitutable for x in ψ.
(3) If φ is $(\alpha \to \beta)$, then t is substitutable for x in φ if and only if t is substitutable for x in α and t is substitutable for x in β.
(4) If φ is $\forall y\, \delta$, then t is substitutable for x in φ if and only if either
    (a) x does not occur free in φ, or
    (b) if y does not occur in t and t is substitutable for x in δ.

For example, x is always substitutable for itself in any formula φ and φ^x_x is just φ (see Problem 7.1). On the other hand, y is not substitutable for x in $\forall y\, x = y$ because if x were to be replaced by y, the new instance of y would be "captured" by the quantifier $\forall y$. This makes a difference to the truth of the formula. The truth of $\forall y\, x = y$ depends on the structure in which it is interpreted — it's true if the universe has only one element and false otherwise — but $\forall y\, y = y$ is a tautology by Problem 6.12 so it is true in any structure whatsoever. This sort of difficulty makes it necessary to be careful when substituting for variables.

DEFINITION 7.2. Suppose x is a variable, t is a term, and φ is a formula. If t is substitutable for x in φ, then φ_t^x (*i.e.* φ with t substituted for x) is defined as follows:

(1) If φ is atomic, then φ_t^x is the formula obtained by replacing each occurrence of x in φ by t.
(2) If φ is $(\neg\psi)$, then φ_t^x is the formula $(\neg\psi_t^x)$.
(3) If φ is $(\alpha \to \beta)$, then φ_t^x is the formula $(\alpha_t^x \to \beta_t^x)$.
(4) If φ is $\forall y\, \delta$, then φ_t^x is the formula
    (a) $\forall y\, \delta$ if x is y, and
    (b) $\forall y\, \delta_t^x$ if x isn't y.

PROBLEM 7.1. (1) *Is x substitutable for z in ψ if ψ is $z = x \to \forall z\, z = x$? If so, what is ψ_x^z?*
(2) *Show that if t is any term and σ is a sentence, then t is substitutable in σ for any variable x. What is σ_t^x?*
(3) *Show that if t is a term in which no variable occurs that occurs in the formula φ, then t is substitutable in φ for any variable x.*
(4) *Show that x is substitutable for x in φ for any variable x and any formula φ, and that φ_x^x is just φ.*

Along with the notion of substitutability, we need an additional notion in order to define the logical axioms of $\mathcal{L}$.

DEFINITION 7.3. If φ is any formula and $x_1, \ldots, x_n$ are any variables, then $\forall x_1 \ldots \forall x_n\, \varphi$ is said to be a *generalization* of φ.

For example, $\forall y\, \forall x\, (x = y \to fx = fy)$ and $\forall z\, (x = y \to fx = fy)$ are (different) generalizations of $x = y \to fx = fy$, but $\forall x\, \exists y\, (x = y \to fx = fy)$ is not. Note that the variables being quantified don't have to occur in the formula being generalized.

LEMMA 7.2. *Any generalization of a tautology is a tautology.*

DEFINITION 7.4. Every first-order language $\mathcal{L}$ has eight *logical axiom schema*:

A1: $(\alpha \to (\beta \to \alpha))$
A2: $((\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)))$
A3: $(((\neg\beta) \to (\neg\alpha)) \to (((\neg\beta) \to \alpha) \to \beta))$
A4: $(\forall x\, \alpha \to \alpha_t^x)$, if t is substitutable for x in α.
A5: $(\forall x\, (\alpha \to \beta) \to (\forall x\, \alpha \to \forall x\, \beta))$
A6: $(\alpha \to \forall x\, \alpha)$, if x does not occur free in α.
A7: $x = x$

A8: $(x = y \to (\alpha \to \beta))$, if α is atomic and β is obtained from α by replacing some occurrences (possibly all or none) of x in α by y.

Plugging in any particular formulas of $\mathcal{L}$ for α, β, and γ, and any particular variables for x and y, in any of A1–A8 gives a *logical axiom* of $\mathcal{L}$. In addition, any generalization of a logical axiom of $\mathcal{L}$ is also a logical axiom of $\mathcal{L}$.

The reason for calling the instances of A1–A8 the logical axioms, instead of just axioms, is to avoid conflict with Definition 6.8.

PROBLEM 7.3. *Determine whether or not each of the following formulas is a logical axiom.*

(1) $\forall x \, \forall z \, (x = y \to (x = c \to x = y))$
(2) $x = y \to (y = z \to z = x)$
(3) $\forall z \, (x = y \to (x = c \to y = c))$
(4) $\forall w \, \exists x \, (Pwx \to Pww) \to \exists x \, (Pxx \to Pxx)$
(5) $\forall x \, (\forall x \, c = fxc \to \forall x \, \forall x \, c = fxc)$
(6) $(\exists x \, Px \to \exists y \, \forall z \, Rzfy) \to ((\exists x \, Px \to \forall y \, \neg \forall z \, Rzfy) \to \forall x \, \neg Px)$

PROPOSITION 7.4. *Every logical axiom is a tautology.*

Note that we have recycled our axiom schemas A1—A3 from propositional logic. We will also recycle MP as the sole rule of inference for first-order logic.

DEFINITION 7.5 (Modus Ponens). Given the formulas φ and $(\varphi \to \psi)$, one may infer ψ.

As in propositional logic, we will usually refer to Modus Ponens by its initials, MP. That MP preserves truth in the sense of Chapter 6 follows from Problem 6.10. Using the logical axioms and MP, we can execute deductions in first-order logic just as we did in propositional logic.

DEFINITION 7.6. Let Δ be a set of formulas of the first-order language $\mathcal{L}$. A *deduction* or *proof* from Δ in $\mathcal{L}$ is a finite sequence $\varphi_1 \varphi_2 \ldots \varphi_n$ of formulas of $\mathcal{L}$ such that for each $k \leq n$,

(1) φ_k is a logical axiom, or
(2) $\varphi_k \in \Delta$, or
(3) there are $i, j < k$ such that φ_k follows from φ_i and φ_j by MP.

A formula of Δ appearing in the deduction is usually referred to as a *premiss* of the deduction. Δ *proves* a formula α, written as $\Delta \vdash \alpha$, if α is the last formula of a deduction from Δ. We'll usually write $\vdash \alpha$

instead of $\emptyset \vdash \alpha$. Finally, if Γ and Δ are sets of formulas, we'll take $\Gamma \vdash \Delta$ to mean that $\Gamma \vdash \delta$ for every formula $\delta \in \Delta$.

NOTE. We have reused the axiom schema, the rule of inference, and the definition of deduction from propositional logic. It follows that any deduction of propositional logic can be converted into a deduction of first-order logic simply by replacing the formulas of $\mathcal{L}_P$ occurring in the deduction by first-order formulas. Feel free to appeal to the deductions in the exercises and problems of Chapter 3. *You should probably review the Examples and Problems of Chapter 3 before going on, since most of the rest of this Chapter concentrates on what is* different *about deductions in first-order logic.*

EXAMPLE 7.1. We'll show that $\{\alpha\} \vdash \exists x\, \alpha$ for any first-order formula α and any variable x.

(1) $(\forall x\, \neg\alpha \to \neg\alpha) \to (\alpha \to \neg\forall x\, \neg\alpha)$	Problem 3.9.5
(2) $\forall x\, \neg\alpha \to \neg\alpha$	A4
(3) $\alpha \to \neg\forall x\, \neg\alpha$	1,2 MP
(4) α	Premiss
(5) $\neg\forall x\, \neg\alpha$	3,4 MP
(6) $\exists x\, \alpha$	Definition of $\exists$

Strictly speaking, the last line is just for our convenience, like $\exists$ itself.

PROBLEM 7.5. *Show that:*

(1) $\vdash \forall x\, \varphi \to \forall y\, \varphi^x_y$, *if* y *does not occur at all in* φ.
(2) $\vdash \alpha \vee \neg\alpha$.
(3) $\{c = d\} \vdash \forall z\, Qazc \to Qazd$.
(4) $\vdash x = y \to y = x$.
(5) $\{\exists x\, \alpha\} \vdash \alpha$ *if* x *does not occur free in* α.

Many general facts about deductions can be recycled from propositional logic, including the Deduction Theorem.

PROPOSITION 7.6. *If* $\varphi_1\varphi_2 \dots \varphi_n$ *is a deduction of* $\mathcal{L}$, *then* $\varphi_1 \dots \varphi_\ell$ *is also a deduction of* $\mathcal{L}$ *for any* ℓ *such that* $1 \leq \ell \leq n$.

PROPOSITION 7.7. *If* $\Gamma \vdash \delta$ *and* $\Gamma \vdash \delta \to \beta$, *then* $\Gamma \vdash \beta$.

PROPOSITION 7.8. *If* $\Gamma \subseteq \Delta$ *and* $\Gamma \vdash \alpha$, *then* $\Delta \vdash \alpha$.

PROPOSITION 7.9. *Then if* $\Gamma \vdash \Delta$ *and* $\Delta \vdash \sigma$, *then* $\Gamma \vdash \sigma$.

THEOREM 7.10 (Deduction Theorem). *If* Σ *is any set of formulas and* α *and* β *are any formulas, then* $\Sigma \vdash \alpha \to \beta$ *if and only if* $\Sigma \cup \{\alpha\} \vdash \beta$.

Just as in propositional logic, the Deduction Theorem is useful because it often lets us take shortcuts when trying to show that deductions exist. There is also another result about first-order deductions which often supplies useful shortcuts.

THEOREM 7.11 (Generalization Theorem). *Suppose x is a variable, Γ is a set of formulas in which x does not occur free, and φ is a formula such that $\Gamma \vdash \varphi$. Then $\Gamma \vdash \forall x\, \varphi$.*

THEOREM 7.12 (Generalization On Constants). *Suppose that c is a constant symbol, Γ is a set of formulas in which c does not occur, and φ is a formula such that $\Gamma \vdash \varphi$. Then there is a variable x which does not occur in φ such that $\Gamma \vdash \forall x\, \varphi^c_x$.*[1] *Moreover, there is a deduction of $\forall x\, \varphi^c_x$ from Γ in which c does not occur.*

EXAMPLE 7.2. We'll show that if φ and ψ are any formulas, x is any variable, and $\vdash \varphi \to \psi$, then $\vdash \forall x\, \varphi \to \forall x\, \psi$.

Since x does not occur free in any formula of $\emptyset$, it follows from $\vdash \varphi \to \psi$ by the Generalization Theorem that $\vdash \forall x\, (\varphi \to \psi)$. But then

(1) $\forall x\, (\varphi \to \psi)$	above
(2) $\forall x\, (\varphi \to \psi) \to (\forall x\, \varphi \to \forall x\, \psi)$	A5
(3) $\forall x\, \varphi \to \forall x\, \psi$	1,2 MP

is the tail end of a deduction of $\forall x\, \varphi \to \forall x\, \psi$ from $\emptyset$.

PROBLEM 7.13. *Show that:*

(1) $\vdash \forall x\, \forall y\, \forall z\, (x = y \to (y = z \to x = z))$.
(2) $\vdash \forall x\, \alpha \to \exists x\, \alpha$.
(3) $\vdash \exists x\, \gamma \to \forall x\, \gamma$ *if x does not occur free in γ.*

We conclude with a bit of terminology.

DEFINITION 7.7. If Σ is a set of sentences, then the *theory* of Σ is

$$\mathrm{Th}(\Sigma) = \{\, \tau \mid \tau \text{ is a sentence and } \Sigma \vdash \tau \,\}.$$

That is, the theory of Σ is just the collection of all sentences which can be proved from Σ.

[1] φ^c_x is φ with every occurence of the constant c replaced by x.

CHAPTER 8

Soundness and Completeness

As with propositional logic, first-order logic had better satisfy the Soundness Theorem and it is desirable that it satisfy the Completeness Theorem. These theorems do hold for first-order logic. The Soundness Theorem is proved in a way similar to its counterpart for propositional logic, but the Completeness Theorem will require a fair bit of additional work.[1] It is in this extra work that the distinction between formulas and sentences becomes useful.

Let $\mathcal{L}$ be a fixed countable first-order language throughout this chapter. All formulas will be assumed to be formulas of $\mathcal{L}$ unless stated otherwise.

First, we rehash many of the definitions and facts we proved for propositional logic in Chapter 4 for first-order logic.

THEOREM 8.1 (Soundness Theorem). *If α is a sentence and Δ is a set of sentences such that $\Delta \vdash \alpha$, then $\Delta \models \alpha$.*

DEFINITION 8.1. A set of sentences Γ is *inconsistent* if $\Gamma \vdash \neg(\psi \to \psi)$ for some formula ψ, and is *consistent* if it is not inconsistent.

Recall that a set of sentences Γ is satisfiable if $\mathfrak{M} \models \Gamma$ for some structure $\mathfrak{M}$.

PROPOSITION 8.2. *If a set of sentences Γ is satisfiable, then it is consistent.*

PROPOSITION 8.3. *Suppose Δ is an inconsistent set of sentences. Then $\Delta \vdash \psi$ for any formula ψ.*

PROPOSITION 8.4. *Suppose Σ is an inconsistent set of sentences. Then there is a finite subset Δ of Σ such that Δ is inconsistent.*

COROLLARY 8.5. *A set of sentences Γ is consistent if and only if every finite subset of Γ is consistent.*

[1] This is not too surprising because of the greater complexity of first-order logic. Also, it turns out that first-order logic is about as powerful as a logic can get and still have the Completeness Theorem hold.

DEFINITION 8.2. A set of sentences Σ is *maximally consistent* if Σ is consistent but $\Sigma \cup \{\tau\}$ is inconsistent whenever τ is a sentence such that $\tau \notin \Sigma$.

One quick way of finding examples of maximally consistent sets is given by the following proposition.

PROPOSITION 8.6. *If $\mathfrak{M}$ is a structure, then Th($\mathfrak{M}$) is a maximally consistent set of sentences.*

EXAMPLE 8.1. $\mathfrak{M} = (\{5\})$ is a structure for $\mathcal{L}_=$, so $\mathrm{Th}(\mathfrak{M})$ is a maximally consistent set of sentences. Since it turns out that $\mathrm{Th}(\mathfrak{M}) = \mathrm{Th}(\{\forall x \forall y\, x = y\})$, this also gives us an example of a set of sentences $\Sigma = \{\forall x \forall y\, x = y\}$ such that $\mathrm{Th}(\Sigma)$ is maximally consistent.

PROPOSITION 8.7. *If Σ is a maximally consistent set of sentences, τ is a sentence, and $\Sigma \vdash \tau$, then $\tau \in \Sigma$.*

PROPOSITION 8.8. *Suppose Σ is a maximally consistent set of sentences and τ is a sentence. Then $\neg\tau \in \Sigma$ if and only if $\tau \notin \Sigma$.*

PROPOSITION 8.9. *Suppose Σ is a maximally consistent set of sentences and φ and ψ are any sentences. Then $\varphi \to \psi \in \Sigma$ if and only if $\varphi \notin \Sigma$ or $\psi \in \Sigma$.*

THEOREM 8.10. *Suppose Γ is a consistent set of sentences. Then there is a maximally consistent set of sentences Σ with $\Gamma \subseteq \Sigma$.*

The counterparts of these notions and facts for propositional logic sufficed to prove the Completeness Theorem, but here we will need some additional tools. The basic problem is that instead of defining a suitable truth assignment from a maximally consistent set of formulas, we need to construct a suitable structure from a maximally consistent set of sentences. Unfortunately, structures for first-order languages are usually more complex than truth assignments for propositional logic. The following definition supplies the key new idea we will use to prove the Completeness Theorem.

DEFINITION 8.3. Suppose Σ is a set of sentences and C is a set of (some of the) constant symbols of $\mathcal{L}$. Then C is a *set of witnesses* for Σ in $\mathcal{L}$ if for every formula φ of $\mathcal{L}$ with at most one free variable x, there is a constant symbol $c \in C$ such that $\Sigma \vdash \exists x\, \varphi \to \varphi^x_c$.

The idea is that every element of the universe which Σ proves must exist is named, or "witnessed", by a constant symbol in C. Note that if $\Sigma \vdash \neg\exists x\, \varphi$, then $\Sigma \vdash \exists x\, \varphi \to \varphi^x_c$ for any constant symbol c.

PROPOSITION 8.11. *Suppose* Γ *and* Σ *are sets of sentences of* $\mathcal{L}$*,* $\Gamma \subseteq \Sigma$*, and* C *is a set of witnesses for* Γ *in* $\mathcal{L}$*. Then* C *is a set of witnesses for* Σ *in* $\mathcal{L}$*.*

EXAMPLE 8.2. Let $\mathcal{L}'_O$ be the first-order language with a single 2-place relation symbol, $<$, and countably many constant symbols, c_q for each $q \in \mathbb{Q}$. Let Σ include all the sentences

(1) $c_p < c_q$, for every $p, q \in \mathbb{Q}$ such that $p < q$,
(2) $\forall x\, (\neg x < x)$,
(3) $\forall x\, \forall y\, (x < y \vee x = y \vee y < x)$,
(4) $\forall x\, \forall y\, \forall z\, (x < y \to (y < z \to x < z))$,
(5) $\forall x\, \forall y\, (x < y \to \exists z\, (x < z \wedge z < y))$,
(6) $\forall x\, \exists y\, (x < y)$, and
(7) $\forall x\, \exists y\, (y < x)$.

In effect, Σ asserts that $<$ is a linear order on the universe (2–4) which is dense (5) and has no endpoints (6–7), and which has a suborder isomorphic to $\mathbb{Q}$ (1). Then $C = \{\, c_q \mid q \in \mathbb{Q} \,\}$ is a set of witnesses for Σ in $\mathcal{L}'_O$.

In the example above, one can "reverse-engineer" a model for the set of sentences in question from the set of witnesses simply by letting the universe of the structure *be* the set of witnesses. One can also define the necessary relation interpreting $<$ in a pretty obvious way from Σ.[2] This example is obviously contrived: there are no constant symbols around which are not witnesses, Σ proves that distinct constant symbols aren't equal to to each other, there is little by way of non-logical symbols needing interpretation, and Σ explicitly includes everything we need to know about $<$.

In general, trying to build a model for a set of sentences Σ in this way runs into a number of problems. First, how do we know whether Σ has a set of witnesses at all? Many first-order languages have few or no constant symbols, after all. Second, if Σ has a set of witnesses C, it's unlikely that we'll be able to get away with just letting the universe of the model be C. What if $\Sigma \vdash c = d$ for some distinct witnesses c and d? Third, how do we handle interpreting constant symbols which are not in C? Fourth, what if Σ doesn't prove enough about whatever relation and function symbols exist to let us define interpretations of them in the structure under construction? (Imagine, if you like, that someone hands you a copy of Joyce's *Ulysses* and asks you to produce a

[2]Note, however, that an isomorphic copy of $\mathbb{Q}$ is not the only structure for $\mathcal{L}'_O$ satisfying Σ. For example, $\mathfrak{R} = (\mathbb{R}, <, q + \pi \colon q \in \mathbb{Q})$ will also satisfy Σ if we intepret c_q by $q + \pi$.

complete road map of Dublin on the basis of the book. Even if it has no geographic contradictions, you are unlikely to find all the information in the novel needed to do the job.) Finally, even if Σ does prove all we need to define functions and relations on the universe to interpret the function and relation symbols, just how do we do it? Getting around all these difficulties requires a fair bit of work. One can get around many by sticking to maximally consistent sets of sentences in suitable languages.

LEMMA 8.12. *Suppose Σ is a set of sentences, φ is any formula, and x is any variable. Then $\Sigma \vdash \varphi$ if and only if $\Sigma \vdash \forall x\, \varphi$.*

THEOREM 8.13. *Suppose Γ is a consistent set of sentences of $\mathcal{L}$. Let C be an infinite countable set of constant symbols which are* not *symbols of $\mathcal{L}$, and let $\mathcal{L}' = \mathcal{L} \cup C$ be the language obtained by adding the constant symbols in C to the symbols of $\mathcal{L}$. Then there is a maximally consistent set Σ of sentences of $\mathcal{L}'$ such that $\Gamma \subseteq \Sigma$ and C is a set of witnesses for Σ.*

This theorem allows one to use a certain measure of brute force: No set of witnesses? Just add one! The set of sentences doesn't decide enough? Decide *everything* one way or the other!

THEOREM 8.14. *Suppose Σ is a maximally consistent set of sentences and C is a set of witnesses for Σ. Then there is a structure $\mathfrak{M}$ such that $\mathfrak{M} \models \Sigma$.*

The important part here is to define $\mathfrak{M}$ — proving that $\mathfrak{M} \models \Sigma$ is tedious but fairly straightforward if you have the right definition. Proposition 6.17 now lets us deduce the fact we really need.

COROLLARY 8.15. *Suppose Γ is a consistent set of sentences of a first-order language $\mathcal{L}$. Then there is a structure $\mathfrak{M}$ for $\mathcal{L}$ satisfying Γ.*

With the above facts in hand, we can rejoin our proof of Soundness and Completeness, already in progress:

THEOREM 8.16. *A set of sentences Σ in $\mathcal{L}$ is consistent if and only if it is satisfiable.*

The rest works just like it did for propositional logic.

THEOREM 8.17 (Completeness Theorem). *If α is a sentence and Δ is a set of sentences such that $\Delta \models \alpha$, then $\Delta \vdash \alpha$.*

It follows that in a first-order logic, as in propositional logic, a sentence is implied by some set of premisses if and only if it has a proof from those premisses.

THEOREM 8.18 (Compactness Theorem). *A set of sentences Δ is satisfiable if and only if every finite subset of Δ is satisfiable.*

CHAPTER 9

Applications of Compactness

After wading through the preceding chapters, it should be obvious that first-order logic is, in principle, adequate for the job it was originally developed for: the essentially philosophical exercise of formalizing most of mathematics. As something of a bonus, first-order logic can supply useful tools for doing "real" mathematics. The Compactness Theorem is the simplest of these tools and glimpses of two ways of using it are provided below.

From the finite to the infinite. Perhaps the simplest use of the Compactness Theorem is to show that if there exist arbitrarily large finite objects of some type, then there must also be an infinite object of this type.

EXAMPLE 9.1. We will use the Compactness Theorem to show that there is an infinite commutative group in which every element is of order 2, *i.e.* such that $g \cdot g = e$ for every element g.

Let $\mathcal{L}_G$ be the first-order language with just two non-logical symbols:

- Constant symbol: e
- 2-place function symbol: $\cdot$

Here e is intended to name the group's identity element and $\cdot$ the group operation. Let Σ be the set of sentences of $\mathcal{L}_G$ including:

(1) The axioms for a commutative group:
   - $\forall x\, x \cdot e = x$
   - $\forall x\, \exists y\, x \cdot y = e$
   - $\forall x\, \forall y\, \forall z\, x \cdot (y \cdot z) = (x \cdot y) \cdot z$
   - $\forall x\, \forall y\, y \cdot x = x \cdot y$

(2) A sentence which asserts that every element of the universe is of order 2:
   - $\forall x\, x \cdot x = e$

(3) For each $n \geq 2$, a sentence, σ_n, which asserts that there are at least n different elements in the universe:
   - $\exists x_1 \ldots \exists x_n\, ((\neg x_1 = x_2) \wedge (\neg x_1 = x_3) \wedge \cdots \wedge (\neg x_{n-1} = x_n))$

We claim that every finite subset of Σ is satisfiable. The most direct way to verify this is to show how, given a finite subset Δ of Σ, to produce a model $\mathfrak{M}$ of Δ. Let n be the largest integer such that $\sigma_n \in \Delta \cup \{\sigma_2\}$ (Why is there such an n?) and choose an integer k such that $2^k \geq n$. Define a structure $(G, \circ)$ for $\mathcal{L}_G$ as follows:

- $G = \{\, \langle a_\ell \mid 1 \leq \ell \leq k \rangle \mid a_\ell = 0 \text{ or } 1 \,\}$
- $\langle a_\ell \mid 1 \leq \ell \leq k \rangle \circ \langle b_\ell \mid 1 \leq \ell \leq k \rangle = \langle a_\ell + b_\ell \pmod 2 \mid 1 \leq \ell \leq k \rangle$

That is, G is the set of binary sequences of length k and $\circ$ is coordinatewise addition modulo 2 of these sequences. It is easy to check that $(G, \circ)$ is a commutative group with 2^k elements in which every element has order 2. Hence $(G, \circ) \models \Delta$, so Δ is satisfiable.

Since every finite subset of Σ is satisfiable, it follows by the Compactness Theorem that Σ is satisfiable. A model of Σ, however, must be an infinite commutative group in which every element is of order 2. (To be sure, it is quite easy to build such a group directly; for example, by using coordinatewise addition modulo 2 of infinite binary sequences.)

PROBLEM 9.1. *Use the Compactness Theorem to show that there is an infinite*

(1) *bipartite graph,*
(2) *non-commutative group, and*
(3) *field of characteristic 3,*

and also give concrete examples of such objects.

Most applications of this method, including the ones above, are not really interesting: it is usually more valuable, and often easier, to directly construct examples of the infinite objects in question rather than just show such must exist. Sometimes, though, the technique can be used to obtain a non-trivial result more easily than by direct methods. We'll use it to prove an important result from graph theory, Ramsey's Theorem. Some definitions first:

DEFINITION 9.1. If X is a set, let the set of unordered pairs of elements of X be $[X]^2 = \{\, \{a, b\} \mid a, b \in X \text{ and } a \neq b \,\}$. (See Definition A.1.)

(1) A *graph* is a pair (V, E) such that V is a non-empty set and $E \subseteq [V]^2$. Elements of V are called *vertices* of the graph and elements of E are called *edges*.
(2) A *subgraph* of (V, E) is a pair (U, F), where $U \subset V$ and $F = E \cap [U]^2$.

(3) A subgraph (U, F) of (V, E) is a *clique* if $F = [U]^2$.
(4) A subgraph (U, F) of (V, E) is an *independent set* if $F = \emptyset$.

That is, a graph is some collection of vertices, some of which are joined to one another. A subgraph is just a subset of the vertices, together with all edges joining vertices of this subset in the whole graph. It is a clique if it happens that the original graph joined every vertex in the subgraph to all other vertices in the subgraph, and an independent set if it happens that the original graph joined none of the vertices in the subgraph to each other. The question of when a graph must have a clique or independent set of a given size is of some interest in many applications, especially in dealing with colouring problems.

THEOREM 9.2 (Ramsey's Theorem). *For every $n \geq 1$ there is an integer R_n such that any graph with at least R_n vertices has a clique with n vertices or an independent set with n vertices.*

R_n is the *nth Ramsey number*. It is easy to see that $R_1 = 1$ and $R_2 = 2$, but R_3 is already 6, and R_n grows very quickly as a function of n thereafter. Ramsey's Theorem is fairly hard to prove directly, but the corresponding result for infinite graphs is comparatively straightforward.

LEMMA 9.3. *If (V, E) is a graph with infinitely many vertices, then it has an infinite clique or an infinite independent set.*

A relatively quick way to prove Ramsey's Theorem is to first prove its infinite counterpart, Lemma 9.3, and then get Ramsey's Theorem out of it by way of the Compactness Theorem. (If you're an ambitious minimalist, you can try to do this using the Compactness Theorem for propositional logic instead!)

Elementary equivalence and non-standard models. One of the common uses for the Compactness Theorem is to construct "non-standard" models of the theories satisfied by various standard mathematical structures. Such a model satisfies all the same first-order sentences as the standard model, but differs from it in some way not expressible in the first-order language in question. This brings home one of the intrinsic limitations of first-order logic: it can't always tell essentially different structures apart. Of course, we need to define just what constitutes essential difference.

DEFINITION 9.2. Suppose $\mathcal{L}$ is a first-order language and $\mathfrak{N}$ and $\mathfrak{M}$ are two structures for $\mathcal{L}$. Then $\mathfrak{N}$ and $\mathfrak{M}$ are:

(1) *isomorphic*, written as $\mathfrak{N} \cong \mathfrak{M}$, if there is a function $F\colon |\mathfrak{N}| \to |\mathfrak{M}|$ such that

(a) F is $1-1$ and onto,
(b) $F(c^{\mathfrak{N}}) = c^{\mathfrak{M}}$ for every constant symbol c of $\mathcal{L}$,
(c) $F(f^{\mathfrak{N}}(a_1, \ldots, a_k) = f^{\mathfrak{M}}(F(a_1), \ldots, F(a_k))$ for every k-place function symbol f of $\mathcal{L}$ and elements $a_1, \ldots, a_k \in |\mathfrak{N}|$, and
(d) $P^{\mathfrak{N}}(a_1, \ldots, a_k)$ holds if and only if $P^{\mathfrak{N}}(F(a_1), \ldots, F(a_k))$ for every k-place relation symbol of $\mathcal{L}$ and elements a_1, $\ldots$, a_k of $|\mathfrak{N}|$;

and

(2) *elementarily equivalent*, written as $\mathfrak{N} \equiv \mathfrak{M}$, if $\mathrm{Th}(\mathfrak{N}) = \mathrm{Th}(\mathfrak{M})$, *i.e.* if $\mathfrak{N} \models \sigma$ if and only if $\mathfrak{M} \models \sigma$ for every sentence σ of $\mathcal{L}$.

That is, two structures for a given language are isomorphic if they are structurally identical and elementarily equivalent if no statement in the language can distinguish between them. Isomorphic structures are elementarily equivalent:

PROPOSITION 9.4. *Suppose $\mathcal{L}$ is a first-order language and $\mathfrak{N}$ and $\mathfrak{M}$ are structures for $\mathcal{L}$ such that $\mathfrak{N} \cong \mathfrak{M}$. Then $\mathfrak{N} \equiv \mathfrak{M}$.*

However, as the following application of the Compactness Theorem shows, elementarily equivalent structures need not be isomorphic:

EXAMPLE 9.2. Note that $\mathfrak{C} = (\mathbb{N})$ is an infinite structure for $\mathcal{L}_=$. Expand $\mathcal{L}_=$ to $\mathcal{L}_R$ by adding a constant symbol c_r for every real number r, and let Σ be the set of sentences of $\mathcal{L}_=$ including

- every sentence τ of $\mathrm{Th}(\mathfrak{C})$, *i.e.* such that $\mathfrak{C} \models \tau$, and
- $\neg c_r = c_s$ for every pair of real numbers r and s such that $r \neq s$.

Every finite subset of Σ is satisfiable. (Why?) Thus, by the Compactness Theorem, there is a structure $\mathfrak{U}'$ for $\mathcal{L}_R$ satisfying Σ, and hence $\mathrm{Th}(\mathfrak{C})$. The structure $\mathfrak{U}$ obtained by dropping the interpretations of all the constant symbols c_r from $\mathfrak{U}'$ is then a structure for $\mathcal{L}_=$ which satisfies $\mathrm{Th}(\mathfrak{C})$. Note that $|\mathfrak{U}| = |\mathfrak{U}'|$ is at least large as the set of all real numbers $\mathbb{R}$, since $\mathfrak{U}'$ requires a distinct element of the universe to interpret each constant symbol c_r of $\mathcal{L}_R$.

Since $\mathrm{Th}(\mathfrak{C})$ is a maximally consistent set of sentences of $\mathcal{L}_=$ by Problem 8.6, it follows from the above that $\mathfrak{C} \equiv \mathfrak{U}$. On the other hand, $\mathfrak{C}$ cannot be isomorphic to $\mathfrak{U}$ because there cannot be an onto map between a countable set, such as $\mathbb{N} = |\mathfrak{C}|$, and a set which is at least as large as $\mathbb{R}$, such as $|\mathfrak{U}|$.

In general, the method used above can be used to show that if a set of sentences in a first-order language has an infinite model, it has many different ones. In $\mathcal{L}_=$ that is essentially all that can happen:

PROPOSITION 9.5. *Two structures for $\mathcal{L}_=$ are elementarily equivalent if and only if they are isomorphic or infinite.*

PROBLEM 9.6. *Let $\mathfrak{N} = (\mathbb{N}, 0, 1, S, +, \cdot, E)$ be the standard structure for $\mathcal{L}_N$. Use the Compactness Theorem to show there is a structure $\mathfrak{M}$ for $\mathcal{L}_N$ such that $\mathfrak{N} \equiv \mathfrak{N}$ but not $\mathfrak{N} \cong \mathfrak{M}$.*

Note that because $\mathfrak{N}$ and $\mathfrak{M}$ both satisfy $\mathrm{Th}(\mathfrak{N})$, which is maximally consistent by Problem 8.6, there is absolutely no way of telling them apart in $\mathcal{L}_N$.

PROPOSITION 9.7. *Every model of* $\mathrm{Th}(\mathfrak{N})$ *which is* not *isomorphic to $\mathfrak{N}$ has*

(1) *an isomorphic copy of $\mathfrak{N}$ embedded in it,*
(2) *an infinite number,* i.e. *one larger than all of those in the copy of $\mathfrak{N}$, and*
(3) *an infinite decreasing sequence.*

The apparent limitation of first-order logic that non-isomorphic structures may be elementarily equivalent can actually be useful. A non-standard model may have features that make it easier to work with than the standard model one is really interested in. Since both structures satisfy exactly the same sentences, if one uses these features to prove that some statement expressible in the given first-order language is true about the non-standard structure, one gets for free that it must be true of the standard structure as well. A prime example of this idea is the use of non-standard models of the real numbers containing infinitesimals (numbers which are infinitely small but different from zero) in some areas of analysis.

THEOREM 9.8. *Let $\mathfrak{R} = (\mathbb{R}, 0, 1, +, \cdot)$ be the field of real numbers, considered as a structure for $\mathcal{L}_F$. Then there is a model of Th($\mathfrak{R}$) which contains a copy of $\mathbb{R}$ and in which there is an infinitesimal.*

The non-standard models of the real numbers actually used in analysis are usually obtained in more sophisticated ways in order to have more information about their internal structure. It is interesting to note that infinitesimals were the intuition behind calculus for Leibniz when it was first invented, but no one was able to put their use on a rigourous footing until Abraham Robinson did so in 1950.

Hints for Chapters 5–9

Hints for Chapter 5.

5.1. Try to disassemble each string using Definition 5.2. Note that some might be valid terms of more than one of the given languages.

5.2. This is similar to Problem 1.5.

5.3. This is similar to Proposition 1.7.

5.4. Try to disassemble each string using Definitions 5.2 and 5.3. Note that some might be valid formulas of more than one of the given languages.

5.5. This is just like Problem 1.2.

5.6. This is similar to Problem 1.5. You may wish to use your solution to Problem 5.2.

5.7. This is similar to Proposition 1.7.

5.8. You might want to rephrase some of the given statements to make them easier to formalize.

(1) Look up associativity if you need to.
(2) "There is an object such that every object is not in it."
(3) This should be easy.
(4) Ditto.
(5) "Any two things must be the same thing."

5.9. If necessary, don't hesitate to look up the definitions of the given structures.

(1) Read the discussion at the beginning of the chapter.
(2) You really need only one non-logical symbol.
(3) There are two sorts of objects in a vector space, the vectors themselves and the scalars of the field, which you need to be able to tell apart.

5.10. Use Definition 5.3 in the same way that Definition 1.2 was used in Definition 1.3.

5.11. The scope of a quantifier ought to be a certain subformula of the formula in which the quantifier occurs.

5.12. Check to see whether they satisfy Definition 5.4.

5.13. Check to see which pairs satisfy Definition 5.5.

5.14. Proceed by induction on the length of φ using Definition 5.3.

5.15. This is similar to Theorem 1.12.

5.16. This is similar to Theorem 1.12 and uses Theorem 5.15.

Hints for Chapter 6.

6.1. In each case, apply Definition 6.1.

(1) This should be easy.
(2) Ditto.
(3) Invent objects which are completely different except that they happen to have the right number of the right kind of components.

6.2. Figure out the relevant values of $s(v_n)$ and apply Definition 6.3.

6.3. Suppose $\mathbf{s}$ and $\mathbf{r}$ both extend the assignment s. Show that $\mathbf{s}(t) = \mathbf{r}(t)$ by induction on the length of the term t.

6.4. Unwind the formulas using Definition 6.4 to get informal statements whose truth you can determine.

6.5. Unwind the abbreviation $\exists$ and use Definition 6.4.

6.6. Unwind each of the formulas using Definitions 6.4 and 6.5 to get informal statements whose truth you can determine.

6.7. This is much like Proposition 6.3.

6.8. Proceed by induction on the length of the formula using Definition 6.4 and Lemma 6.7.

6.9. How many free variables does a sentence have?

6.10. Use Definition 6.4.

6.12. Unwind the sentences in question using Definition 6.4.

6.11. Use Definitions 6.4 and 6.5; the proof is similar in form to the proof of Proposition 2.9.

6.14. Use Definitions 6.4 and 6.5; the proof is similar in form to the proof for Problem 2.10.

6.15. Use Definitions 6.4 and 6.5 in each case, plus the meanings of our abbreviations.

6.17. In one direction, you need to add appropriate objects to a structure; in the other, delete them. In both cases, you still have to verify that Γ is still satisfied.

6.18. Here are some appropriate languages.

(1) $\mathcal{L}_=$
(2) Modify your language for graph theory from Problem 5.9 by adding a 1-place relation symbol.
(3) Use your language for group theory from Problem 5.9.
(4) $\mathcal{L}_F$

Hints for Chapter 7.

7.1. (1) Use Definition 7.1.
(2) Ditto.
(3) Ditto.
(4) Proceed by induction on the length of the formula φ.

7.2. Use the definitions and facts about $\models$ from Chapter 6.

7.3. Check each case against the schema in Definition 7.4. Don't forget that any generalization of a logical axiom is also a logical axiom.

7.4. You need to show that any instance of the schemas A1–A8 is a tautology and then apply Lemma 7.2. That each instance of schemas A1–A3 is a tautology follows from Proposition 6.15. For A4–A8 you'll have to use the definitions and facts about $\models$ from Chapter 6.

7.5. You may wish to appeal to the deductions that you made or were given in Chapter 3.

(1) Try using A4 and A6.
(2) You don't need A4–A8 here.
(3) Try using A4 and A8.
(4) A8 is the key; you may need it more than once.
(5) This is just A6 in disguise.

7.6. This is just like its counterpart for propositional logic.

7.7. Ditto.

7.8. Ditto.

7.9. Ditto.

7.10. Ditto.

7.11. Proceed by induction on the length of the shortest proof of φ from Γ.

7.12. Ditto.

7.13. As usual, don't take the following suggestions as gospel.

(1) Try using A8.
(2) Start with Example 7.1.
(3) Start with part of Problem 7.5.

Hints for Chapter 8.

8.1. This is similar to the proof of the Soundness Theorem for propositional logic, using Proposition 6.10 in place of Proposition 3.2.

8.2. This is similar to its counterpart for prpositional logic, Proposition 4.2. Use Proposition 6.10 instead of Proposition 3.2.

8.3. This is just like its counterpart for propositional logic.

8.4. Ditto.

8.5. Ditto.

8.6. This is a counterpart to Problem 4.6; use Proposition 8.2 instead of Proposition 4.2 and Proposition 6.15 instead of Proposition 2.4.

8.7. This is just like its counterpart for propositional logic.

8.8. Ditto

8.9. Ditto.

8.10. This is much like its counterpart for propositional logic, Theorem 4.10.

8.11. Use Proposition 7.8.

8.12. Use the Generalization Theorem for the hard direction.

8.13. This is essentially a souped-up version of Theorem 8.10. To ensure that C is a set of witnesses of the maximally consistent set of sentences, enumerate all the formulas φ of $\mathcal{L}'$ with one free variable and take care of one at each step in the inductive construction.

8.14. To construct the required structure, $\mathfrak{M}$, proceed as follows. Define an equivalence relation $\sim$ on C by setting $c \sim d$ if and only if $c = d \in \Sigma$, and let $[c] = \{\, a \in C \mid a \sim c \,\}$ be the equivalence class of $c \in C$. The universe of $\mathfrak{M}$ will be $M = \{\, [c] \mid c \in C \,\}$. For each k-place function symbol f define $f^{\mathfrak{M}}$ by setting $f^{\mathfrak{M}}([a_1], \ldots, [a_k]) = [b]$ if and only if $fa_1 \ldots a_k = b$ is in Σ. Define the interpretations of constant symbols and relation symbols in a similar way. You need to show that all these things are well-defined, and then show that $\mathfrak{M} \models \Sigma$.

8.15. Expand Γ to a maximally consistent set of sentences with a set of witnesses in a suitable extension of $\mathcal{L}$, apply Theorem 8.14, and then cut down the resulting structure to one for $\mathcal{L}$.

8.16. One direction is just Proposition 8.2. For the other, use Corollary 8.15.

8.17. This follows from Theorem 8.16 in the same way that the Completeness Theorem for propositional logic followed from Theorem 4.11.

8.18. This follows from Theorem 8.16 in the same way that the Compactness Theorem for propositional logic followed from Theorem 4.11.

Hints for Chapter 9.

9.1. In each case, apply the trick used in Example 9.1. For definitions and the concrete examples, consult texts on combinatorics and abstract algebra.

9.2. Suppose Ramsey's Theorem fails for some n. Use the Compactness Theorem to get a contradiction to Lemma 9.3 by showing there must be an infnite graph with no clique or independent set of size n.

9.3. Inductively define a sequence a_0, a_1, ..., of vertices so that for every n, either it is the case that for all $k \geq n$ there is an edge joining a_n to a_k or it is the case that for all $k \geq n$ there is no edge joining a_n to a_k. There will then be a subsequence of the sequence which is an infinite clique or a subsequence which is an infinite independent set.

9.4. The key is to figure out how, given an assignment for one structure, one should define the corresponding assignment in the other structure. After that, proceed by induction using the definition of satisfaction.

9.5. When are two finite structures for $\mathcal{L}_=$ elementarily equivalent?

9.6. In a suitable expanded language, consider $\mathrm{Th}(\mathfrak{N})$ together with the sentences $\exists x\, 0 + x = c$, $\exists x\, S0 + x = c$, $\exists x\, SS0 + x = c$, ...

9.7. Suppose $\mathfrak{M} \models \mathrm{Th}(\mathfrak{N})$ but is not isomorphic to $\mathfrak{N}$.

(1) Consider the subset of $|\mathfrak{M}|$ given by $0^{\mathfrak{M}}$, $S^{\mathfrak{M}}(0^{\mathfrak{M}})$, $S^{\mathfrak{M}}(S^{\mathfrak{M}}(0^{\mathfrak{M}}))$, ...
(2) If it didn't have one, it would be a copy of $\mathfrak{N}$.
(3) Start with a infinite number and work down.

9.8. Expand $\mathcal{L}_F$ by throwing in a constant symbol for every real number, plus an extra one, and take it from there.

Part III

Computability

CHAPTER 10

Turing Machines

Of the various ways to formalize the notion an "effective method", the most commonly used are the simple abstract computers called Turing machines, which were introduced more or less simultaneously by Alan Turing and Emil Post in 1936.[1] Like most real-life digital computers, Turing machines have two main parts, a processing unit and a memory (which doubles as the input/output device), which we will consider separately before seeing how they interact. The memory can be thought of as an infinite tape which is divided up into cells like the frames of a movie. The Turing machine proper is the processing unit. It has a scanner or head which can read from or write to a single cell of the tape, and which can be moved to the left or right one cell at a time.

Tapes. To keep things simple, in this chapter we will only allow Turing machines to read and write the symbols 0 and 1. (One symbol per cell!) Moreover, we will allow the tape to be infinite in only one direction. That these restrictions do not affect what a Turing machine can, in principle, compute follows from the results in the next chapter.

Definition 10.1. A *tape* is an infinite sequence

$$\mathbf{a} = a_0\, a_1\, a_2\, a_3 \ldots$$

such that for each integer i the *cell* $a_i \in \{0, 1\}$. The ith cell is said to be *blank* if a_i is 0, and *marked* if a_i is 1.

A blank tape is one in which every cell is 0.

Example 10.1. A blank tape looks like:

$$000000000000000000000000\cdots$$

The 0th cell is the leftmost one, cell 1 is the one immediately to the right, cell 2 is the one immediately to the right of cell 1, and so on.

The following is a slightly more exciting tape:

$$010110111000100000000000000\cdots$$

[1]Both papers are reprinted in [**6**]. Post's brief paper gives a particularly lucid informal description.

In this case, cell 1 is marked (*i.e.* contains a 1), as do cells 3, 4, 5, 7, 8, and 12; all the rest are blank (*i.e.* contain a 0).

PROBLEM 10.1. *Write down tapes satisfying the following.*

(1) *Entirely blank except for cells* 3*,* 12*, and* 20*.*
(2) *Entirely marked except for cells* 0*,* 2*, and* 3*.*
(3) *Entirely blank except that 1025 is written out in binary just to the right of cell* 2*.*

To keep track of which cell the Turing machine's scanner is at, plus which instruction the Turing machine is to execute next, we will usually attach additional information to our description of the tape.

DEFINITION 10.2. A *tape position* is a triple $(s, i, \mathbf{a})$, where s and i are natural numbers with $s > 0$, and $\mathbf{a}$ is a tape. Given a tape position $(s, i, \mathbf{a})$, we will refer to cell i as the *scanned cell* and to s as the *state*.

Note that if $(s, i, \mathbf{a})$ is a tape position, then the corresponding Turing machine's scanner is presently reading a_i (which is one of 0 or 1).

Conventions for tapes. Unless stated otherwise, we will assume that all but finitely many cells of any given tape are blank, and that any cells not explicitly described or displayed are blank. We will usually depict as little of a tape as possible and omit the $\cdots$s we used above. Thus

$$0101101110001$$

represents the tape given in the Example 10.1. In many cases we will also use z^n to abbreviate n consecutive copies of z, so the same tape could be represented by

$$0101^2 01^3 0^3 1\,.$$

Similarly, if σ is a finite sequence of elements of $\{0, 1\}$, we may write σ^n for the sequence consisting of n copies of σ stuck together end-to-end. For example, $(010)^3$ is short for 010010010.

In displaying tape positions we will usually underline the scanned cell and write s to the left of the tape. For example, we would display the tape position using the tape from Example 10.1 with cell 3 being scanned and state 2 as follows:

$$2\colon 010\underline{1}101110001$$

Note that in this example, the scanner is reading a 1.

PROBLEM 10.2. *Using the tapes you gave in the corresponding part of Problem 10.1, write down tape positions satisfying the following conditions.*

(1) *Cell* 7 *being scanned and state* 4.
(2) *Cell* 4 *being scanned and state* 3.
(3) *Cell* 3 *being scanned and state* 413.

Turing machines. The "processing unit" of a Turing machine is just a finite list of specifications describing what the machine will do in various situations. (Remember, this is an *abstract* computer...) The formal definition may not seem to amount to this at first glance.

DEFINITION 10.3. A *Turing machine* is a function M such that for some natural number n,

$$\begin{aligned} \mathrm{dom}(M) &\subseteq \{1,\ldots,n\} \times \{0,1\} \\ &= \{\,(s,b) \mid 1 \le s \le n \text{ and } b \in \{0,1\}\,\} \end{aligned}$$

and

$$\begin{aligned} \mathrm{ran}(M) &\subseteq \{0,1\} \times \{-1,1\} \times \{1,\ldots,n\} \\ &= \{\,(c,d,t) \mid c \in \{0,1\} \text{ and } d \in \{-1,1\} \text{ and } 1 \le t \le n\,\}. \end{aligned}$$

Note that M does not have to be defined for all possible pairs

$$(s,b) \in \{1,\ldots,n\} \times \{0,1\}.$$

We will sometimes refer to a Turing machine simply as a *machine* or TM . If $n \ge 1$ is least such that M satisfies the definition above, we shall say that M is an *n-state Turing machine* and that $\{1,\ldots,n\}$ is the set of *states* of M.

Intuitively, we have a processing unit which has a finite list of basic instructions, the states, which it can execute. Given a combination of current state and the symbol marked in the currently scanned cell of the tape, the list specifies

- a symbol to be written in the currently scanned cell, overwriting the symbol being read, then
- a move of the scanner one cell to the left or right, and then
- the next instruction to be executed.

That is, $M(s,c) = (b,d,t)$ means that if our machine is in state s (*i.e.* executing instruction number s) and the scanner is presently reading a c in cell i, then the machine M should

- set $a_i = b$ (*i.e.* write b instead of c in the scanned cell), then
- move the scanner to a_{i+d} (*i.e.* move one cell left if $d = -1$ and one cell right if $d = 1$), and then
- enter state t (*i.e.* go to instruction t).

If our processor isn't equipped to handle input c for instruction s (*i.e.* $M(s,c)$ is undefined), then the computation in progress will simply stop dead or *halt*.

EXAMPLE 10.2. We will usually present Turing machines in the form of a table, with a row for each state and a column for each possible entry in the scanned cell. Instead of -1 and 1, we will usually use L and R when writing such tables in order to make them more readable. Thus the table

M	0	1
1	$1R2$	$0R1$
2	$0L2$	

defines a Turing machine M with two states such that $M(1,0) = (1,1,2)$, $M(1,1) = (0,1,1)$, and $M(2,0) = (0,-1,2)$, but $M(2,1)$ is undefined. In this case M has domain $\{(1,0), (1,1), (2,0)\}$ and range $\{(1,1,2), (0,1,1), (0,-1,2)\}$. If the machine M were faced with the tape position

$$1\colon 0100\underline{1}1111\,,$$

it would, since it was in state 1 while scanning a cell containing 0,

- write a 1 in the scanned cell,
- move the scanner one cell to the right, and
- go to state 2.

This would give the new tape position

$$2\colon 0101\underline{1}111\,.$$

Since M doesn't know what to do on input 1 in state 2, it would then halt, ending the computation.

PROBLEM 10.3. *In each case, give the table of a Turing machine M meeting the given requirement.*

(1) *M has three states.*
(2) *M changes* 0 *to* 1 *and* vice versa *in any cell it scans.*
(3) *M is as simple as possible. How many possibilities are there here?*

Computations. Informally, a computation is a sequence of actions of a machine M on a tape according to the rules above, starting with instruction 1 and the scanner at cell 0 on the given tape. A computation ends (or *halts*) when and if the machine encounters a tape position which it does not know what to do in If it never halts, and doesn't *crash* by running the scanner off the left end of the tape[2] either, the

[2]Be warned that most authors prefer to treat running the scanner off the left end of the tape as being just another way of halting. Halting with the scanner

computation will never end. The formal definition makes all this seem much more formidable.

DEFINITION 10.4. Suppose M is a Turing machine. Then:

- If $p = (s, i, \mathbf{a})$ is a tape position and $M(s, a_i) = (b, d, t)$ is defined, then $\mathbf{M}(p) = (t, i+d, \mathbf{a}')$ is the *successor tape position*, where $a'_i = b$ and $a'_j = a_j$ whenever $j \neq i$.
- A *partial computation* with respect to M is a sequence $p_1 p_2 \dots$ of tape positions such that $p_{\ell+1} = \mathbf{M}(p_\ell)$ for each $\ell < k$.
- A partial computation $p_1 p_2 \dots p_k$ with respect to M is a *computation* (with respect to M) with *input tape* $\mathbf{a}$ if $p_1 = (1, 0, \mathbf{a})$ and $\mathbf{M}(p_k)$ is undefined (and *not* because the scanner would run off the end of the tape). The *output tape* of the computation is the tape of the final tape position p_k.

Note that a partial computation is a computation only if the Turing machine halts but doesn't crash in the final tape position. The requirement that it halt means that any computation can have only finitely many steps. Unless stated otherwise, we will assume that every partial computation on a given input begins in state 1. We will often omit the "partial" when speaking of computations that might not strictly satisfy the definition of computation.

EXAMPLE 10.3. Let's see the machine M of Example 10.2 perform a computation. Our input tape will be $\mathbf{a} = 1100$, that is, the tape which is entirely blank except that $a_0 = a_1 = 1$. The initial tape position of the computation of M with input tape $\mathbf{a}$ is:

$$1\colon \underline{1}100$$

The subsequent steps in the computation are:

$$1\colon 0\underline{1}00$$
$$1\colon 00\underline{0}0$$
$$2\colon 001\underline{0}$$
$$2\colon 00\underline{1}$$

We leave it to the reader to check that this is indeed a partial computation with respect to M. Since $M(2, 1)$ is undefined the process terminates at this point and this partial computation is therefore a computation.

on the tape is more convenient, however, when putting together different Turing machines to make more complex ones.

PROBLEM 10.4. *Give the (partial) computation of the Turing machine M of Example 10.2 starting in state* 1 *with the input tape:*

(1) $\underline{0}0$
(2) $1\underline{1}0$
(3) *The tape with all cells marked and cell* 5 *being scanned.*

PROBLEM 10.5. *For which possible input tapes does the partial computation of the Turing machine M of Example 10.2 eventually terminate? Explain why.*

PROBLEM 10.6. *Find a Turing machine that (eventually!) fills a blank input tape with the pattern* $01011000101100010110 0 \ldots$.

PROBLEM 10.7. *Find a Turing machine that never halts (or crashes), no matter what is on the tape.*

Building Turing Machines. It will be useful later on to have a library of Turing machines that manipulate blocks of 1s in various ways, and very useful to be able to combine machines peforming simpler tasks to perform more complex ones.

EXAMPLE 10.4. The Turing machine S given below is intended to halt with output $01^k\underline{0}$ on input $\underline{0}1^k$, if $k > 0$; that is, it just moves past a single block of 1s without disturbing it.

S	0	1
1	$0R2$	
2		$1R2$

Trace this machine's computation on, say, input $\underline{0}1^3$ to see how it works.

The following machine, which is itself a variation on S, does the reverse of what S does: on input $01^k\underline{0}$ it halts with output $\underline{0}1^k$.

T	0	1
1	$0L2$	
2		$1L2$

We can combine S and T into a machine U which does nothing to a block of 1s: given input $\underline{0}1^k$ it halts with output $\underline{0}1^k$. (Of course, a better way to do nothing is to really do nothing!)

T	0	1
1	$0R2$	
2	$0L3$	$1R2$
3		$1L3$

Note how the states of T had to be renumbered to make the combination work.

EXAMPLE 10.5. The Turing machine P given below is intended to move a block of 1s: on input $\underline{0}0^n1^k$, where $n \geq 0$ and $k > 0$, it halts with output $\underline{0}1^k$.

P	0	1
1	$0R2$	
2	$1R3$	$1L8$
3	$0R3$	$0R4$
4	$0R7$	$1L5$
5	$0L5$	$1R6$
6	$1R3$	
7	$0L7$	$1L8$
8		$1L8$

Trace P's computation on, say, input $\underline{0}0^31^3$ to see how it works. Trace it on inputs $\underline{0}1^2$ and $\underline{0}0^21$ as well to see how it handles certain special cases.

NOTE. In both Examples 10.4 and 10.5 we do not really care what the given machines do on other inputs, so long as they perform as intended on the particular inputs we are concerned with.

PROBLEM 10.8. *We can combine the machine P of Example 10.5 with the machines S and T of Example 10.4 to get the following machine.*

R	0	1
1	$0R2$	
2	$0R3$	$1R2$
3	$1R4$	$1L9$
4	$0R4$	$0R5$
5	$0R8$	$1L6$
6	$0L6$	$1R7$
7	$1R4$	
8	$0L8$	$1L9$
9	$0L10$	$1L9$
10		$1L10$

What task involving blocks of 1*s is this machine intended to perform?*

PROBLEM 10.9. *In each case, devise a Turing machine that:*

(1) *Halts with output $\underline{0}1^4$ on input $\underline{0}$.*
(2) *Halts with output $01^n\underline{0}$ on input $\underline{0}0^n1$.*
(3) *Halts with output $\underline{0}1^{2n}$ on input $\underline{0}1^n$.*
(4) *Halts with output $\underline{0}(10)^n$ on input $\underline{0}1^n$.*
(5) *Halts with output $\underline{0}1^m$ on input $\underline{0}1^n01^m$ whenever $n, m > 0$.*

(6) *Halts with output* $\underline{0}1^m01^n01^k$ *on input* $\underline{0}1^n01^k01^m$*, if* $n, m, k > 0$.
(7) *Halts with output* $\underline{0}1^m01^n01^k01^m01^n01^k$ *on input* $\underline{0}1^m01^n01^k$*, if* $n, m, k > 0$.
(8) *On input* $\underline{0}1^m01^n$*, where* $m, n > 0$*, halts with output* $\underline{0}1$ *if* $m \neq n$ *and output* $\underline{0}11$ *if* $m = n$.

It doesn't matter what the machine you define in each case may do on other inputs, so long as it does the right thing on the given one(s).

CHAPTER 11

Variations and Simulations

The definition of a Turing machine given in Chapter 10 is arbitrary in a number of ways, among them the use of the symbols 0 and 1, a single read-write scanner, and a single one-way infinite tape. One could further restrict the definition we gave by allowing

- the machine to move the scanner only to one of left or right in each state,

or expand it by allowing the use of

- any finite alphabet of at least two symbols,
- separate read and write heads,
- multiple heads,
- two-way infinite tapes,
- multiple tapes,
- two- and higher-dimensional tapes,

or various combinations of these, among many other possibilities. We will construct a number of Turing machines that simulate others with additional features; this will show that various of the modifications mentioned above really change what the machines can compute. (In fact, none of them turn out to do so.)

Example 11.1. Consider the following Turing machine:

M	0	1
1	$1R2$	$0L1$
2	$0L2$	$1L1$

Note that in state 1, this machine may move the scanner to either the left or the right, depending on the contents of the cell being scanned. We will construct a Turing machine using the same alphabet that emulates the action of M on any input, but which moves the scanner to only one of left or right in each state. There is no problem with state 2 of M, by the way, because in state 2 M always moves the scanner to the left.

The basic idea is to add some states to M which replace part of the description of state 1.

M'	0	1
1	$1R2$	$0R3$
2	$0L2$	$1L1$
3	$0L4$	$1L4$
4	$0L1$	

This machine is just like M except that in state 1 with input 1, instead of moving the scanner to the left and going to state 1, the machine moves the scanner to the right and goes to the new state 3. States 3 and 4 do nothing between them except move the scanner two cells to the left without changing the tape, thus putting it where M would have put it, and then entering state 1, as M would have.

PROBLEM 11.1. *Compare the computations of the machines M and M' of Example 11.1 on the input tapes*

(1) 0
(2) 011

and explain why is it not necessary to define M' for state 4 on input 1.

PROBLEM 11.2. *Explain in detail how, given an arbitrary Turing machine M, one can construct a machine M' that simulates what M does on any input, but which moves the scanner only to one of left or right in each state.*

It should be obvious that the converse, simulating a Turing machine that moves the scanner only to one of left or right in each state by an ordinary Turing machine, is easy to the point of being trivial.

It is often very convenient to add additional symbols to the alphabet that Turing machines are permitted to use. For example, one might want to have special symbols to use as place markers in the course of a computation. (For a more spectacular application, see Example 11.3 below.) It is conventional to include 0, the "blank" symbol, in an alphabet used by a Turing machine, but otherwise any finite set of symbols goes.

PROBLEM 11.3. *How do you need to change Definitions 10.1 and 10.3 to define Turing machines using a finite alphabet Σ?*

While allowing arbitary alphabets is often convenient when designing a machine to perform some task, it doesn't actually change what can, in principle, be computed.

EXAMPLE 11.2. Consider the machine W below which uses the alphabet $\{0, x, y, z\}$.

W	0	x	y	z
1	$0R1$	$xR1$	$0L2$	$zR1$

For example, on input $\underline{0}xzyxy$, W will eventually halt with output $0x\underline{z}0xy$. Note that state 2 of W is used only to halt, so we don't bother to make a row for it on the table.

To simulate W with a machine Z using the alphabet $\{0, 1\}$, we first have to decide how to represent W's tape. We will use the following scheme, arbitrarily chosen among a number of alternatives. Every cell of W's tape will be represented by two consecutive cells of Z's tape, with a 0 on W's tape being stored as 00 on Z's, an x as 01, a y as 10, and a z as 11. Thus, if W had input tape $\underline{0}xzyxy$, the corresponding input tape for Z would be $\underline{0}$00111100110.

Designing the machine Z that simulates the action of W on the representation of W's tape is a little tricky. In the example below, each state of W corresponds to a "subroutine" of states of Z which between them read the information in each representation of a cell of W's tape and take appropriate action.

Z	0	1
1	$0R2$	$1R3$
2	$0L4$	$1L6$
3	$0L8$	$1L13$
4	$0R5$	
5	$0R1$	
6	$0R7$	
7		$1R1$
8		$0R9$
9	$0L10$	
10	$0L11$	
11	$0L12$	$1L12$
12	$0L15$	$1L15$
13		$1R14$
14		$1R1$

States 1–3 of Z read the input for state 1 of W and then pass on control to subroutines handling each entry for state 1 in W's table. Thus states 4–5 of Z take action for state 1 of W on input 0, states 6–7 of Z take action for state 1 of W on input x, states 8–12 of Z take action for state 1 of W on input y, and states 13–14 take action for state 1 of W on input z. State 15 of Z does what state 2 of W does: nothing but halt.

PROBLEM 11.4. *Trace the (partial) computations of W, and their counterparts for Z, for the input $\underline{0}xzyxy$ for W. Why is the subroutine for state 1 of W on input y so much longer than the others? How much can you simplify it?*

PROBLEM 11.5. *Given a Turing machine M with an arbitrary alphabet Σ, explain in detail how to construct a machine N with alphabet $\{0,1\}$ that simulates M.*

Doing the converse of this problem, simulating a Turing machine with alphabet $\{0,1\}$ by one using an arbitrary alphabet, is pretty easy.

To define Turing machines with two-way infinite tapes we need only change Definition 10.1: instead of having tapes $\mathbf{a} = a_0a_1a_2\ldots$ indexed by $\mathbb{N}$, we let them be $\mathbf{b} = \ldots b_{-2}b_{-1}b_0b_1b_2\ldots$ indexed by $\mathbb{Z}$. In defining computations for machines with two-way infinite tapes, we adopt the same conventions that we did for machines with one-way infinite tapes, such as having the scanner start off scanning cell 0 on the input tape. The only real difference is that a machine with a two-way infinite tape cannot crash by running off the left end of the tape; it can only stop by halting.

EXAMPLE 11.3. Consider the following two-way infinite tape Turing machine with alphabet $\{0,1\}$:

T	0	1
1	$1L1$	$0R2$
2	$0R2$	$1L1$

To emulate T with a Turing machine O that has a one-way infinite tape, we need to decide how to represent a two-way infinite tape on a one-way infinite tape. This is easier to do if we allow ourselves to use an alphabet for O other than $\{0,1\}$, chosen with malice aforethought:

$$\left\{ \begin{smallmatrix} 0 \\ S \end{smallmatrix}, \begin{smallmatrix} 1 \\ S \end{smallmatrix}, \begin{smallmatrix} 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 1 \end{smallmatrix} \right\}$$

We can now represent the tape $\mathbf{a} = \ldots a_{-2}a_{-1}a_0a_1a_2\ldots$ for T by the tape $\mathbf{a}' = \begin{smallmatrix} a_0 \\ S \end{smallmatrix} \begin{smallmatrix} a_1 \\ a_{-1} \end{smallmatrix} \begin{smallmatrix} a_2 \\ a_{-2} \end{smallmatrix} \ldots$ for O. In effect, this trick allows us to split O's tape into two tracks, each of which accomodates half of the tape of T.

To define O, we split each state of T into a pair of states for O, one for the lower track and one for the upper track. One must take care to keep various details straight: when O changes a "cell" on one track, it should not change the corresponding "cell" on the other track; directions are reversed on the lower track; one has to "turn a corner" moving past cell 0; and so on.

O	0	$\begin{smallmatrix} 0 \\ S \end{smallmatrix}$	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}$	$\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$	$\begin{smallmatrix} 1 \\ S \end{smallmatrix}$	$\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}$	$\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}$
1	$\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}L1$	$\begin{smallmatrix} 1 \\ S \end{smallmatrix}R3$	$\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}L1$	$\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}L1$	$\begin{smallmatrix} 0 \\ S \end{smallmatrix}R2$	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}R2$	$\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}R2$
2	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}R2$	$\begin{smallmatrix} 0 \\ S \end{smallmatrix}R2$	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}R2$	$\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}R2$	$\begin{smallmatrix} 1 \\ S \end{smallmatrix}R3$	$\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}L1$	$\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}L1$
3	$\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}R3$	$\begin{smallmatrix} 1 \\ S \end{smallmatrix}R3$	$\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}R3$	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}L4$	$\begin{smallmatrix} 0 \\ S \end{smallmatrix}R2$	$\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}R3$	$\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}L4$
4	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}L4$	$\begin{smallmatrix} 0 \\ S \end{smallmatrix}R2$	$\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}L4$	$\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}R3$	$\begin{smallmatrix} 1 \\ S \end{smallmatrix}R3$	$\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}L4$	$\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}R3$

States 1 and 3 are the upper- and lower-track versions, respectively, of T's state 1; states 2 and 4 are the upper- and lower-track versions, respectively, of T's state 2. We leave it to the reader to check that O actually does simulate T...

PROBLEM 11.6. *Trace the (partial) computations of T, and their counterparts for O, for each of the following input tapes for T:*

(1) $\underline{0}$ *(*i.e. *a blank tape)*
(2) $1\underline{0}$
(3) $\ldots 111\underline{1}111\ldots$ *(*i.e. *every cell marked with* 1*)*

PROBLEM 11.7. *Explain in detail how, given a Turing machine N with alphabet Σ and a two-way infinite tape, one can construct a Turing machine P with an one-way infinite tape that simulates N.*

PROBLEM 11.8. *Explain in detail how, given a Turing machine P with alphabet Σ and an one-way infinite tape, one can construct a Turing machine N with a two-way infinite tape that simulates P.*

Combining the techniques we've used so far, we could simulate any Turing machine with a two-way infinite tape and arbitrary alphabet by a Turing machine with a one-way infinite tape and alphabet $\{0, 1\}$.

PROBLEM 11.9. *Give a precise definition for Turing machines with two tapes. Explain how, given any such machine, one could construct a single-tape machine to simulate it.*

PROBLEM 11.10. *Give a precise definition for Turing machines with two-dimensional tapes. Explain how, given any such machine, one could construct a single-tape machine to simulate it.*

These results, and others like them, imply that none of the variant types of Turing machines mentioned at the start of this chapter differ essentially in what they can, in principle, compute.

In Chapter 14 we will construct a Turing machine that can simulate *any* (standard) Turing machine.

CHAPTER 12

Computable and Non-Computable Functions

A lot of computational problems in the real world have to do with doing arithmetic, and any notion of computation that can't deal with arithmetic is unlikely to be of great use.

Notation and conventions. To keep things as simple as possible, we will stick to computations involving the *natural numbers*, *i.e.* the non-negative integers, the set of which is usually denoted by $\mathbb{N} = \{ 0, 1, 2, \dots \}$.. The set of all k-tuples $(n_1, \dots, n_k)$ of natural numbers is denoted by $\mathbb{N}^k$. For all practical purposes, we may take $\mathbb{N}^1$ to be $\mathbb{N}$ by identifying the 1-tuple (n) with the natural number n.

For $k \geq 1$, f is a *k-place function* (from the natural numbers to the natural numbers), often written as $f \colon \mathbb{N}^k \to \mathbb{N}$, if it associates a value, $f(n_1, \dots, n_k)$, to each k-tuple $(n_1, n_2, \dots, n_k) \in \mathbb{N}^k$. Strictly speaking, though we will frequently forget to be explicit about it, we will often be working with k-place *partial functions* which might not be defined for all the k-tuples in $\mathbb{N}^k$. If f is a k-place partial function, the *domain* of f is the set

$$\mathrm{dom}(f) = \{ (n_1, \dots, n_k) \in \mathbb{N}^k \mid f(n_1, \dots, n_k) \text{ is defined} \} .$$

Similarly, the *range* of f is the set

$$\mathrm{ran}(f) = \{ f(n_1, \dots, n_k) \in \mathbb{N} \mid (n_1, \dots, n_k) \in \mathrm{dom}(f) \} .$$

In subsequent chapters we will also work with relations on the natural numbers. Recall that a *k-place relation* on $\mathbb{N}$ is formally a subset P of $\mathbb{N}^k$; $P(n_1, \dots, n_k)$ is *true* if $(n_1, \dots, n_k) \in P$ and *false* otherwise. In particular, a 1-place relation is really just a subset of $\mathbb{N}$.

Relations and functions are closely related. All one needs to know about a k-place function f can be obtained from the $(k+1)$-place relation P_f given by

$$P_f(n_1, \dots, n_k, n_{k+1}) \iff f(n_1, \dots, n_k) = n_{k+1} .$$

Similarly, all one needs to know about the k-place relation P can be obtained from its *characteristic function* :

$$\chi_P(n_1, \ldots, n_k) = \begin{cases} 1 & \text{if } P(n_1, \ldots, n_k) \text{ is true;} \\ 0 & \text{if } P(n_1, \ldots, n_k) \text{ is false.} \end{cases}$$

The basic convention for representing natural numbers on the tape of a standard Turing machine is a slight variation of *unary notation* : n is represented by 1^{n+1}. (Why would using 1^n be a bad idea?) A k-tuple $(n_1, n_2, \ldots, n_k) \in \mathbb{N}$ will be represented by $1^{n_1+1}01^{n_2+1}0 \ldots 01^{n_k+1}$, *i.e.* with the representations of the individual numbers separated by 0s. This scheme is inefficient in its use of space — compared to binary notation, for example — but it is simple and can be implemented on Turing machines restricted to the alphabet $\{1\}$.

Turing computable functions. With suitable conventions for representing the input and output of a function on the natural numbers on the tape of a Turing machine in hand, we can define what it means for a function to be computable by a Turing machine.

Definition 12.1. A k-place function f is *Turing computable*, or just *computable*, if there is a Turing machine M such that for any k-tuple $(n_1, \ldots, n_k) \in \mathrm{dom}(f)$ the computation of M with input tape $\underline{0}1^{n_1+1}01^{n_2+1} \ldots 01^{n_k+1}$ eventually halts with output tape $\underline{0}1^{f(n_1,\ldots,n_k)+1}$. Such a machine M is said to *compute* f.

Note that for a Turing machine M to compute a function f, M need only do the right thing on the right kind of input: what M does in other situations does not matter. In particular, it does not matter what M might do with k-tuple which is not in the domain of f.

Example 12.1. The identity function $i_{\mathbb{N}} \colon \mathbb{N} \to \mathbb{N}$, *i.e.* $i_{\mathbb{N}}(n) = n$, is computable. It is computed by $M = \emptyset$, the Turing machine with an empty table that does absolutely nothing on any input.

Example 12.2. The projection function $\pi^2_1 : \mathbb{N}^2 \to \mathbb{N}$ given by $\pi^2_1(n, m) = n$ is computed by the Turing machine:

P^2_1	0	1
1	$0R2$	
2	$0R3$	$1R2$
3	$0L4$	$0R3$
4	$0L4$	$1L5$
5		$1L5$

P_1^2 acts as follows: it moves to the right past the first block of 1s without disturbing it, erases the second block of 1s, and then returns to the left of first block and halts.

The projection function $\pi_2^2 : \mathbb{N}^2 \to \mathbb{N}$ given by $\pi_2^2(n, m) = m$ is also computable: the Turing machine P of Example 10.5 does the job.

PROBLEM 12.1. *Find Turing machines that compute the following functions and explain how they work.*

(1) $\mathrm{O}(n) = 0$.
(2) $\mathrm{S}(n) = n + 1$.
(3) $\textsc{Sum}(n, m) = n + m$.
(4) $\textsc{Pred}(n) = \begin{cases} n-1 & n \geq 1 \\ 0 & n = 0 \end{cases}$.
(5) $\textsc{Diff}(n, m) = \begin{cases} n-m & n \geq m \\ 0 & n < m \end{cases}$.
(6) $\pi_2^3(p, q, r) = q$.
(7) $\pi_i^k(a_1, \ldots, a_i, \ldots, a_k) = a_i$

We will consider methods for building functions computable by Turing machines out of simpler ones later on.

A non-computable function. In the meantime, it is worth asking whether or not every function on the natural numbers is computable. No such luck!

PROBLEM 12.2. *Show that there is some* 1*-place function* $f : \mathbb{N} \to \mathbb{N}$ *which is not computable by comparing the number of such functions to the number of Turing machines.*

The argument hinted at above is unsatisfying in that it tells us there is a non-computable function without actually producing an explicit example. We can have some fun on the way to one.

DEFINITION 12.2 (Busy Beaver Competition). A machine M is an *n-state entry in the busy beaver competition* if:

- M has a two-way infinite tape and alphabet $\{1\}$ (see Chapter 11;
- M has $n+1$ states, but state $n+1$ is used only for halting (so both $M(n+1, 0)$ and $M(n+1, 1)$ are undefined);
- M eventually halts when given a blank input tape.

M's *score* in the competition is the number of 1's on the output tape of its computation from a blank input tape. The greatest possible score of an n-state entry in the competition is denoted by $\Sigma(n)$.

Note that there are only finitely many possible n-state entries in the busy beaver competition because there are only finitely many $(n+1)$-state Turing machines with alphabet $\{1\}$. Since there is at least one n-state entry in the busy beaver competition for every $n \geq 0$, it follows that $\Sigma(n)$ is well-defined for each $n \in \mathbb{N}$.

EXAMPLE 12.3. $M = \emptyset$ is the *only* 0-state entry in the busy beaver competition, so $\Sigma(0) = 0$.

EXAMPLE 12.4. The machine P given by

P	0	1
1	$1R2$	$1L2$
2	$1L1$	$1L3$

is a 2-state entry in the busy beaver competition with a score of 4, so $\Sigma(2) \geq 4$.

The function Σ grows extremely quickly. It is known that $\Sigma(0) = 0$, $\Sigma(1) = 1$, $\Sigma(2) = 4$, $\Sigma(3) = 6$, and $\Sigma(4) = 13$. The value of $\Sigma(5)$ is still unknown, but must be quite large.[1]

PROBLEM 12.3. *Show that:*

(1) *The 2-state entry given in Example 12.4 actually scores* 4.
(2) $\Sigma(1) = 1$.
(3) $\Sigma(3) \geq 6$.
(4) $\Sigma(n) < \Sigma(n+1)$ *for every* $n \in \mathbb{N}$.

PROBLEM 12.4. *Devise as high-scoring* 4- *and* 5*-state entries in the busy beaver competition as you can.*

The serious point of the busy beaver competition is that the function Σ is *not* a Turing computable function.

PROPOSITION 12.5. Σ *is not computable by any Turing machine.*

Anyone interested in learning more about the busy beaver competition should start by reading the paper [**16**] in which it was first introduced.

Building more computable functions. One of the most common methods for assembling functions from simpler ones in many parts of mathematics is composition. It turns out that compositions of computable functions are computable.

[1]The best score known to the author by a 5-state entry in the busy beaver competition is 4098. One of the two machines achieving this score does so in a computation that takes over 40 million steps! The other requires only 11 million or so...

DEFINITION 12.3. Suppose that $m, k \geq 1$, g is an m-place function, and $h_1, \ldots, h_m$ are k-place functions. Then the k-place function f is said to be obtained from $g, h_1, \ldots, h_m$ by *composition*, written as

$$f = g \circ (h_1, \ldots, h_m),$$

if for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$,

$$f(n_1, \ldots, n_k) = g(h_1(n_1, \ldots, n_k), \ldots, h_m(n_1, \ldots, n_k)).$$

EXAMPLE 12.5. The constant function c_1^1, where $c_1^1(n) = 1$ for all n, can be obtained by composition from the functions S and O. For any $n \in \mathbb{N}$,

$$c_1^1(n) = (\mathrm{S} \circ \mathrm{O})(n) = \mathrm{S}(\mathrm{O}(n)) = \mathrm{S}(0) = 0 + 1 = 1.$$

PROBLEM 12.6. *Suppose $k \geq 1$ and $a \in \mathbb{N}$. Use composition to define the constant function c_a^k, where $c_a^k(n_1, \ldots, n_k) = a$ for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$, from functions already known to be computable.*

PROPOSITION 12.7. *Suppose that $1 \leq k$, $1 \leq m$, g is a Turing computable m-place function, and $h_1, \ldots, h_m$ are Turing computable k-place functions. Then $g \circ (h_1, \ldots, h_m)$ is also Turing computable.*

Starting with a small set of computable functions, and applying computable ways (such as composition) of building functions from simpler ones, we will build up a useful collection of computable functions. This will also provide a characterization of computable functions which does not mention any type of computing device.

The "small set of computable functions" that will be the fundamental building blocks is infinite only because it includes all the projection functions.

DEFINITION 12.4. The following are the *initial functions*:

- O, the 1-place function such that $\mathrm{O}(n) = 0$ for all $n \in \mathbb{N}$;
- S, the 1-place function such that $\mathrm{S}(n) = n + 1$ for all $n \in \mathbb{N}$; and,
- for each $k \geq 1$ and $1 \leq i \leq k$, π_i^k, the k-place function such that $\pi_i^k(n_1, \ldots, n_k) = n_i$ for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$.

O is often referred to as the *zero function*, S is the *successor function*, and the functions π_i^k are called the *projection functions*.

Note that π_1^1 is just the identity function on $\mathbb{N}$.

We have already shown, in Problem 12.1, that all the initial functions are computable. It follows from Proposition 12.7 that every function defined from the initial functions using composition (any number of times) is computable too. Since one can build relatively few functions from the initial functions using only composition...

PROPOSITION 12.8. *Suppose f is a 1-place function obtained from the initial functions by finitely many applications of composition. Then there is a constant $c \in \mathbb{N}$ such that $f(n) \leq n + c$ for all $n \in \mathbb{N}$.*

. . . in the next chapter we will add other methods of building functions to our repertoire that will allow us to build all computable functions from the initial functions.

CHAPTER 13

Recursive Functions

We will add two other methods of building computable functions from computable functions to composition, and show that one can use the three methods to construct all computable functions on $\mathbb{N}$ from the initial functions.

Primitive recursion. The second of our methods is simply called recursion in most parts of mathematics and computer science. Historically, the term "primitive recursion" has been used to distinguish it from the other recursive method of defining functions that we will consider, namely unbounded minimalization. ... Primitive recursion boils down to defining a function inductively, using different functions to tell us what to do at the base and inductive steps. Together with composition, it suffices to build up just about all familiar arithmetic functions from the initial functions.

DEFINITION 13.1. Suppose that $k \geq 1$, g is a k-place function, and h is a $k+2$-place function. Let f be the $(k+1)$-place function such that

(1) $f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k)$ and
(2) $f(n_1, \ldots, n_k, m+1) = h\left(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)\right)$

for every $(n_1, \ldots, n_k) \in \mathbb{N}^k$ and $m \in \mathbb{N}$. Then f is said to be obtained from g and h by *primitive recursion*.

That is, the initial values of f are given by g, and the rest are given by h operating on the given input and the preceding value of f.

For a start, primitive recursion and composition let us define addition and multiplication from the initial functions.

EXAMPLE 13.1. $\textsc{Sum}(n, m) = n + m$ is obtained by primitive recursion from the initial function π^1_1 and the composition $\mathrm{S} \circ \pi^3_3$ of initial functions as follows:

- $\textsc{Sum}(n, 0) = \pi^1_1(n)$;
- $\textsc{Sum}(n, m+1) = (\mathrm{S} \circ \pi^3_3)(n, m, \textsc{Sum}(n, m))$.

To see that this works, one can proceed by induction on m:

At the base step, $m = 0$, we have

$$\textsc{Sum}(n, 0) = \pi^1_1(n) = n = n + 0\,.$$

Assume that $m \geq 0$ and $\textsc{Sum}(n, m) = n + m$. Then

$$\begin{aligned}\textsc{Sum}(n, m+1) &= (\mathrm{S} \circ \pi^3_3)(n, m, \textsc{Sum}(n, m)) \\ &= \mathrm{S}(\pi^3_3(n, m, \textsc{Sum}(n, m))) \\ &= \mathrm{S}(\textsc{Sum}(n, m)) \\ &= \textsc{Sum}(n, m) + 1 \\ &= n + m + 1\,,\end{aligned}$$

as desired.

As addition is to the successor function, so multiplication is to addition.

Example 13.2. $\textsc{Mult}(n, m) = nm$ is obtained by primitive recursion from O and $\textsc{Sum} \circ (\pi^3_3, \pi^3_1)$:

- $\textsc{Mult}(n, 0) = \mathrm{O}(n)$;
- $\textsc{Mult}(n, m + 1) = (\textsc{Sum} \circ (\pi^3_3, \pi^3_1))(n, m, \textsc{Mult}(n, m))$.

We leave it to the reader to check that this works.

Problem 13.1. *Use composition and primitive recursion to obtain each of the following functions from the initial functions or other functions already obtained from the initial functions.*

(1) $\textsc{Exp}(n, m) = n^m$
(2) $\textsc{Pred}(n)$ *(defined in Problem 12.1)*
(3) $\textsc{Diff}(n, m)$ *(defined in Problem 12.1)*
(4) $\textsc{Fact}(n) = n!$

Proposition 13.2. *Suppose $k \geq 1$, g is a Turing computable k-place function, and h is a Turing computable $(k+2)$-place function. If f is obtained from g and h by primitive recursion, then f is also Turing computable.*

Primitive recursive functions and relations. The collection of functions which can be obtained from the initial functions by (possibly repeatedly) using composition and primitive recursion is useful enough to have a name.

Definition 13.2. A function f is *primitive recursive* if it can be defined from the initial functions by finitely many applications of the operations of composition and primitive recursion.

So we already know that all the initial functions, addition, and multiplication, among others, are primitive recursive.

PROBLEM 13.3. *Show that each of the following functions is primitive recursive.*

(1) *For any $k \geq 0$ and primitive recursive $(k+1)$-place function g, the $(k+1)$-place function f given by*

$$\begin{aligned} f(n_1, \ldots, n_k, m) &= \Pi_{i=0}^{m} g(n_1, \ldots, n_k, i) \\ &= g(n_1, \ldots, n_k, 0) \cdot \ldots \cdot g(n_1, \ldots, n_k, m)\,. \end{aligned}$$

(2) *For any constant $a \in \mathbb{N}$, $\chi_{\{a\}}(n) = \begin{cases} 0 & n \neq a \\ 1 & n = a\,. \end{cases}$*

(3) $h(n_1, \ldots, n_k) = \begin{cases} f(n_1, \ldots, n_k) & (n_1, \ldots, n_k) \neq (c_1, \ldots, c_k) \\ a & (n_1, \ldots, n_k) = (c_1, \ldots, c_k) \end{cases}$, *if f is a primitive recursive k-place function and $a, c_1, \ldots, c_k \in \mathbb{N}$ are constants.*

THEOREM 13.4. *Every primitive recursive function is Turing computable.*

Be warned, however, that there are computable functions which are not primitive recursive.

We can extend the idea of "primitive recursive" to relations by using their characteristic functions.

DEFINITION 13.3. Suppose $k \geq 1$. A k-place relation $P \subseteq \mathbb{N}^k$ is *primitive recursive* if its characteristic function

$$\chi_P(n_1, \ldots, n_k) = \begin{cases} 1 & (n_1, \ldots, n_k) \in P \\ 0 & (n_1, \ldots, n_k) \notin P \end{cases}$$

is primitive recursive.

EXAMPLE 13.3. $P = \{2\} \subset \mathbb{N}$ is primitive recursive since $\chi_{\{2\}}$ is recursive by Problem 13.3.

PROBLEM 13.5. *Show that the following relations and functions are primitive recursive.*

(1) *$\neg P$,* i.e. *$\mathbb{N}^k \setminus P$, if P is a primitive recursive k-place relation.*
(2) *$P \vee Q$,* i.e. *$P \cup Q$, if P and Q are primitive recursive k-place relations.*
(3) *$P \wedge Q$,* i.e. *$P \cap Q$, if P and Q are primitive recursive k-place relations.*
(4) EQUAL, *where* $\text{EQUAL}(n, m) \iff n = m$.
(5) *$h(n_1, \ldots, n_k, m) = \sum_{i=0}^{m} g(n_1, \ldots, n_k, i)$, for any $k \geq 0$ and primitive recursive $(k+1)$-place function g.*
(6) DIV, *where* $\text{DIV}(n, m) \iff n \mid m$.

(7) IsPrime, *where* $\text{IsPrime}(n) \iff n$ *is prime.*
(8) $\text{Prime}(k) = p_k$, *where* $p_0 = 1$ *and* p_k *is the* k*th prime if* $k \geq 1$.
(9) $\text{Power}(n, m) = k$, *where* $k \geq 0$ *is maximal such that* $n^k \mid m$.
(10) $\text{Length}(n) = \ell$, *where* ℓ *is maximal such that* $p_\ell \mid n$.
(11) $\text{Element}(n, i) = n_i$, *if* $n = p_1^{n_1} \dots p_k^{n_k}$ *(and* $n_i = 0$ *if* $i > k$*).*
(12) $\text{Subseq}(n, i, j) = \begin{cases} p_i^{n_i} p_{i+1}^{n_{i+1}} \dots p_j^{n_j} & \text{if } 1 \leq i \leq j \leq k \\ 0 & \text{otherwise} \end{cases}$, *whenever* $n = p_1^{n_1} \dots p_k^{n_k}$.
(13) $\text{Concat}(n, m) = p_1^{n_1} \dots p_k^{n_k} p_{k+1}^{m_1} \dots p_{k+\ell}^{m_l}$, *if* $n = p_1^{n_1} \dots p_k^{n_k}$ *and* $m = p_1^{m_1} \dots p_\ell^{m_\ell}$.

Parts of Problem 13.5 give us tools for representing finite sequences of integers by single integers, as well as some tools for manipulating these representations. This lets us reduce, in principle, all problems involving primitive recursive functions and relations to problems involving only 1-place primitive recursive functions and relations.

THEOREM 13.6. *A* k*-place* g *is primitive recursive if and only if the* 1*-place function* h *given by* $h(n) = g(n_1, \dots, n_k)$ *if* $n = p_1^{n_1} \dots p_k^{n_k}$ *is primitive recursive.*

NOTE. It doesn't matter what the function h may do on an n which does not represent a sequence of length k.

COROLLARY 13.7. *A* k*-place relation* P *is primitive recursive if and only if the* 1*-place relation* P' *is primitive recursive, where*

$$(n_1, \dots, n_k) \in P \iff p_1^{n_1} \dots p_k^{n_k} \in P'.$$

A computable but not primitive recursive function. While primitive recursion and composition do not quite suffice to build all Turing computable functions from the initial functions, they are powerful enough that specific counterexamples are not all that easy to find.

EXAMPLE 13.4 (Ackerman's Function). Define the 2-place function A from as follows:

- $\text{A}(0, \ell) = \text{S}(\ell)$
- $\text{A}(\text{S}(k), 0) = \text{A}(k, 1)$
- $\text{A}(\text{S}(k), \text{S}(\ell)) = \text{A}(k, \text{A}(\text{S}(k), \ell))$

Given A, define the 1-place function α by $\alpha(n) = \text{A}(n, n)$.

It isn't too hard to show that A, and hence also α, are Turing computable. However, though it takes considerable effort to prove it, α grows faster with n than any primitive recursive function. (Try working out the first few values of $\alpha \dots$)

PROBLEM 13.8. *Show that the functions* A *and* α *defined in Example 13.4 are Turing computable.*

If you are very ambitious, you can try to prove the following theorem.

THEOREM 13.9. *Suppose* α *is the function defined in Example 13.4 and* f *is any primitive recursive function. Then there is an* $n \in \mathbb{N}$ *such that for all* $k > n$, $\alpha(k) > f(k)$.

COROLLARY 13.10. *The function* α *defined in Example 13.4 is not primitive recursive.*

. . . but if you aren't, you can still try the following exercise.

PROBLEM 13.11. *Informally, define a computable function which must be different from every primitive recursive function.*

Unbounded minimalization. The last of our three method of building computable functions from computable functions is unbounded minimalization. The functions which can be defined from the initial functions using unbounded minimalization, as well as composition and primitive recursion, turn out to be precisely the Turing computable functions.

Unbounded minimalization is the counterpart for functions of "brute force" algorithms that try every possibility until they succeed. (Which, of course, they might not. . .)

DEFINITION 13.4. Suppose $k \geq 1$ and g is a $(k+1)$-place function. Then the *unbounded minimalization* of g is the k-place function f defined by

$f(n_1, \ldots, n_k) = m$ where m is least so that $g(n_1, \ldots, n_k, m) = 0$.

This is often written as $f(n_1, \ldots, n_k) = \mu m[g(n_1, \ldots, n_k, m) = 0]$.

NOTE. If there is no m such that $g(n_1, \ldots, n_k, m) = 0$, then the unbounded minimalization of g is not defined on $(n_1, \ldots, n_k)$. This is one reason we will occasionally need to deal with partial functions.

If the unbounded minimalization of a computable function is to be computable, we have a problem even if we ask for some default output (0, say) to ensure that it is defined for all k-tuples. The obvious procedure which tests successive values of g to find the needed m will run forever if there is no such m, and the incomputability of the Halting Problem suggests that other procedure's won't necessarily succeed either. It follows that it is desirable to be careful, so far as possible, which functions unbounded minimalization is applied to.

DEFINITION 13.5. A $(k+1)$-place function g is said to be *regular* if for every $(n_1, \ldots, n_k) \in \mathbb{N}^k$, there is at least one $m \in \mathbb{N}$ so that $g(n_1, \ldots, n_k, m) = 0$.

That is, g is regular precisely if the obvious strategy of computing $g(n_1, \ldots, n_k, m)$ for $m = 0, 1, \ldots$ in succession until an m is found with $g(n_1, \ldots, n_k, m) = 0$ always succeeds.

PROPOSITION 13.12. *If g is a Turing computable regular $(k+1)$-place function, then the unbounded minimalization of g is also Turing computable.*

While unbounded minimalization adds something essentially new to our repertoire, it is worth noticing that *bounded minimalization* does not.

PROBLEM 13.13. *Suppose g is a $(k+1)$-place primitive recursive regular function such that for some primitive recursive k-place function h,*

$$\mu m[g(n_1, \ldots, n_k, m) = 0] \leq h(n_1, \ldots, n_k)$$

for all $(n_1, \ldots, n_k) \in \mathbb{N}$. Show that $\mu m[g(n_1, \ldots, n_k, m) = 0]$ is also primitive recursive.

Recursive functions and relations. We can finally define an equivalent notion of computability for functions on the natural numbers which makes no mention of any computational device.

DEFINITION 13.6. A k-place function f is *recursive* if it can be defined from the initial functions by finitely many applications of composition, primitive recursion, and the unbounded minimalization of regular functions.

Similarly, k-place partial function is *recursive* if it can be defined from the initial functions by finitely many applications of composition, primitive recursion, and the unbounded minimalization of (possibly non-regular) functions.

In particular, every primitive recursive function is a recursive function.

THEOREM 13.14. *Every recursive function is Turing computable.*

We shall show that every Turing computable function is recursive later on. Similarly to primitive recursive relations we have the following.

DEFINITION 13.7. A k-place relation P is said to be *recursive* (*Turing computable*) if its characteristic function χ_P is recursive (Turing computable).

Since every recursive function is Turing computable, and *vice versa*, "recursive" is just a synonym of "Turing computable", for functions and relations alike.

Also, similarly to Theorem 13.6 and Corollary 13.7 we have the following.

THEOREM 13.15. *A k-place function g is recursive if and only if the 1-place function h given by $h(n) = g(n_1, \ldots, n_k)$ if $n = p_1^{n_1} \ldots p_k^{n_k}$ is recursive.*

As before, it doesn't really matter what the function h does on an n which does not represent a sequence of length k.

COROLLARY 13.16. *A k-place relation P is recursive if and only if the 1-place relation P' is recursive, where*

$$(n_1, \ldots, n_k) \in P \iff p_1^{n_1} \ldots p_k^{n_k} \in P' .$$

CHAPTER 14

Characterizing Computability

By putting together some of the ideas in Chapters 12 and 13, we can use recursive functions to simulate Turing machines. This will let us show that Turing computable functions are recursive, completing the argument that Turing machines and recursive functions are essentially equivalent models of computation. We will also use these techniques to construct an *universal Turing machine* (or *UTM*): a machine U that, when given as input (a suitable description of) some Turing machine M and an input tape $\mathbf{a}$ for M, simulates the computation of M on input $\mathbf{a}$. In effect, an universal Turing machine is a single piece of hardware that lets us treat other Turing machines as software.

Turing computable functions are recursive. Our basic strategy is to show that any Turing machine can be simulated by some recursive function. Since recursive functions operate on integers, we will need to encode the tape positions of Turing machines, as well as Turing machines themselves, by integers. For simplicity, we shall stick to Turing machines with alphabet $\{1\}$; we already know from Chapter 11 that such machines can simulate Turing machines with bigger alphabets.

Definition 14.1. Suppose $(s, i, \mathbf{a})$ is a tape position such that all but finitely many cells of $\mathbf{a}$ are blank. Let n be any positive integer such that $a_k = 0$ for all $k > n$. Then the *code* of $(s, i, \mathbf{a})$ is

$$\ulcorner (s, i, \mathbf{a}) \urcorner = 2^s 3^i 5^{a_0} 7^{a_1} 11^{a_2} \dots p_{n+3}^{a_n} .$$

Example 14.1. Consider the tape position $(2, 1, 1001)$. Then

$$\ulcorner (2, 1, 1001) \urcorner = 2^2 3^1 5^1 7^0 11^0 13^1 = 780 .$$

Problem 14.1. *Find the codes of the following tape positions.*

(1) $(1, 0, \mathbf{a})$, *where* $\mathbf{a}$ *is entirely blank.*
(2) $(4, 3, \mathbf{a})$, *where* $\mathbf{a}$ *is* 1011100101.

Problem 14.2. *What is the tape position whose code is* 10314720*?*

When dealing with computations, we will also need to encode sequences of tape positions by integers.

DEFINITION 14.2. Suppose $t_1 t_2 \dots t_n$ is a sequence of tape positions. Then the *code* of this sequence is

$$\ulcorner t_1 t_2 \dots t_n \urcorner = 2^{\ulcorner t_1 \urcorner} 3^{\ulcorner t_2 \urcorner} \dots p_n^{\ulcorner t_n \urcorner}.$$

NOTE. Both tape positions and sequences of tape positions have unique codes.

PROBLEM 14.3. *Pick some (short!) sequence of tape positions and find its code.*

Having defined how to represent tape positions as integers, we now need to manipulate these representations using recursive functions. The recursive functions and relations in Problems 13.3 and 13.5 provide most of the necessary tools.

PROBLEM 14.4. *Show that both of the following relations are primitive recursive.*

(1) TAPEPOS, *where* $\text{TAPEPOS}(n) \iff$ *n is the code of a tape position.*
(2) TAPEPOSSEQ, *where* $\text{TAPEPOSSEQ}(n) \iff$ *n is the code of a sequence of tape positions.*

PROBLEM 14.5. *Show that each of the following is primitive recursive.*

(1) *The 4-place function* ENTRY *such that*

$$\text{ENTRY}(j, w, t, n) = \begin{cases} \ulcorner (t, i+w-1, \mathbf{a}') \urcorner & \text{if } n = \ulcorner (s, i, \mathbf{a}) \urcorner,\ j \in \{0,1\}, \\ & w \in \{0,2\},\ i+w-1 \geq 0, \text{ and } t \geq 1, \\ & \text{where } a'_k = a_k \text{ for } k \neq i \text{ and } a'_i = j; \\ 0 & \text{otherwise.} \end{cases}$$

(2) *For any Turing machine M with alphabet $\{1\}$, the 1-place function* STEP_M *such that*

$$\text{STEP}_M(n) = \begin{cases} \ulcorner \mathbf{M}(s, i, \mathbf{a}) \urcorner & \text{if } n = \ulcorner (s, i, \mathbf{a}) \urcorner \text{ and} \\ & \mathbf{M}(s, i, \mathbf{a}) \text{ is defined;} \\ 0 & \text{otherwise.} \end{cases}$$

(3) *For any Turing machine M with alphabet $\{1\}$, the 1-place relation* COMP_M, *where*

$$\text{COMP}_M(n) \iff \text{n is the code of a computation of } M.$$

The functions and relations above may be primitive recursive, but the last big step in showing that Turing computable functions are recursive requires unbounded minimalization.

PROPOSITION 14.6. *For any Turing machine M with alphabet $\{1\}$, the 1-place (partial) function* SIM_M *is recursive, where*

$$\text{SIM}_M(n) = \ulcorner(t, j, \mathbf{b})\urcorner$$

if $n = \ulcorner(1, 0, \mathbf{a})\urcorner$ for some input tape $\mathbf{a}$ and M eventually halts in position $(t, j, \mathbf{b})$ on input $\mathbf{a}$. (Note that $\text{SIM}_M(n)$ may be undefined if $n \neq \ulcorner(1, 0, \mathbf{a})\urcorner$ for an input tape $\mathbf{a}$, or if M does not eventually halt on input $\mathbf{a}$.)

LEMMA 14.7. *Show that the following functions are primitive recursive:*

(1) *For any fixed $k \geq 1$,* $\text{CODE}_k(n_1, \ldots, n_k) = \ulcorner(1, 0, 01^{n_1}0 \ldots 01^{n_k})\urcorner$.
(2) $\text{DECODE}(t) = n$ *if* $t = \ulcorner(s, i, 01^{n+1})\urcorner$ *(and anything you like otherwise).*

THEOREM 14.8. *Any k-place Turing computable function is recursive.*

COROLLARY 14.9. *A function $f : \mathbb{N}^k \to \mathbb{N}$ is Turing computable if and only if it is recursive.*

Thus Turing machines and recursive functions are essentially equivalent models of computation.

An universal Turing machine. One can push the techniques used above little farther to get a recursive function that can simulate *any* Turing machine. Since every recursive function can be computed by some Turing machine, this effectively gives us an universal Turing machine.

PROBLEM 14.10. *Devise a suitable definition for the code $\ulcorner M\urcorner$ of a Turing machine M with alphabet $\{1\}$.*

PROBLEM 14.11. *Show, using your definition of $\ulcorner M\urcorner$ from Problem 14.10, that the following are primitive recursive.*

(1) *The 2-place function* STEP, *where*

$$\text{STEP}(m, n) = \begin{cases} \ulcorner\mathbf{M}(s, i, \mathbf{a})\urcorner & \text{if } m = \ulcorner M\urcorner \text{ for some machine } M, \\ & n = \ulcorner(s, i, \mathbf{a})\urcorner, \text{ \& } \mathbf{M}(s, i, \mathbf{a}) \text{ is defined;} \\ 0 & \text{otherwise.} \end{cases}$$

(2) *The 2-place relation* COMP*, where*

$$\text{COMP}(m,n) \iff m = \ulcorner M \urcorner$$

for some Turing machine M and n is the code of a computation of M.

PROPOSITION 14.12. *The 2-place (partial) function* SIM *is recursive, where, for any Turing machine M with alphabet $\{1\}$ and input tape $\mathbf{a}$ for M,*

$$\text{SIM}(\ulcorner M \urcorner, \ulcorner (1,0,\mathbf{a}) \urcorner) = \ulcorner (t,j,\mathbf{b}) \urcorner$$

if M eventually halts in position $(t,j,\mathbf{b})$ on input $\mathbf{a}$. (Note that $\text{SIM}(m,n)$ may be undefined if m is not the code of some Turing machine M, or if $n \neq \ulcorner (1,0,\mathbf{a}) \urcorner$ for an input tape $\mathbf{a}$, or if M does not eventually halt on input $\mathbf{a}$.)

COROLLARY 14.13. *There is a Turing machine U which can simulate any Turing machine M.*

COROLLARY 14.14. *There is a recursive function f which can compute any other recursive function.*

The Halting Problem. An effective method to determine whether or not a given machine will eventually halt on a given input — short of waiting forever! — would be nice to have. For example, assuming Church's Thesis is true, such a method would let us identify computer programs which have infinite loops before we attempt to execute them.

THE HALTING PROBLEM. Given a Turing machine M and an input tape $\mathbf{a}$, is there an effective method to determine whether or not M eventually halts on input $\mathbf{a}$?

Given that we are using Turing machines to formalize the notion of an effective method, one of the difficulties with solving the Halting Problem is representing a given Turing machine and its input tape as input for another machine. As this is one of the things that was accomplished in the course of constructing an universal Turing machine, we can now formulate a precise version of the Halting Problem and solve it.

THE HALTING PROBLEM. Is there a Turing machine T which, for any Turing machine M with alphabet $\{1\}$ and tape $\mathbf{a}$ for M, halts on input

$$\underline{0}1^{\ulcorner M \urcorner + 1} 0 1^{\ulcorner (1,0,\mathbf{a}) \urcorner + 1}$$

with output $\underline{0}11$ if M halts on input $\mathbf{a}$, and with output $\underline{0}1$ if M does not halt on input $\mathbf{a}$?

Note that this precise version of the Halting Problem is equivalent to the informal one above only if Church's Thesis is true.

PROBLEM 14.15. *Show that there is a Turing machine C which, for any Turing machine M with alphabet $\{1\}$, on input*

$$\underline{0}1^{\ulcorner M \urcorner + 1}$$

eventually halts with output

$$\underline{0}1^{\ulcorner M \urcorner + 1} 0 1^{\ulcorner (0,1,0 1^{\ulcorner M \urcorner + 1}) \urcorner + 1}$$

THEOREM 14.16. *The answer to (the precise version of) the Halting Problem is "No."*

Recursively enumerable sets. The following notion is of particular interest in the advanced study of computability.

DEFINITION 14.3. A subset (*i.e.* a 1-place relation) P of $\mathbb{N}$ is *recursively enumerable*, often abbreviated as *r.e.*, if there is a 1-place recursive function f such that $P = \mathrm{im}(f) = \{\, f(n) \mid n \in \mathbb{N} \,\}$.

Since the image of any recursive 1-place function is recursively enumerable by definition, we do not lack for examples. For one, the set E of even natural numbers is recursively enumerable, since it is the image of $f(n) = \textsc{Mult}(\mathrm{S}(\mathrm{S}(\mathrm{O}(n))), n)$.

PROPOSITION 14.17. *If P is a 1-place recursive relation, then P is recursively enumerable.*

This proposition is not reversible, but it does come close.

PROPOSITION 14.18. *$P \subseteq \mathbb{N}$ is recursive if and only if both P and $\mathbb{N} \setminus P$ are recursively enumerable.*

PROBLEM 14.19. *Find an example of a recursively enumerable set which is not recursive.*

PROBLEM 14.20. *Is $P \subseteq \mathbb{N}$ primitive recursive if and only if both P and $\mathbb{N} \setminus P$ are enumerable by primitive recursive functions?*

PROBLEM 14.21. *$P \subseteq \mathbb{N}$ recursively enumerable if and only if there is a 1-place recursive partial function g such that $P = dom(g) = \{\, n \mid g(n)$ is defined $\}$*

Hints for Chapters 10–14

Hints for Chapter 10.

10.1. This should be easy...

10.2. Ditto.

10.3. (1) Any machine with the given alphabet and a table with three non-empty rows will do.
(2) Every entry in the table in the 0 column must write a 1 in the scanned cell; similarly, every entry in the 1 column must write a 0 in the scanned cell.
(3) What's the simplest possible table for a given alphabet?

10.4. Unwind the definitions step by step in each case. Not all of these are computations...

10.5. Examine your solutions to the previous problem and, if necessary, take the computations a little farther.

10.6. Have the machine run on forever to the right, writing down the desired pattern as it goes no matter what may be on the tape already.

10.7. Consider your solution to Problem 10.6 for one possible approach. It should be easy to find simpler solutions, though.

10.8. Consider the tasks S and T are intended to perform.

10.9. (1) Use four states to write the 1s, one for each.
(2) The input has a convenient marker.
(3) Run back and forth to move one marker n cells *from* the block of 1's while moving another *through* the block, and then fill in.
(4) Modify the previous machine by having it delete every other 1 after writing out 1^{2n}.
(5) Run back and forth to move the right block of 1s cell by cell to the desired position.
(6) Run back and forth to move the left block of 1s cell by cell past the other two, and then apply a minor modification of the machine in part 5.

(7) Variations on the ideas used in part 6 should do the job.
(8) Run back and forth between the blocks, moving a marker through each. After the race between the markers to the ends of their respective blocks has been decided, erase everything and write down the desired output.

Hints for Chapter 11.

11.1. This ought to be easy.

11.2. Generalize the technique of Example 11.1, adding two new states to help with each old state that may cause a move in different directions. You do have to be a bit careful not to make a machine that would run off the end of the tape when the original would not.

11.3. You only need to change the parts of the definitions involving the symbols 0 and 1.

11.4. If you have trouble figuring out whether the subroutine of Z simulating state 1 of W on input y, try tracing the partial computations of W and Z on other tapes involving y.

11.5. Generalize the concepts used in Example 11.2. Note that the simulation must operate with coded versions of Ms tape, unless $\Sigma = \{1\}$. The key idea is to use the tape of the simulator in blocks of some fixed size, with the patterns of 0s and 1s in each block corresponding to elements of Σ.

11.6. This should be straightforward, if somewhat tedious. You do need to be careful in coming up with the appropriate input tapes for O.

11.7. Generalize the technique of Example 11.3, splitting up the tape of the simulator into upper and lower tracks and splitting each state of N into two states in P. You will need to be quite careful in describing just how the latter is to be done.

11.8. This is mostly pretty easy. The only problem is to devise N so that one can tell from its output whether P halted or crashed, and this is easy to indicate using some extra symbol in Ns alphabet.

11.9. If you're in doubt, go with one read/write scanner for each tape, and have each entry in the table of a two-tape machine take both scanners into account. Simulating such a machine is really just a variation on the techniques used in Example 11.3.

11.10. Such a machine should be able to move its scanner to cells up and down from the current one, as well to the side. (Diagonally too, if you want to!) Simulating such a machine on a single tape machine is a challenge. You might find it easier to first describe how to simulate it on a suitable multiple-tape machine.

Hints for Chapter 12.

12.1.
(1) Delete most of the input.
(2) Add a one to the far end of the input.
(3) Add a little to the input, and delete a little more elsewhere.
(4) Delete a little from the input *most* of the time.
(5) Run back and forth between the two blocks in the input, deleting until one side disappears. Clean up appropriately! (This is a relative of Problem 10.9.8.)
(6) Delete two of blocks and move the remaining one.
(7) This is just a souped-up version of the machine immediately preceding...

12.2. There are just as many functions $\mathbb{N} \to \mathbb{N}$ as there are real numbers, but only as many Turing machines as there are natural numbers.

12.3.
(1) Trace the computation through step-by-step.
(2) Consider the scores of each of the 1-state entries in the busy beaver competition.
(3) Find a 3-state entry in the busy beaver competition which scores six.
(4) Show how to turn an n-state entry in the busy beaver competition into an $(n+1)$-state entry that scores just one better.

12.4. You could start by looking at modifications of the 3-state entry you devised in Problem 12.3.3, but you will probably want to do some serious fiddling to do better than what Problem 12.3.4 do from there.

12.5. Suppose Σ was computable by a Turing machine M. Modify M to get an n-state entry in the busy beaver competition for some n which achieves a score greater than $\Sigma(n)$. The key idea is to add a "pre-processor" to M which writes a block with more 1s than the number odf states that M and the pre-processor have between them.

12.6. Generalize Example 12.5.

12.7. Use machines computing g, h_1, ..., h_m as sub-machines of the machine computing the composition. You might also find sub-machines that copy the original input and various stages of the output useful. It is important that each sub-machine get all the data it needs and does not damage the data needed by other sub-machines.

12.8. Proceed by induction on the number of applications of composition used to define f from the initial functions.

Hints for Chapter 13.

13.1. (1) Exponentiation is to multiplication as multiplication is to addition.
(2) This is straightforward except for taking care of $\textsc{Pred}(0) = \textsc{Pred}(1) = 0$.
(3) Diff is to Pred as S is to Sum.
(4) This is straightforward if you let $0! = 1$.

13.2. Machines used to compute g and h are the principal parts of the machine computing f, along with parts to copy, move, and/or delete data on the tape between stages in the recursive process.

13.3. (1) f is to g as Fact is to the identity function.
(2) Use Diff and a suitable constant function as the basic building blocks.
(3) This is a slight generalization of the preceding part.

13.4. Proceed by induction on the number of applications of primitive recursion and composition.

13.5. (1) Use a composition including Diff, χ_P, and a suitable constant function.
(2) A suitable composition will do the job; it's just a little harder than it looks.
(3) A suitable composition will do the job; it's rather more straightforward than the previous part.
(4) Note that $n = m$ exactly when $n - m = 0 = m - n$.
(5) Adapt your solution from the first part of Problem 13.3.
(6) First devise a characteristic function for the relation

$$\textsc{Product}(n, k, m) \iff nk = m\,,$$

and then sum up.
(7) Use $\chi_{\textsc{Div}}$ and sum up.
(8) Use IsPrime and some ingenuity.
(9) Use Exp and Div and some more ingenuity.
(10) A suitable combination of Prime with other things will do.

(11) A suitable combination of PRIME and POWER will do.
(12) Throw the kitchen sink at this one...
(13) Ditto.

13.6. In each direction, use a composition of functions already known to be primitive recursive to modify the input as necessary.

13.7. A straightforward application of Theorem 13.6.

13.8. This is not unlike, though a little more complicated than, showing that primitive recursion preserves computability.

13.9. It's *not* easy! Look it up...

13.10. This is a very easy consequence of Theorem 13.9.

13.11. Listing the definitions of all possible primitive recursive functions is a computable task. Now borrow a trick from Cantor's proof that the real numbers are uncountable. (A formal argument to this effect could be made using techniques similar to those used to show that all Turing computable functions are recursive in the next chapter.)

13.12. The strategy should be easy. Make sure that at each stage you preserve a copy of the original input for use at later stages.

13.13. The primitive recursive function you define only needs to check values of $g(n_1, \ldots, n_k, m)$ for m such that $0 \leq m \leq h(n_1, \ldots, n_k)$, but it still needs to pick the least m such that $g(n_1, \ldots, n_k, m) = 0$.

13.14. This is very similar to Theorem 13.4.

13.15. This is virtually identical to Theorem 13.6.

13.16. This is virtually identical to Corollary 13.7.

Hints for Chapter 14.

14.1. Emulate Example 14.1 in both parts.

14.2. Write out the prime power expansion of the given number and unwind Definition 14.1.

14.3. Find the codes of each of the positions in the sequence you chose and then apply Definition 14.2.

14.4. (1) $\chi_{\text{TAPEPOS}}(n) = 1$ exactly when the power of 2 in the prime power expansion of n is at least 1 and every other prime appears in the expansion with a power of 0 or 1. This can be achieved with a composition of recursive functions from Problems 13.3 and 13.5.

(2) $\chi_{\text{TapePosSeq}}(n) = 1$ exactly when n is the code of a sequence of tape positions, *i.e.* every power in the prime power expansion of n is the code of a tape position.

14.5. (1) If the input is of the correct form, make the necessary changes to the prime power expansion of n using the tools in Problem 13.5.
(2) Piece Step_M together by cases using the function Entry in each case. The piecing-together works a lot like redefining a function at a particular point in Problem 13.3.
(3) If the input is of the correct form, use the function Step_M to check that the successive elements of the sequence of tape positions are correct.

14.6. The key idea is to use unbounded minimalization on χ_{Comp}, with some additions to make sure the computation found (if any) starts with the given input, and then to extract the output from the code of the computation.

14.7. (1) To define Code_k, consider what $\ulcorner(1, 0, 01^{n_1}0 \dots 01^{n_k})\urcorner$ is as a prime power expansion, and arrange a suitable composition to obrtain it from $(n_1, \dots, n_k)$.
(2) To define Decode you only need to count how many powers of primes other than 3 in the prime-power expansion of $\ulcorner(s, i, 01^{n+1})\urcorner$ are equal to 1.

14.8. Use Proposition 14.6 and Lemma 14.7.

14.9. This follows directly from Theorems 13.14 and 14.8.

14.10. Take some creative inspiration from Definitions 14.1 and 14.2. For example, if $(s, i) \in \text{dom}(M)$ and $M(s, i) = (j, d, t)$, you could let the code of $M(s, i)$ be

$$\ulcorner M(s, i)\urcorner = 2^s 3^i 5^j 7^{d+1} 11^t .$$

14.11. Much of what you need for both parts is just what was needed for Problem 14.5, except that Step is probably easier to define than Step_M was. (Define it as a composition...) The additional ingredients mainly have to do with using $m = \ulcorner M\urcorner$ properly.

14.12. Essentially, this is to Problem 14.11 as proving Proposition 14.6 is to Problem 14.5.

14.13. The machine that computes SIM does the job.

14.14. A modification of SIM does the job. The modifications are needed to handle appropriate input and output. Check Theorem 13.15 for some ideas on what may be appropriate.

14.15. This can be done directly, but may be easier to think of in terms of recursive functions.

14.16. Suppose the answer was yes and such a machine T did exist. Create a machine U as follows. Give T the machine C from Problem 14.15 as a pre-processor and alter its behaviour by having it run forever if M halts and halt if M runs forever. What will T do when it gets itself as input?

14.17. Use χ_P to help define a function f such that $\mathrm{im}(f) = P$.

14.18. One direction is an easy application of Proposition 14.17. For the other, given an $n \in \mathbb{N}$, run the functions enumerating P and $\mathbb{N} \setminus P$ concurrently until one or the other outputs n.

14.19. Consider the set of natural numbers coding (according to some scheme you must devise) Turing machines together with input tapes on which they halt.

14.20. See how far you can adapt your argument for Proposition 14.18.

14.21. This may well be easier to think of in terms of Turing machines. Run a Turing machine that computes g for a few steps on the first possible input, a few on the second, a few more on the first, a few more on the second, a few on the third, a few more on the first, ...

Part IV

Incompleteness

CHAPTER 15

Preliminaries

It was mentioned in the Introduction that one of the motivations for the development of notions of computability was the following question.

ENTSCHEIDUNGSPROBLEM. Given a reasonable set Σ of formulas of a first-order language $\mathcal{L}$ and a formula φ of $\mathcal{L}$, is there an effective method for determining whether or not $\Sigma \vdash \varphi$? □

Armed with knowledge of first-order logic on the one hand and of computability on the other, we are in a position to formulate this question precisely and then solve it. To cut to the chase, the answer is usually "no". Gödel's Incompleteness Theorem asserts, roughly, that given any set of axioms in a first-order language which are computable and also powerful enough to prove certain facts about arithmetic, it is possible to formulate statements in the language whose truth is not decided by the axioms. In particular, it turns out that no consistent set of axioms can hope to prove its own consistency.

We will tackle the Incompleteness Theorem in three stages. First, we will code the formulas and proofs of a first-order language as numbers and show that the functions and relations involved are recursive. This will, in particular, make it possible for us to define a "computable set of axioms" precisely. Second, we will show that all recursive functions and relations can be defined by first-order formulas in the presence of a fairly minimal set of axioms about elementary number theory. Finally, by putting recursive functions talking about first-order formulas together with first-order formulas defining recursive functions, we will manufacture a self-referential sentence which asserts its own unprovability.

NOTE. It will be assumed in what follows that you are familiar with the basics of the syntax and semantics of first-order languages, as laid out in Chapters 5–8 of this text. Even if you are already familiar with the material, you may wish to look over Chapters 5–8 to familiarize yourself with the notation, definitions, and conventions used here, or at least keep them handy in case you need to check some such point.

A language for first-order number theory. To keep things as concrete as possible we will work with and in the following language for first-order number theory, mentioned in Example 5.2.

DEFINITION 15.1. $\mathcal{L}_N$ is the first-order language with the following symbols:

(1) Parentheses: (and)
(2) Connectives: $\neg$ and $\rightarrow$
(3) Quantifier: $\forall$
(4) Equality: $=$
(5) Variable symbols: v_0, v_2, v_3, ...
(6) Constant symbol: 0
(7) 1-place function symbol: S
(8) 2-place function symbols: $+$, $\cdot$, and E.

The non-logical symbols of $\mathcal{L}_N$, 0, S, $+$, $\cdot$, and E, are intended to name, respectively, the number zero, and the successor, addition, multiplication, and exponentiation functions on the natural numbers. That is, the (standard!) structure this language is intended to discuss is $\mathfrak{N} = (\mathbb{N}, 0, \mathrm{S}, +, \cdot, \mathrm{E})$.

Completeness. The notion of completeness used in the Incompleteness Theorem is different from the one used in the Completeness Theorem.[1] "Completeness" in the latter sense is a property of a logic: it asserts that whenever $\Gamma \models \sigma$ (*i.e.* the truth of the sentence σ follows from that of the set of sentences Γ), $\Gamma \vdash \sigma$ (*i.e.* there is a deduction of σ from Γ). The sense of "completeness" in the Incompleteness Theorem, defined below, is a property of a set of sentences.

DEFINITION 15.2. A set of sentences Σ of a first-order language $\mathcal{L}$ is said to be *complete* if for every sentence τ either $\Sigma \vdash \tau$ or $\Sigma \vdash \neg\tau$.

That is, a set of sentences, or non-logical axioms, is complete if it suffices to prove or disprove every sentence of the langage in in question.

PROPOSITION 15.1. *A consistent set Σ of sentences of a first-order language $\mathcal{L}$ is complete if and only if the theory of Σ,*

$$\mathrm{Th}(\Sigma) = \{\, \tau \mid \tau \text{ is a sentence of } \mathcal{L} \text{ and } \Sigma \vdash \tau \,\},$$

is maximally consistent.

[1] Which, to confuse the issue, was also first proved by Kurt Gödel.

CHAPTER 16

Coding First-Order Logic

We will encode the symbols, formulas, and deductions of $\mathcal{L}_N$ as natural numbers in such a way that the operations necessary to manipulate these codes are recursive. Although we will do so just for $\mathcal{L}_N$, any countable first-order language can be coded in a similar way.

Gödel coding. The basic approach of the coding scheme we will use was devised by Gödel in the course of his proof of the Incompleteness Theorem.

DEFINITION 16.1. To each symbol s of $\mathcal{L}_N$ we assign an unique positive integer $\ulcorner s \urcorner$, the *Gödel code* of s, as follows:

(1) $\ulcorner (\urcorner = 1$ and $\ulcorner) \urcorner = 2$
(2) $\ulcorner \neg \urcorner = 3$ and $\ulcorner \to \urcorner = 4$
(3) $\ulcorner \forall \urcorner = 5$
(4) $\ulcorner = \urcorner = 6.$
(5) $\ulcorner v_k \urcorner = k + 12$
(6) $\ulcorner 0 \urcorner = 7$
(7) $\ulcorner S \urcorner = 8$
(8) $\ulcorner + \urcorner = 9$, $\ulcorner \cdot \urcorner = 10$, and $\ulcorner E \urcorner = 11$

Note that each positive integer is the Gödel code of one and only one symbol of $\mathcal{L}_N$. We will also need to code sequences of the symbols of $\mathcal{L}_N$, such as terms and formulas, as numbers, not to mention sequences of sequences of symbols of $\mathcal{L}_N$, such as deductions.

DEFINITION 16.2. Suppose $s_1 s_2 \dots s_k$ is a sequence of symbols of $\mathcal{L}_N$. Then the *Gödel code* of this sequence is

$$\ulcorner s_1 \dots s_k \urcorner = p_1^{\ulcorner s_1 \urcorner} \dots p_k^{\ulcorner s_k \urcorner},$$

where p_n is the nth prime number.

Similarly, if $\sigma_1 \sigma_2 \dots \sigma_\ell$ is a sequence of sequences of symbols of $\mathcal{L}_N$, then the *Gödel code* of this sequence is

$$\ulcorner \sigma_1 \dots \sigma_\ell \urcorner = p_1^{\ulcorner \sigma_1 \urcorner} \dots p_k^{\ulcorner \sigma_\ell \urcorner}.$$

EXAMPLE 16.1. The code of the formula $\forall v_1 = \cdot v_1 S0 v_1$ (the official form of $\forall v_1\, v_1 \cdot S0 = v_1$), $\ulcorner \forall v_1 = \cdot v_1 S0 v_1 \urcorner$, works out to

$$2^{\ulcorner\forall\urcorner} 3^{\ulcorner v_1 \urcorner} 5^{\ulcorner=\urcorner} 7^{\ulcorner\cdot\urcorner} 11^{\ulcorner v_1\urcorner} 13^{\ulcorner S\urcorner} 17^{\ulcorner 0\urcorner} 19^{\ulcorner v_1\urcorner} = 2^5 3^{13} 5^6 7^{10} 11^{13} 13^8 17^7 19^{13}$$

$$= 109425289274918632559342112641443058962750733001979829025245569500000\,.$$

This is *not* the most efficient conceivable coding scheme!

EXAMPLE 16.2. The code of the sequence of formulas

$= 00$	*i.e.* $0 = 0$
$(= 00 \rightarrow = S0S0)$	*i.e.* $0 = 0 \rightarrow S0 = S0$
$= S0S0$	*i.e.* $S0 = S0$

works out to

$$\begin{aligned} &2^{\ulcorner=00\urcorner} 3^{\ulcorner(=00\rightarrow=S0S0)\urcorner} 5^{\ulcorner=S0S0\urcorner} \\ &\quad = 2^{2^{\ulcorner=\urcorner} 3^{\ulcorner 0\urcorner} 5^{\ulcorner 0\urcorner}} \\ &\qquad \cdot 3^{2^{\ulcorner(\urcorner} 3^{\ulcorner=\urcorner} 5^{\ulcorner 0\urcorner} 7^{\ulcorner 0\urcorner} 11^{\ulcorner\rightarrow\urcorner} 13^{\ulcorner=\urcorner} 17^{\ulcorner S\urcorner} 19^{\ulcorner 0\urcorner} 23^{\ulcorner S\urcorner} 29^{\ulcorner 0\urcorner} 31^{\ulcorner)\urcorner}} \\ &\qquad \cdot 5^{2^{\ulcorner=\urcorner} 3^{\ulcorner S\urcorner} 5^{\ulcorner 0\urcorner} 7^{\ulcorner S\urcorner} 11^{\ulcorner 0\urcorner}} \\ &\quad = 2^{2^6 3^7 5^7} 3^{2^1 3^6 5^7 7^7 11^4 13^6 17^8 19^7 23^8 29^7 31^2} 5^{2^6 3^8 5^7 7^8 11^7}\,, \end{aligned}$$

which is large enough not to be worth the bother of working it out explicitly.

PROBLEM 16.1. *Pick a short sequence of short formulas of $\mathcal{L}_N$ and find the code of the sequence.*

A particular integer n may simultaneously be the Gödel code of a symbol, a sequence of symbols, and a sequence of sequences of symbols of $\mathcal{L}_N$. We shall rely on context to avoid confusion, but, with some more work, one could set things up so that no integer was the code of more than one kind of thing. In any case, we will be most interested in the cases where sequences of symbols are (official) terms or formulas and where sequences of sequences of symbols are sequences of (official) formulas. In these cases things are a little simpler.

PROBLEM 16.2. *Is there a natural number n which is simultaneously the code of a symbol of $\mathcal{L}_N$, the code of a formula of $\mathcal{L}_N$, and the code of a sequence of formulas of $\mathcal{L}_N$? If not, how many of these three things can a natural number be?*

Recursive operations on Gödel codes. We will need to know that various relations and functions which recognize and manipulate Gödel codes are recursive, and hence computable.

PROBLEM 16.3. *Show that each of the following relations is primitive recursive.*

(1) $\textsc{Term}(n) \iff n = \ulcorner t \urcorner$ *for some term* t *of* $\mathcal{L}_N$.
(2) $\textsc{Formula}(n) \iff n = \ulcorner \varphi \urcorner$ *for some formula* φ *of* $\mathcal{L}_N$.
(3) $\textsc{Sentence}(n) \iff n = \ulcorner \sigma \urcorner$ *for some sentence* σ *of* $\mathcal{L}_N$.
(4) $\textsc{Logical}(n) \iff n = \ulcorner \gamma \urcorner$ *for some logical axiom* γ *of* $\mathcal{L}_N$.

Using these relations as building blocks, we will develop relations and functions to handle deductions of $\mathcal{L}_N$. First, though, we need to make "a computable set of formulas" precise.

DEFINITION 16.3. A set Δ of formulas of $\mathcal{L}_N$ is said to be *recursive* if the set of Gödel codes of formulas of Δ,

$$\ulcorner \Delta \urcorner = \{ \ulcorner \delta \urcorner \mid \delta \in \Delta \},$$

is a recursive subset of $\mathbb{N}$ (*i.e.* a recursive 1-place relation). Similarly, Δ is said to be *recursively enumerable* if $\ulcorner \Delta \urcorner$ is recursively enumerable.

PROBLEM 16.4. *Suppose* Δ *is a recursive set of sentences of* $\mathcal{L}_N$. *Show that each of the following relations is recursive.*

(1) $\textsc{Premiss}_\Delta(n) \iff n = \ulcorner \beta \urcorner$ *for some formula* β *of* $\mathcal{L}_N$ *which is either a logical axiom or in* Δ.
(2) $\textsc{Formulas}(n) \iff n = \ulcorner \varphi_1 \dots \varphi_k \urcorner$ *for some sequence* $\varphi_1 \dots \varphi_k$ *of formulas of* $\mathcal{L}_N$.
(3) $\textsc{Inference}(n, i, j) \iff n = \ulcorner \varphi_1 \dots \varphi_k \urcorner$ *for some sequence* $\varphi_1 \dots \varphi_k$ *of formulas of* $\mathcal{L}_N$, $1 \le i, j \le k$, *and* φ_k *follows from* φ_i *and* φ_j *by Modus Ponens.*
(4) $\textsc{Deduction}_\Delta(n) \iff n = \ulcorner \varphi_1 \dots \varphi_k \urcorner$ *for a deduction* $\varphi_1 \dots \varphi_k$ *from* Δ *in* $\mathcal{L}_N$.
(5) $\textsc{Conclusion}_\Delta(n, m) \iff n = \ulcorner \varphi_1 \dots \varphi_k \urcorner$ *for a deduction* $\varphi_1 \dots \varphi_k$ *from* Δ *in* $\mathcal{L}_N$ *and* $m = \ulcorner \varphi_k \urcorner$.

If $\ulcorner \Delta \urcorner$ *is primitive recursive, which of these are primitive recursive?*

It is at this point that the connection between computability and completeness begins to appear.

THEOREM 16.5. *Suppose* Δ *is a recursive set of sentences of* $\mathcal{L}_N$. *Then* $\ulcorner \mathrm{Th}(\Delta) \urcorner$ *is*

(1) *recursively enumerable, and*
(2) *recursive if and only if* Δ *is complete.*

NOTE. It follows that if Δ is not complete, then $\ulcorner \mathrm{Th}(\Delta) \urcorner$ is an example of a recursively enumerable but not recursive set.

CHAPTER 17

Defining Recursive Functions In Arithmetic

The definitions and results in Chapter 17 let us use natural numbers and recursive functions to code and manipulate formulas of $\mathcal{L}_N$. We will also need complementary results that let us use terms and formulas of $\mathcal{L}_N$ to represent and manipulate natural numbers and recursive functions.

Axioms for basic arithmetic. We will define a set of non-logical axioms in $\mathcal{L}_N$ which prove enough about the operations of successor, addition, mutliplication, and exponentiation to let us define all the recursive functions using formulas of $\mathcal{L}_N$. The non-logical axioms in question essentially guarantee that basic arithmetic works properly.

DEFINITION 17.1. Let $\mathcal{A}$ be the following set of sentences of $\mathcal{L}_N$, written out in official form.

N1: $\forall v_0 \, (\neg = Sv_0 0)$
N2: $\forall v_0 \, ((\neg = v_0 0) \rightarrow (\neg \forall v_1 \, (\neg = Sv_1 v_0)))$
N3: $\forall v_0 \forall v_1 \, (= Sv_0 Sv_1 \rightarrow = v_0 v_1)$
N4: $\forall v_0 \; = +v_0 0 v_0$
N5: $\forall v_0 \forall v_1 \; = +v_0 Sv_1 S + v_0 v_1$
N6: $\forall v_0 \; = \cdot v_0 00$
N7: $\forall v_0 \forall v_1 \; = \cdot v_0 Sv_1 + \cdot v_0 v_1 v_0$
N8: $\forall v_0 \; = Ev_0 0S0$
N9: $\forall v_0 \forall v_1 \; = Ev_0 Sv_1 \cdot Ev_0 v_1 v_0$

Translated from the official forms, $\mathcal{A}$ consists of the following axioms about the natural numbers:

N1: For all n, $n + 1 \neq 0$.
N2: For all n, $n \neq 0$ there is a k such that $k + 1 = n$.
N3: For all n and k, $n + 1 = k + 1$ implies that $n = k$.
N4: For all n, $n + 0 = n$.
N5: For all n and k, $n + (k + 1) = (n + k) + 1$.
N6: For all n, $n \cdot 0 = 0$.
N7: For all n and k, $n \cdot (k + 1) = (n \cdot k) + n$.
N8: For all n, $n^0 = 1$.
N9: For all n and k, $n^{k+1} = (n^k) \cdot n$.

Each of the axioms in $\mathcal{A}$ is true of the natural numbers:

PROPOSITION 17.1. $\mathfrak{N} \models \mathcal{A}$, *where* $\mathfrak{N} = (\mathbb{N}, 0, \mathrm{S}, +, \cdot, \mathrm{E})$ *is the structure consisting of the natural numbers with the usual zero and the usual successor, addition, multiplication, and exponentiation operations.*

However, $\mathcal{A}$ is a long way from being able to prove all the sentences of first-order arithmetic true in $\mathfrak{N}$. For example, though we won't prove it, it turns out that $\mathcal{A}$ is not enough to ensure that induction works: that for every formula φ with at most the variable x free, if φ^x_0 and $\forall y\,(\varphi^x_y \to \varphi^x_{Sy})$ hold, then so does $\forall x\,\varphi$. On the other hand, neither $\mathcal{L}_N$ nor $\mathcal{A}$ are quite as minimal as they might be. For example, with some (considerable) extra effort one could do without E and define it from $\cdot$ and $+$.

Representing functions and relations. For convenience, we will adopt the following conventions. First, we will often abbreviate the term of $\mathcal{L}_N$ consisting of m Ss followed by 0 by S^m0. For example, S^30 abbreviates $SSS0$. The term S^m0 is a convenient name for the natural number m in the language $\mathcal{L}_N$ since the interpretation of S^m0 in $\mathfrak{N}$ is m:

LEMMA 17.2. *For every* $m \in \mathbb{N}$ *and every assignment* s *for* $\mathfrak{N}$, $\mathbf{s}(S^m0) = m$.

Second, if φ is a formula of $\mathcal{L}_N$ with all of its free variables among $v_1, \ldots, v_k$, and $m_0, m_1, \ldots, m_k$ are natural numbers, we will write $\varphi(S^{m_1}0, \ldots, S^{m_k}0)$ for the sentence $\varphi^{v_1 \ldots v_k}_{S^{m_1}0,\ldots,S^{m_k}0}$, *i.e.* φ with $S^{m_i}0$ substituted for every free occurrence of v_i. Since the term $S^{m_i}0$ involves no variables, it is substitutable for v_i in φ.

DEFINITION 17.2. Suppose Σ is a set of sentences of $\mathcal{L}_N$. A k-place function f is said to be *representable* in $\mathrm{Th}(\Sigma) = \{\, \tau \mid \Sigma \vdash \tau \,\}$ if there is a formula φ of $\mathcal{L}_N$ with at most $v_1, \ldots, v_k$, and v_{k+1} as free variables such that

$$\begin{aligned} f(n_1, \ldots, n_k) = m &\iff \varphi(S^{n_1}0, \ldots, S^{n_k}0, S^m0) \in \mathrm{Th}(\Sigma) \\ &\iff \Sigma \vdash \varphi(S^{n_1}0, \ldots, S^{n_k}0, S^m0) \end{aligned}$$

for all $n_1, \ldots, n_k$, and m in $\mathbb{N}$. The formula φ is said to *represent* f in $\mathrm{Th}(\Sigma)$.

We will use this definition mainly with $\Sigma = \mathcal{A}$.

EXAMPLE 17.1. The constant function c^1_3 given by $c^1_3(n) = 3$ is representable in $\mathrm{Th}(\mathcal{A})$; $v_2 = S^30$ is a formula representing it. Note

that that this formula has no free variable for the input of the 1-place function, but then the input is irrelevant...

To see that $v_2 = S^30$ really does represent c_3^1 in $\mathrm{Th}(\mathcal{A})$, we need to verify that

$$\begin{aligned} c_3^1(n) = m &\iff \mathcal{A} \vdash v_2 = S^30(S^n0, S^m0) \\ &\iff \mathcal{A} \vdash S^m0 = S^30 \end{aligned}$$

for all $n, m \in \mathbb{N}$.

In one direction, suppose that $c_3^1(n) = m$. Then, by the definition of c_3^1, we must have $m = 3$. Now

(1) $\forall x\, x = x \rightarrow S^30 = S^30$	A4
(2) $\forall x\, x = x$	A8
(3) $S^30 = S^30$	1,2 MP

is a deduction of $S^30 = S^30$ from $\mathcal{A}$. Hence if $c_3^1(n) = m$, then $\mathcal{A} \vdash S^m0 = S^30$.

In the other direction, suppose that $\mathcal{A} \vdash S^m0 = S^30$. Since $\mathfrak{N} \models \mathcal{A}$, it follows that $\mathfrak{N} \models S^m0 = S^30$. It follows from Lemma 17.2 that $m = 3$, so $c_3^1(n) = m$. Hence if $\mathcal{A} \vdash S^m0 = S^30$, then $c_3^1(n) = m$.

PROBLEM 17.3. *Show that the projection function π_2^3 can be represented in* $\mathrm{Th}(\mathcal{A})$.

DEFINITION 17.3. A k-place relation $P \subseteq \mathbb{N}^k$ is said to be *representable* in $\mathrm{Th}(\Sigma)$ if there is a formula ψ of $\mathcal{L}_N$ with at most $v_1, \ldots, v_k$ as free variables such that

$$\begin{aligned} P(n_1, \ldots, n_k) &\iff \psi(S^{n_1}0, \ldots, S^{n_k}0) \in \mathrm{Th}(\Sigma) \\ &\iff \Sigma \vdash \psi(S^{n_1}0, \ldots, S^{n_k}0) \end{aligned}$$

for all $n_1, \ldots, n_k$ in $\mathbb{N}$. The formula ψ is said to *represent* P in $\mathrm{Th}(\Sigma)$.

We will also use this definition mainly with $\Sigma = \mathcal{A}$.

EXAMPLE 17.2. Almost the same formula, $v_1 = S^30$, serves to represent the set — *i.e.* 1-place relation — $\{3\}$ in $\mathrm{Th}(\mathcal{A})$. Showing that $v_1 = S^30$ really does represent $\{3\}$ in $\mathrm{Th}(\mathcal{A})$ is virtually identical to the corresponding argument in Example 17.1.

PROBLEM 17.4. *Explain why $v_2 = SSS0$ does not represent the set $\{3\}$ in* $\mathrm{Th}(\mathcal{A})$ *and $v_1 = SSS0$ does not represent the constant function c_3^1 in* $\mathrm{Th}(\mathcal{A})$.

PROBLEM 17.5. *Show that the set of all even numbers can representable in* $\mathrm{Th}(\mathcal{A})$.

PROBLEM 17.6. *Show that the initial functions are representable in* $\mathrm{Th}(\mathcal{A})$*:*

(1) *The zero function* $\mathrm{O}(n) = 0$.
(2) *The successor function* $\mathrm{S}(n) = n + 1$.
(3) *For every positive* k *and* $i \leq k$*, the projection function* π_i^k.

It turns out that all recursive functions and relations are representable in $\mathrm{Th}(\mathcal{A})$.

PROPOSITION 17.7. *A* k*-place function* f *is representable in* $\mathrm{Th}(\mathcal{A})$ *if and only if the* $k+1$*-place relation* P_f *defined by*

$$P_f(n_1, \ldots, n_k, n_{k+1}) \iff f(n_1, \ldots, n_k) = n_{k+1}$$

is representable in $\mathrm{Th}(\mathcal{A})$.

Also, a relation $P \subseteq \mathbb{N}^k$ *is representable in* $\mathrm{Th}(\mathcal{A})$ *if and only if its characteristic function* χ_P *is representable in* $\mathrm{Th}(\mathcal{A})$.

PROPOSITION 17.8. *Suppose* $g_1, \ldots, g_m$ *are* k*-place functions and* h *is an* m*-place function, all of them representable in* $\mathrm{Th}(\mathcal{A})$*. Then* $f = h \circ (g_1, \ldots, g_m)$ *is a* k*-place function representable in* $\mathrm{Th}(\mathcal{A})$.

PROPOSITION 17.9. *Suppose* g *is a* $k+1$*-place regular function which is representable in* $\mathrm{Th}(\mathcal{A})$*. Then the unbounded minimalization of* g *is a* k*-place function representable in* $\mathrm{Th}(\mathcal{A})$.

Between them, the above results supply most of what is needed to conclude that all recursive functions and relations on the natural numbers are representable. The exception is showing that functions defined by primitive recursion from representable functions are also representable, which requires some additional effort. The basic problem is that it is not obvious how a formula defining a function can get at previous values of the function. To accomplish this, we will borrow a trick from Chapter 13.

PROBLEM 17.10. *Show that each of the following relations and functions (first defined in Problem 13.5) is representable in* $\mathrm{Th}(\mathcal{A})$.

(1) $\textsc{Div}(n, m) \iff n \mid m$
(2) $\textsc{IsPrime}(n) \iff n$ *is prime*
(3) $\textsc{Prime}(k) = p_k$*, where* $p_0 = 1$ *and* p_k *is the* k*th prime if* $k \geq 1$.
(4) $\textsc{Power}(n, m) = k$*, where* $k \geq 0$ *is maximal such that* $n^k \mid m$.
(5) $\textsc{Length}(n) = \ell$*, where* ℓ *is maximal such that* $p_\ell \mid n$.
(6) $\textsc{Element}(n, i) = n_i$*, where* $n = p_1^{n_1} \ldots p_k^{n_k}$ *(and* $n_i = 0$ *if* $i > k$*).*

Using the representable functions and relations given above, we can represent a "history function" of any representable function...

PROBLEM 17.11. *Suppose f is a k-place function representable in* $\mathrm{Th}(\mathcal{A})$. *Show that*

$$\begin{aligned} F(n_1, \ldots, n_k, m) &= p_1^{f(n_1,\ldots,n_k,0)} \ldots p_{m+1}^{f(n_1,\ldots,n_k,m)} \\ &= \prod_{i=0}^{m} p_i^{f(n_1,\ldots,n_k,i)} \end{aligned}$$

is also representable in $\mathrm{Th}(\mathcal{A})$.

. . . and use it!

PROPOSITION 17.12. *Suppose g is a k-place function and h is a $k+2$-place function, both representable in* $\mathrm{Th}(\mathcal{A})$. *Then the $k+1$-place function f defined by primitive recursion from g and h is also representable in* $\mathrm{Th}(\mathcal{A})$.

THEOREM 17.13. *Recursive functions are representable in* $\mathrm{Th}(\mathcal{A})$.

In particular, it follows that there are formulas of $\mathcal{L}_N$ representing each of the functions from Chapter 16 for manipulating the codes of formulas. This will permit us to construct formulas which encode assertions about terms, formulas, and deductions; we will ultimately prove the Incompleteness Theorem by showing there is a formula which codes its own unprovability.

Representability. We conclude with some more general facts about representability.

PROPOSITION 17.14. *Suppose Σ is a set of sentences of $\mathcal{L}_N$ and f is a k-place function which is representable in* $\mathrm{Th}(\Sigma)$. *Then Σ must be consistent.*

PROBLEM 17.15. *If Σ is a set of sentences of $\mathcal{L}_N$ and P is a k-place relation which is representable in* $\mathrm{Th}(\Sigma)$, *does Σ have to be consistent?*

PROPOSITION 17.16. *Suppose Σ and Γ are consistent sets of sentences of $\mathcal{L}_N$ and $\Sigma \vdash \Gamma$,* i.e. *$\Sigma \vdash \gamma$ for every $\gamma \in \Gamma$. Then every function and relation which is representable in* $\mathrm{Th}(\Gamma)$ *is representable in* $\mathrm{Th}(\Sigma)$.

This lets us use everything we can do with representability in $\mathrm{Th}(\mathcal{A})$ with any set of axioms in $\mathcal{L}_N$ that is at least as powerful as $\mathcal{A}$.

COROLLARY 17.17. *Functions and relations which representable in* $\mathrm{Th}(\mathcal{A})$ *are also representable in* $\mathrm{Th}(\Sigma)$, *for any consistent set of sentences Σ such that $\Sigma \vdash \mathcal{A}$.*

CHAPTER 18

The Incompleteness Theorem

The material in Chapter 16 effectively allows us to use recursive functions to manipulate coded formulas of $\mathcal{L}_N$, while the material in Chapter 17 allows us to represent recursive functions using formulas $\mathcal{L}_N$. Combining these techniques allows us to use formulas of $\mathcal{L}_N$ to refer to and manipulate codes of formulas of $\mathcal{L}_N$. This is the key to proving Gödel's Incompleteness Theorem and related results.

In particular, we will need to know one further trick about manipulating the codes of formulas recursively, that the operation of substituting (the code of) the term S^k0 into (the code of) a formula with one free variable is recursive.

PROBLEM 18.1. *Show that the function*

$$\textsc{Sub}(n,k) = \begin{cases} \ulcorner \varphi(S^k0) \urcorner & \text{if } n = \ulcorner \varphi \urcorner \text{ for a formula } \varphi \text{ of } \mathcal{L}_N \\ & \text{with at most } v_1 \text{ free} \\ 0 & \text{otherwise} \end{cases}$$

is recursive, and hence representable $\mathrm{Th}(\mathcal{A})$.

In order to combine the the results from Chapter 16 with those from Chapter 17, we will also need to know the following.

LEMMA 18.2. *$\mathcal{A}$ is a recursive set of sentences of $\mathcal{L}_N$.*

The First Incompleteness Theorem. The key result needed to prove the First Incompleteness Theorem (another will follow shortly!) is the following lemma. It asserts, in effect, that for any statement about (the code of) some sentence, there is a sentence σ which is true or false exactly when the statement is true or flase of (the code of) σ. This fact will allow us to show that the self-referential sentence we will need to verify the Incompleteness theorem exists.

LEMMA 18.3 (Fixed-Point Lemma). *Suppose φ is a formula of $\mathcal{L}_N$ with only v_1 as a free variable. Then there is a sentence σ of $\mathcal{L}_N$ such that*

$$\mathcal{A} \vdash \sigma \leftrightarrow \varphi(S^{\ulcorner \sigma \urcorner}0)\,.$$

Note that σ must be different from the sentence $\varphi(S^{\ulcorner\sigma\urcorner}0)$: there is no way to find a formula φ with one free variable and an integer k such that $\ulcorner\varphi(S^k0)\urcorner = k$. (Think about how Gödel codes are defined...)

With the Fixed-Point Lemma in hand, Gödel's First Incompleteness Theorem can be put away in fairly short order.

THEOREM 18.4 (Gödel's First Incompleteness Theorem). *Suppose Σ is a consistent recursive set of sentences of $\mathcal{L}_N$ such that $\Sigma \vdash \mathcal{A}$. Then Σ is not complete.*

That is, any consistent set of sentences which proves at least as much about the natural numbers as $\mathcal{A}$ does can't be both complete and recursive. The First Incompleteness Theorem has many variations, corollaries, and relatives, a few of which will be mentioned below. [**17**] is a good place to learn about more of them.

COROLLARY 18.5. (1) *Let Γ be a complete set of sentences of $\mathcal{L}_N$ such that $\Gamma \cup \mathcal{A}$ is consistent. Then Γ is not recursive.*

(2) *Let Δ be a recursive set of sentences such that $\Delta \cup \mathcal{A}$ is consistent. Then Δ is not complete.*

(3) *The theory of $\mathfrak{N}$,*

$$\mathrm{Th}(\mathfrak{N}) = \{\, \sigma \mid \sigma \text{ is a sentence of } \mathcal{L}_N \text{ and } \mathfrak{N} \models \sigma \,\},$$

is not recursive.

There is nothing really special about working in $\mathcal{L}_N$. The proof of Gödel's Incompleteness Theorem can be executed for any first order language and recursive set of axioms which allow one to code and prove enough facts about arithmetic. In particular, it can be done whenever the language and axioms are powerful enough — as in Zermelo-Fraenkel set theory, for example — to define the natural numbers and prove some modest facts about them.

The Second Incompleteness Theorem. Gödel also proved a strengthened version of the Incompleteness Theorem which asserts that, in particular, a consistent recursive set of sentences Σ of $\mathcal{L}_N$ cannot prove its own consistency. To get at it, we need to express the statement "Σ is consistent" in $\mathcal{L}_N$.

PROBLEM 18.6. *Suppose Σ is a recursive set of sentences of $\mathcal{L}_N$. Find a sentence of $\mathcal{L}_N$, which we'll denote by $\mathrm{Con}(\Sigma)$, such that Σ is consistent if and only if $\mathcal{A} \vdash \mathrm{Con}(\Sigma)$.*

THEOREM 18.7 (Gödel's Second Incompleteness Theorem). *Let Σ be a consistent recursive set of sentences of $\mathcal{L}_N$ such that $\Sigma \vdash \mathcal{A}$. Then $\Sigma \nvdash \mathrm{Con}(\Sigma)$.*

As with the First Incompleteness Theorem, the Second Incompleteness Theorem holds for any recursive set of sentences in a first-order language which allow one to code and prove enough facts about arithmetic. The perverse consequence of the Second Incompleteness Theorem is that only an inconsistent set of axioms can prove its own consistency.

Truth and definability. A close relative of the Incompleteness Theorem is the assertion that truth in $\mathfrak{N} = (\mathbb{N}, \mathrm{S}, +, \cdot, \mathrm{E}, 0)$ is not definable in $\mathfrak{N}$. To make sense of this, of course, we need to sort out what "truth" and "definable in $\mathfrak{N}$" mean here.

"Truth" means what it usually does in first-order logic: all we mean when we say that a sentence σ of $\mathcal{L}_N$ is true in $\mathfrak{N}$ is that when σ is true when interpreted as a statement about the natural numbers with the usual operations. That is, σ is true in $\mathfrak{N}$ exactly when $\mathfrak{N}$ satisfies σ, *i.e.* exactly when $\mathfrak{N} \models \sigma$.

"Definable in $\mathfrak{N}$" we do have to define...

DEFINITION 18.1. A k-place relation is *definable* in $\mathfrak{N}$ if there is a formula φ of $\mathcal{L}_N$ with at most $v_1, \ldots, v_k$ as free variables such that

$$P(n_1, \ldots, n_k) \iff \mathfrak{N} \models \varphi[s(v_1|n_1) \ldots (v_k|n_k)]$$

for every assignment s of $\mathfrak{N}$. The formula φ is said to *define* P in $\mathfrak{N}$.

A definition of "function definable in $\mathfrak{N}$" could be made in a similar way, of course. Definability is a close relative of representability:

PROPOSITION 18.8. *Suppose P is a k-place relation which is representable in* $\mathrm{Th}(\mathcal{A})$. *Then P is definable in $\mathfrak{N}$.*

PROBLEM 18.9. *Is the converse to Proposition 18.8 true?*

The question of whether truth in $\mathfrak{N}$ is definable is then the question of whether the set of Gödel codes of sentences of $\mathcal{L}_N$ true in $\mathfrak{N}$,

$$\ulcorner \mathrm{Th}(\mathfrak{N}) \urcorner = \{ \ulcorner \sigma \urcorner \mid \sigma \text{ is a sentence of } \mathcal{L}_N \text{ and } \mathfrak{N} \models \sigma \},$$

is definable in $\mathfrak{N}$. It isn't:

THEOREM 18.10 (Tarski's Undefinability Theorem). $\ulcorner \mathrm{Th}(\mathfrak{N}) \urcorner$ *is not definable in $\mathfrak{N}$.*

The implications. Gödel's Incompleteness Theorems have some serious consequences.

Since almost all of mathematics can be formalized in first-order logic, the First Incompleteness Theorem implies that there is no effective procedure that will find and prove all theorems. This might be considered as job security for research mathematicians.

The Second Incompleteness Theorem, on the other hand, implies that we can never be completely sure that any reasonable set of axioms is actually consistent unless we take a more powerful set of axioms on faith. It follows that one can never be completely sure — faith aside — that the theorems proved in mathematics are really true. This might be considered as job security for philosophers of mathematics.

We leave the question of who gets job security from Tarski's Undefinability Theorem to you, gentle reader...

Hints for Chapters 15–18

Hints for Chapter 15.

15.1. Compare Definition 15.2 with the definition of maximal consistency.

Hints for Chapter 16.

16.1. Do what is done in Example 16.2 for some other sequence of formulas.

16.2. You need to unwind Definitions 16.1 and 16.2, keeping in mind that you are dealing with formulas and sequences of formulas, not just arbitrary sequences of symbolsof $\mathcal{L}_N$ or sequences of sequences of symbols.

16.3. In each case, use Definitions 16.1 and 16.2, along with the appropriate definitions from first-order logic and the tools developed in Problems 13.3 and 13.5.

(1) Recall that in $\mathcal{L}_N$, a term is either a variable symbol, *i.e.* v_k for some k, the constant symbol 0, of the form St for some (shorter) term t, or $+t_1t_2$ for some (shorter) terms t_1 and t_2. $\chi_{\textsc{Term}}(n)$ needs to check the length of the sequence coded by n. If this is of length 1, it will need to check if the symbol coded is 0 or v_k for some k; otherwise, it needs to check if the sequence coded by n begins with an S or $+$, and then whether the rest of the sequence consists of one or two valid terms. Primitive recursion is likely to be necessary in the latter case if you can't figure out how to do it using the tools from Problems 13.3 and 13.5.

(2) This is similar to showing $\textsc{Term}(n)$ is primitive recursive. Recall that in $\mathcal{L}_N$, a formula is of the form either $= t_1t_2$ for some terms t_1 and t_2, $(\not\alpha)$ for some (shorter) formula α, $(\beta \to \gamma)$ for some (shorter) formulas β and γ, or $\forall v_i\, \delta$ for some variable symbol v_i and some (shorter) formula δ. $\chi_{\textsc{Formula}}(n)$ needs to check the first symbol of the sequence coded by n to identify which case ought to apply and then take it from there.

(3) Recall that a sentence is justa formula with no free variable; that is, every occurrence of a variable is in the scope of a quantifier.
(4) Each logical axiom is an instance of one of the schema A1–A8, or is a generalization thereof.

16.4. In each case, use Definitions 16.1 and 16.2, together with the appropriate definitions from first-order logic and the tools developed in Problems 13.5 and 16.3.

(1) $\ulcorner\Delta\urcorner$ is recursive and LOGICAL is primitive recursive, so...
(2) All $\chi_{\text{FORMULAS}}(n)$ has to do is check that every element of the sequence coded by n is the code of a formula, and FORMULA is already known to be primitive recursive.
(3) $\chi_{\text{INFERENCE}}(n)$ needs to check that n is the code of a sequence of formulas, with the additional property that either φ_i is $(\varphi_j \to \varphi_k)$ or φ_j is $(\varphi_i \to \varphi_k)$. Part of what goes into $\chi_{\text{FORMULA}}(n)$ may be handy for checking the additional property.
(4) Recall that a deduction from Δ is a sequence of formulas $\varphi_1 \dots \varphi_k$ where each formula is either a premiss or follows from preceding formulas by Modus Ponens.
(5) $\chi_{\text{CONCLUSION}}(n, m)$ needs to check that n is the code of a deduction and that m is the code of the last formula in that deduction.

They're all primitive recursive if $\ulcorner\Delta\urcorner$ is, by the way.

16.5. (1) Use unbounded minimalization and the relations in Problem 16.4 to define a function which, given n, returns the nth integer which codes an element of $\text{Th}(\Delta)$.
(2) If Δ is complete, then for any sentence σ, either $\lceil\sigma\rceil$ or $\lceil\neg\sigma$ must eventually turn up in an enumeration of $\ulcorner\text{Th}(\Delta)\urcorner$. The other direction is really just a matter of unwinding the definitions involved.

Hints for Chapter 17.

17.16. Every deduction from Γ can be replaced by a deduction of Σ with the same conclusion.

17.14. If Σ were insconsistent it would prove entirely too much...

17.6. (1) Adapt Example 17.1.
(2) Use the 1-place function symbol S of $\mathcal{L}_N$.
(3) There is much less to this part than meets the eye...

17.7. In each case, you need to use the given representing formula to define the one you need.

17.8. String together the formulas representing $g_1, \ldots, g_m$, and h with $\wedge$s and put some existential quantifiers in front.

17.9. First show that that $<$ is representable in $\mathrm{Th}(\mathcal{A})$ and then exploit this fact.

17.10.
(1) $n \mid m$ if and only if there is some k such that $n \cdot k = m$.
(2) n is prime if and only if there is no ℓ such that $\ell \mid n$ and $1 < \ell < n$.
(3) p_k is the first prime with exactly $k-1$ primes less than it.
(4) Note that k must be minimal such that $n^{k+1} \nmid m$.
(5) You'll need a couple of the previous parts.
(6) Ditto.

17.11. Problem 17.10 has most of the necessary ingredients needed here.

17.12. Problems 17.10 and 17.11 have most of the necessary ingredients between them.

17.13. Proceed by induction on the numbers of applications of composition, primitive recursion, and unbounded minimalization in the recursive definition f, using the previous results in Chapter 17 at the basis and induction steps.

Hints for Chapter 18.

18.2. $\mathcal{A}$ is a *finite* set of sentences.

18.1. First show that recognizing that a formula has at most v_1 as a free variable is recursive. The rest boils down to checking that substituting a term for a free variable is also recursive, which has already had to be done in the solutions to Problem 16.3.

18.3. Let ψ be the formula (with at most v_1, v_2, and v_3 free) which represents the function f of Problem 18.1 in $\mathrm{Th}(\mathcal{A})$. Then the formula $\forall v_3 \, (\psi^{v_2} v_1 \to \varphi^{v_1}_{v_3})$ has only one variable free, namely v_1, and is very close to being the sentence σ needed. To obtain σ you need to substitute $S^k O$ for a suitable k for v_1.

18.4. Try to prove this by contradiction. Observe first that if Σ is recursive, then $\ulcorner \mathrm{Th}(\Sigma) \urcorner$ is representable in $\mathrm{Th}(\mathcal{A})$.

18.5. (1) If Γ were recursive, you could get a contradiction to the Incompleteness Theorem.
(2) If Δ were complete, it couldn't also be recursive.
(3) Note that $\mathcal{A} \subset \mathrm{Th}(\mathfrak{N})$.

18.6. Modify the formula representing the function $\textsc{Conclusion}_\Sigma$ (defined in Problem 16.4) to get $\mathrm{Con}(\Sigma)$.

18.7. Try to do a proof by contradiction in three stages. First, find a formula φ (with just v_1 free) that represents "n is the code of a sentence which cannot be proven from Σ" and use the Fixed-Point Lemma to find a sentence τ such that $\Sigma \vdash \tau \leftrightarrow \varphi(S^{\ulcorner \tau \urcorner})$. Second, show that if Σ is consistent, then $\Sigma \nvdash \tau$. Third — the *hard* part — show that $\Sigma \vdash \mathrm{Con}(\Sigma) \rightarrow \varphi(S^{\ulcorner \tau \urcorner})$. This leads directly to a contradiction.

18.8. Note that $\mathfrak{N} \models \mathcal{A}$.

18.9. If the converse was true, $\mathcal{A}$ would run afoul of the (First) Incompleteness Theorem.

18.10. Suppose, by way of contradiction, that $\ulcorner \mathrm{Th}(\mathfrak{N}) \urcorner$ was definable in $\mathfrak{N}$. Now follow the proof of the (First) Incompleteness Theorem as closely as you can.

Appendices

APPENDIX A

A Little Set Theory

This apppendix is meant to provide an informal summary of the notation, definitions, and facts about sets needed in Chapters 1–9. For a proper introduction to elementary set theory, try [**8**] or [**10**].

DEFINITION A.1. Suppose X and Y are sets. Then

(1) $a \in X$ means that a is an *element* of (*i.e.* a thing in) the set X.
(2) X is a subset of Y, written as $X \subseteq Y$, if $a \in Y$ for every $a \in X$.
(3) The *union* of X and Y is $X \cup Y = \{\, a \mid a \in X \text{ or } a \in Y \,\}$.
(4) The *intersection* of X and Y is $X \cap Y = \{\, a \mid a \in X \text{ and } a \in Y \,\}$.
(5) The *complement of Y relative to X* is $X \setminus Y = \{\, a \mid a \in X \text{ and } a \notin Y \,\}$.
(6) The *cross product* of X and Y is $X \times Y = \{\, (a, b) \mid a \in X \text{ and } b \in Y \,\}$.
(7) The *power set* of X is $\mathcal{P}(X) = \{\, Z \mid Z \subseteq X \,\}$.
(8) $[X]^k = \{\, Z \mid Z \subseteq X \text{ and } |Z| = k \,\}$ is the set of subsets of X of size k.

If all the sets being dealt with are all subsets of some fixed set Z, the complement of Y, $\bar{Y}$, is usually taken to mean the complement of Y relative to Z. It may sometimes be necessary to take unions, intersections, and cross products of more than two sets.

DEFINITION A.2. Suppose A is a set and $\mathbf{X} = \{\, X_a \mid a \in A \,\}$ is a family of sets indexed by A. Then

(1) The union of $\mathbf{X}$ is the set $\bigcup \mathbf{X} = \{\, z \mid \exists a \in A \colon z \in X_a \,\}$.
(2) The intersection of $\mathbf{X}$ is the set $\bigcap \mathbf{X} = \{\, z \mid \forall a \in A \colon z \in X_a \,\}$.
(3) The cross product of $\mathbf{X}$ is the set of sequences (indexed by A) $\prod \mathbf{X} = \prod_{a \in A} X_a = \{\, (\, z_a \mid a \in A \,) \mid \forall a \in A \colon z_a \in X_a \,\}$.

We will denote the cross product of a set X with itself taken n times (*i.e.* the set of all sequences of length n of elements of X) by X^n.

DEFINITION A.3. If X is any set, a *k-place relation on X* is a subset $R \subseteq X^k$.

For example, the set $E = \{ 0, 2, 3, \dots \}$ of even natural numbers is a 1-place relation on $\mathbb{N}$, $D = \{ (x, y) \in \mathbb{N}^2 \mid x \text{ divides } y \}$ is a 2-place relation on $\mathbb{N}$, and $S = \{ (a, b, c) \in \mathbb{N}^3 \mid a + b = c \}$ is a 3-place relation on $\mathbb{N}$. 2-place relations are usually called binary relations.

DEFINITION A.4. A set X is *finite* if there is some $n \in \mathbb{N}$ such that X has n elements, and is *infinite* otherwise. X is *countable* if it is infinite and there is a 1-1 onto function $f : \mathbb{N} \to X$, and *uncountable* if it is infinite but not countable.

Various infinite sets occur frequently in mathematics, such as $\mathbb{N}$ (the natural numbers), $\mathbb{Q}$ (the rational numbers), and $\mathbb{R}$ (the real numbers). Many of these are uncountable, such as $\mathbb{R}$. The basic facts about countable sets needed to do the problems are the following.

PROPOSITION A.1.
(1) *If X is a countable set and $Y \subseteq X$, then Y is either finite or a countable.*
(2) *Suppose $\mathbf{X} = \{ X_n \mid n \in \mathbb{N} \}$ is a finite or countable family of sets such that each X_n is either finite or countable. Then $\bigcup \mathbf{X}$ is also finite or countable.*
(3) *If X is a non-empty finite or countable set, then X^n is finite or countable for each $n \geq 1$.*
(4) *If X is a non-empty finite or countable set, then the set of all finite sequences of elements of X, $X^{<\omega} = \bigcup_{n \in \mathbb{N}} X^n$ is countable.*

The properly sceptical reader will note that setting up propositional or first-order logic formally requires that we have some set theory in hand, but formalizing set theory itself requires one to have first-order logic.[1]

[1]Which came first, the chicken or the egg? Since, biblically speaking, "In the beginning was the Word", maybe we ought to plump for alphabetical order. Which begs the question: In which alphabet?

APPENDIX B

The Greek Alphabet

A	α		alpha
B	β		beta
Γ	γ		gamma
Δ	δ		delta
E	ϵ	ε	epsilon
Z	ζ		zeta
H	η		eta
Θ	θ	ϑ	theta
I	ι		iota
K	κ		kappa
Λ	λ		lambda
M	μ		mu
N	ν		nu
O	o		omicron
Ξ	ξ		xi
Π	π	ϖ	pi
P	ρ	ϱ	rho
Σ	σ	ς	sigma
T	τ		tau
Υ	υ		upsilon
Φ	ϕ	φ	phi
X	χ		chi
Ψ	ψ		psi
Ω	ω		omega

APPENDIX C

Logic Limericks

Deduction Theorem

A Theorem fine is Deduction,
For it allows work-reduction:
To show "A implies B",
Assume A and prove B;
Quite often a simpler production.

Generalization Theorem

When in premiss the variable's bound,
To get a "for all" without wound,
Generalization.
For civilization
Could use some help for reasoning sound.

Soundness Theorem

It's a critical logical creed:
Always check that it's safe to proceed.
To tell us deductions
Are truthful productions,
It's the Soundness of logic we need.

Completeness Theorem

The Completeness of logics is Gödel's.
'Tis advice for looking for mödels:
They're always existent
For statements consistent,
Most helpful for logical labörs.

APPENDIX D

GNU Free Documentation License

Version 1.2, November 2002

Copyright ©2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and

is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG.

Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section

all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them

all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic

equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Bibliography

[1] Jon Barwise (ed.), *Handbook of Mathematical Logic*, North Holland, Amsterdam, 1977, ISBN 0-7204-2285-X.

[2] J. Barwise and J. Etchemendy, *Language, Proof and Logic*, Seven Bridges Press, New York, 2000, ISBN 1-889119-08-3.

[3] Merrie Bergman, James Moor, and Jack Nelson, *The Logic Book*, Random House, NY, 1980, ISBN 0-394-32323-8.

[4] C.C. Chang and H.J. Keisler, *Model Theory*, third ed., North Holland, Amsterdam, 1990.

[5] Martin Davis, *Computability and Unsolvability*, McGraw-Hill, New York, 1958; Dover, New York, 1982, ISBN 0-486-61471-9.

[6] Martin Davis (ed.), *The Undecidable; Basic Papers On Undecidable Propositions, Unsolvable Problems And Computable Functions*, Raven Press, New York, 1965.

[7] Herbert B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[8] Paul R. Halmos, *Naive Set Theory*, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1974, ISBN 0-387-90092-6.

[9] Jean van Heijenoort, *From Frege to Gödel*, Harvard University Press, Cambridge, 1967, ISBN 0-674-32449-8.

[10] James M. Henle, *An Outline of Set Theory*, Problem Books in Mathematics, Springer-Verlag, New York, 1986, ISBN 0-387-96368-5.

[11] Douglas R. Hofstadter, *Gödel, Escher, Bach*, Random House, New York, 1979, ISBN 0-394-74502-7.

[12] Jerome Malitz, *Introduction to Mathematical Logic*, Springer-Verlag, New York, 1979, ISBN 0-387-90346-1.

[13] Yu.I. Manin, *A Course in Mathematical Logic*, Graduate Texts in Mathematics 53, Springer-Verlag, New York, 1977, ISBN 0-387-90243-0.

[14] Roger Penrose, *The Emperor's New Mind*, Oxford University Press, Oxford, 1989.

[15] Roger Penrose, *Shadows of the Mind*, Oxford University Press, Oxford, 1994, ISBN 0 09 958211 2.

[16] T. Rado, *On non-computable functions*, Bell System Tech. J. **41** (1962), 877–884.

[17] Raymond M. Smullyan, *Gödel's Incompleteness Theorems*, Oxford University Press, Oxford, 1992, ISBN 0-19-504672-2.

Index

(, 3, 24
), 3, 24
=, 24, 25
∩, 89, 133
∪, 89, 133
∃, 30
∀, 24, 25, 30
↔, 5, 30
∈, 133
∧, 5, 30, 89
¬P, 89
¬, 3, 24, 25, 89
∨, 5, 30, 89
⊨, 10, 35, 37, 38
⊭, 10, 36, 37
$\prod$, 133
⊢, 12, 43
\, 133
⊆, 133
×, 133
→, 3, 24, 25

$\mathcal{A}$, 117
A1, 11, 42
A2, 11, 42
A3, 11, 42
A4, 42
A5, 42
A6, 42
A7, 42
A8, 43
A_n, 3
Con(Σ), 124
⌜Δ⌝, 115
dom(f), 81
F, 7
$f\colon \mathbb{N}^k \to \mathbb{N}$, 81
$\varphi(S^{m_1}0, \ldots, S^{m_k}0)$, 118
φ_t^x, 42
$\mathcal{L}$, 24
$\mathcal{L}_1$, 26
$\mathcal{L}_=$, 26
$\mathcal{L}_F$, 26
$\mathcal{L}_G$, 53
$\mathcal{L}_N$, 26, 112
$\mathcal{L}_O$, 26
$\mathcal{L}_P$, 3
$\mathcal{L}_S$, 26
$\mathcal{L}_{NT}$, 25
$\mathfrak{M}$, 33
$\mathbb{N}$, 81, 134
$\mathfrak{N}$, 33, 112
N1, 117
N2, 117
N3, 117
N4, 117
N5, 117
N6, 117
N7, 117
N8, 117
N9, 117
$\mathbb{N}^k$, 81
$\mathbb{N}^k \setminus P$, 89
$\mathcal{P}$, 133
$P \cap Q$, 89
$P \cup Q$, 89
$P \wedge Q$, 89
$P \vee Q$, 89
π_i^k, 85, 120
$\mathbb{Q}$, 134
$\mathbb{R}$, 134
ran(f), 81
R_n, 55
$\mathcal{S}$, 6

S^m0, 118
T, 7
Th, 39, 45
Th(Σ), 112
Th($\mathfrak{N}$), 124
v_n, 24
X^n, 133
$[X]^k$, 133
$\bar{Y}$, 133

A, 90
α, 90
Code$_k$, 97, 106
Comp, 98
Comp$_M$, 96
Conclusion$_\Delta$, 115
Decode, 97, 106
Deduction$_\Delta$, 115
Diff, 83, 88
Div, 90, 120
Element, 90, 120
Entry, 96
Equal, 89
Exp, 88
Fact, 88
Formulas, 115
Formula, 115
$i_\mathbb{N}$, 82
Inference, 115
IsPrime, 90, 120
Length, 90, 120
Logical, 115
Mult, 88
O, 83, 85, 120
Power, 90, 120
Pred, 83, 88
Premiss$_\Delta$, 115
Prime, 90, 120
SIM, 106, 107
Sentence, 115
Sim, 98
Sim$_M$, 97
Step, 97, 106
Step$_M$, 96, 106
Subseq, 90
Sub, 123
Sum, 83, 87
S, 83, 85, 120
TapePosSeq, 96, 106
TapePos, 96, 105
Term, 115

abbreviations, 5, 30
Ackerman's Function, 90
all, x
alphabet, 75
and, x, 5
assignment, 7, 34, 35
  extended, 35
  truth, 7
atomic formulas, 3, 27
axiom, 11, 28, 39
  for basic arithmetic, 117
    N1, 117
    N2, 117
    N3, 117
    N4, 117
    N5, 117
    N6, 117
    N7, 117
    N8, 117
    N9, 117
  logical, 43
  schema, 11, 42
    A1, 11, 42
    A2, 11, 42
    A3, 11, 42
    A4, 42
    A5, 42
    A6, 42
    A7, 42
    A8, 43

blank cell, 67
blank tape, 67
bound variable, 29
bounded minimalization, 92
busy beaver competition, 83
  n-state entry, 83
  score in, 83

cell, 67
  blank, 67
  marked, 67
  scanned, 68
characteristic function, 82
chicken, 134
Church's Thesis, xi

clique, 55
code
  Gödel, 113
    of sequences, 113
    of symbols of $\mathcal{L}_N$, 113
  of a sequence of tape positions, 96
  of a tape position, 95
  of a Turing machine, 97
Compactness Theorem, 16, 51
  applications of, 53
complement, 133
complete set of sentences, 112
completeness, 112
Completeness Theorem, 16, 50, 137
composition, 85
computable
  function, 82
  set of formulas, 115
computation, 71
  partial, 71
connectives, 3, 4, 24
consistent, 15, 47
  maximally, 15, 48
constant, 24, 25, 31, 33, 35
constant function, 85
contradiction, 9, 38
convention
  common symbols, 25
  parentheses, 5, 30
countable, 134
crash, 70, 78
cross product, 133

decision problem, x
deduction, 12, 43
Deduction Theorem, 13, 44, 137
definable
  function, 125
  relation, 125
domain (of a function), 81

edge, 54
egg, 134
element, 133
elementary equivalence, 56
Entscheidungsproblem, x, 111
equality, 24, 25
equivalence
  elementary, 56
existential quantifier, 30
extension of a language, 30

finite, 134
first-order
  language for number theory, 112
  languages, 23
  logic, x, 23
Fixed-Point Lemma, 123
for all, 25
formula, 3, 27
  atomic, 3, 27
  unique readability, 6, 32
free variable, 29
function, 24, 31, 33, 35
  k-place, 24, 25, 81
  bounded minimalization of, 92
  composition of, 85
  computable, 82
  constant, 85
  definable in $\mathfrak{N}$, 125
  domain of, 81
  identity, 82
  initial, 85
  partial, 81
  primitive recursion of, 87
  primitive recursive, 88
  projection, 85
  recursive, x, 92
  regular, 92
  successor, 85
  Turing computable, 82
  unbounded minimalization of, 91
  zero, 85

Gödel code
  of sequences, 113
  of symbols of $\mathcal{L}_N$, 113
Gödel Incompleteness Theorem, 111
  First Incompleteness Theorem, 124
  Second Incompleteness Theorem, 124
generalization, 42
Generalization Theorem, 45, 137
On Constants, 45
gothic characters, 33
graph, 54
Greek characters, 3, 28, 135

halt, 70, 78

Halting Problem, 98
head, 67
  multiple, 75
  separate, 75

identity function, 82
if . . . then, x, 3, 25
if and only if, 5
implies, 10, 38
Incompleteness Theorem, 111
  Gödel's First, 124
  Gödel's Second, 124
inconsistent, 15, 47
independent set, 55
inference rule, 11
infinite, 134
Infinite Ramsey's Theorem, 55
infinitesimal, 57
initial function, 85
input tape, 71
intersection, 133
isomorphism of structures, 55

John, 134

k-place function, 81
k-place relation, 81

language, 26, 31
  extension of, 30
  first-order, 23
  first-order number theory, 112
  formal, ix
  natural, ix
  propositional, 3
limericks, 137
logic
  first-order, x, 23
mathematical, ix
natural deductive, ix
predicate, 3
propositional, x, 3
sentential, 3
logical axiom, 43

machine, 69
  Turing, xi, 67, 69
marked cell, 67
mathematical logic, ix
maximally consistent, 15, 48
metalanguage, 31
metatheorem, 31
minimalization
  bounded, 92
  unbounded, 91
model, 37
Modus Ponens, 11, 43
MP, 11, 43

natural deductive logic, ix
natural numbers, 81
non-standard model, 55, 57
  of the real numbers, 57
not, x, 3, 25
n-state
  Turing machine, 69
  entry in busy beaver competition, 83
number theory
  first-order language for, 112

or, x, 5
output tape, 71

parentheses, 3, 24
  conventions, 5, 30
  doing without, 4
partial
  computation, 71
  function, 81
position
  tape, 68
power set, 133
predicate, 24, 25
predicate logic, 3
premiss, 12, 43
primitive recursion, 87
primitive recursive
  function, 88
  recursive relation, 89
projection function, 85
proof, 12, 43
propositional logic, x, 3
proves, 12, 43
punctuation, 3, 25

quantifier
  existential, 30
  scope of, 30
  universal, 24, 25, 30

Ramsey number, 55
Ramsey's Theorem, 55
  Infinite, 55
range of a function, 81
r.e., 99
recursion primitive, 87
recursive
  function, 92
  functions, xi
  relation, 93
  set of formulas, 115
recursively enumerable, 99
  set of formulas, 115
regular function, 92
relation, 24, 31, 33
binary, 25, 134
characteristic function of, 82
definable in $\mathfrak{N}$, 125
k-place, 24, 25, 81, 133
primitive recursive, 89
recursive, 93
Turing computable, 93
represent (in Th(Σ))
  a function, 118
  a relation, 119
representable (in Th(Σ))
  function, 118
  relation, 119
rule of inference, 11, 43

satisfiable, 9, 37
satisfies, 9, 36, 37
scanned cell, 68
scanner, 67, 75
scope of a quantifier, 30
score
  in busy beaver competition, 83
sentence, 29
sentential logic, 3
sequence of tape positions
  code of, 96
set theory, 133
Soundness Theorem, 15, 47, 137
state, 68, 69
structure, 33
subformula, 6, 29
subgraph, 54
subset, 133
substitutable, 41
substitution, 41
successor
  function, 85
  tape position, 71
symbols, 3, 24
  logical, 24
  non-logical, 24

table
  of a Turing machine, 70
tape, 67
  blank, 67
  higher-dimensional, 75
  input, 71
  multiple, 75
  output, 71
  tape position, 68
    code of, 95
    code of a sequence of, 96
    successor, 71
  two-way infinite, 75, 78
Tarski's Undefinability Theorem, 125
tautology, 9, 38
term, 26, 31, 35
theorem, 31
theory, 39, 45
  of $\mathfrak{N}$, 124
  of a set of sentences, 112
there is, x
TM, 69
true in a structure, 37
truth
  assignment, 7
  in a structure, 36, 37
  table, 8, 9
  values, 7
Turing computable
  function, 82
  relation, 93
Turing machine, xi, 67, 69
  code of, 97
  crash, 70
  halt, 70
  n-state, 69
  table for, 70
  universal, 95, 97
two-way infinite tape, 75, 78

unary notation, 82

unbounded minimalization, 91
uncountable, 134
Undefinability Theorem, Tarski's, 125
union, 133
unique readability
  of formulas, 6, 32
  of terms, 32
Unique Readability Theorem, 6, 32
universal
  quantifier, 30
  Turing machine, 95, 97
universe (of a structure), 33
UTM, 95

variable, 24, 31, 34, 35
  bound, 29
  free, 29
vertex, 54

witnesses, 48
Word, 134

zero function, 85